Heuristic Strategies for Single Document Analysis

Thomas Bohne

Vollständiger Abdruck der von der Fakultät für Informatik der
Universität der Bundeswehr München zur Erlangung des akademischen Grades eines

Doktor-Ingenieurs (Dr.-Ing.)

genehmigten Dissertation.

Gutachter:
1. Prof. Dr. Uwe M. Borghoff
2. Prof. Dr.-Ing. Mark Minas

Die Dissertation wurde am 11. Mai 2015 bei der Universität der Bundeswehr
München eingereicht und durch die Fakultät für Informatik am 28. Mai 2015
angenommen. Die mündliche Prüfung fand am 21. September 2015 statt.

Kurzfassung

Das enorme Wachstum digitaler Textdaten bewirkt eine hohe Nachfrage nach automatischen Textanalysewerkzeugen zur Informationsgewinnung. Klartext beinhaltet ausreichend Informationen, um mit einem heuristischen Ansatz sinnvolle Schlüsselwörter extrahieren zu können. Texte in der Form von Dokumenten und Textdatenströmen weisen außerdem eine inhärente Struktur auf, die Informationen über ihren Inhalt enthält.

In dieser Arbeit werden zwei Ansätze zur Gewinnung von aussagekräftigen Informationen aus Einzeldokumenten entwickelt: Schlüsselwortextraktion und die Detektion von Strukturänderungen in Texten.

Eine Kombination mehrerer heuristischer Schlüsselwortextraktionsalgorithmen ist einzelnen Algorithmen überlegen und kann die Qualität der Resultate signifikant verbessern. Im ersten Teil meiner Arbeit vergleiche ich verschiedene Kombinationsmethoden und verwende die Hauptkomponentenanalyse als parameterfreie und effektive Methode zur Auswahl von optimalen Kandidaten für eine Kombination. Anschließend demonstriere ich den erfolgreichen Einsatz einer Kombination mit einem effizienten und flexiblen Schlüsselwortextraktionsalgorithmus. Dieser ist sprachunabhängig, schnell und benötigt kein Training. Die extrahierten Schlüsselwörter sind sinnvoll und er ist schneller als der bekannte Term Frequency - Inverse Document Frequency (TF-IDF)-Algorithmus.

Im zweiten Teil meiner Arbeit analysiere ich die Struktur von Textdokumenten und entwickle einen neuen Algorithmus zur Detektion von Strukturänderungen. Dieser Algorithmus identifiziert Veränderungen in der Zusammensetzung eines Textes. Er ist flexibel, sprachunabhängig und anwendbar auf Einzeldokumente und unbegrenzte Textdatenströme. Ich weise die Genauigkeit meines Ansatzes mit konkreten Beispielen nach und demonstriere seine überzeugende Leistung mit einem Vergleichsalgorithmus.

In einer Kollaboration habe ich eine Kombination von Schlüsselwortextraktionsmethoden in die praktische Anwendung CommunityMashup integriert. Das CommunityMashup ist eine Datenaggregationslösung für unterschiedliche Soziale Netzwerke. Mit der Extraktion von Schlüsselwörtern in nahezu Echtzeit sind wir in der Lage, neue Beziehungen zwischen Inhalten und Personen zu erzeugen und visualisieren diese mit einer interaktiven, plattformunabhängigen Lösung.

Abstract

The immense growth of digital text data evokes demand for automatic text analysis tools for information retrieval. A plain text provides sufficient information for a heuristic approach to identify meaningful keywords. Text as documents and text streams also feature an inherent structure that inform about their content.

In this thesis, two approaches for retrieval of meaningful information from single documents are developed: keyword extraction and the detection of structural changes in texts.

The combination of multiple heuristic keyword extraction algorithms is superior to individual methods, and can improve the quality of the results significantly. To further this idea in the first part of my thesis, I compare different combination methods and utilize Principal Component Analysis (PCA) as a parameter-free and effective method to determine optimal combination candidates. Then, I demonstrate the success of these methods with an efficient and flexible keyword extraction approach that is language-independent, fast, and does not require a training phase. The results of this algorithm are deemed meaningful, and its performance is superior to the well known TF-IDF.

In the second part of my thesis, I analyze the structure of text documents and develop a novel algorithm that detects structural changes. This algorithm identifies fluctuations in the composition of a text. It is flexible, language-independent, and performs on single documents as well as indefinite text streams. I demonstrate the accuracy of my approach using cogent real-world examples, and present its compelling performance with a benchmark algorithm.

As an application of my work, I implement a keyword extraction approach into the CommunityMashup in a collaboration. The CommunityMashup is a data aggregation solution for different social networks. With the extraction of keywords in almost real time, we are able to identify new relations between contents and people and visualize them with an interactive and platform-independent solution.

Acknowledgements

I would like to thank my supervisor Prof. Uwe M. Borghoff for his encouragement, strategic advice, and freedom to develop my own ideas. I am very grateful for the remarks that Prof. Mark Minas provided and the worthwhile discussions with him during numerous lunch breaks. The time at the Universität der Bundeswehr was characterized by an excellent working environment and very supportive colleagues: I would like to thank Peter Lachenmaier, Florian Ott, Sebastian Behrendt, Sebastian Rönnau, Sonja Maier, Nico Krebs, Prof. Lothar Schmitz, Volker Rönnau, Prof. Klaus Buchenrieder, Johannes Metscher, and Sebastian Müller. Franz Schmalhofer furthermore contributed unconventional insights and he is also a great climbing and skiing partner.

Prof. Jerzy Rozenblit continually supported since the beginning of my diploma thesis. He also gave me a very inspiring insight into the American research community. I also thank Prof. Nikolaus Ballenberger for his fruitful discussions about statistical procedures and the early morning ski tours.

I am grateful to the Bundeswehr for giving me the opportunity to spend three years of my career for the scientific work that led to this dissertation.

Thank you Judy for your time, your patience, and your loving support during all these years. Now we have time for adventures again. Finally, I thank my parents for supporting me with the care, consideration, and the love only parents can provide.

Legend

Symbols and Notations

Notation	Description
D	Document collection $D = d_1..d_N$
d	single document d
$\|d\|$	total number terms in d
$\|d\|_{avg}$	average document length
$d(t)$	number of documents containing term t
$w_1..w_N$	windows $w_1, ..., w_N$ comprising the document
w_t	window containing the term t
$\|w\|$	number of terms in w
$w(t)$	number of windows containing the term t
$\mathcal{W}(t)$	the term weight of term t
q	query q
p	probability for a success
$P(X)$	probability distribution of X
x_t	number of occurrences of term t
xn_t	normalized number of occurrences of term t
x_t^d	number of occurrences of term t in document d
x_t^w	number of occurrences of term t in window w
iwf	inverse window frequency
z_w	$z_w = F_w$ or $z_w = N_w$
N	number of windows / documents in the collection
X	random variable X with outcomes $x_1, ..., x_n$
$E(X)$	expected value of the random variable X
A	matrix with m rows and n columns
a_{ij}	the element in the i-th row and j-th column of A
$a_{i:}$	the i-th row vector of A
$a_{\cdot j}$	the j-th column vector of A
t_i	$\{t_0, t_1, ..., t_m\}$ series of term occurrences
σ^2	variance
μ	the mean
$\rho(t)$	ranking position of term t
$\delta_1..\delta_n$	dimensions $1..n$ of term weights

List of Figures

List of Tables

Contents

Chapter 1

Introduction

The Voynich manuscript is one of the most widely known undeciphered handwritten scripts. It is named after the Wilfried Voynich – a Polish book dealer – who purchased it in 1912. It contains the remaining 240 pages of text and illustrations. This “world’s most mysterious manuscript” was presumably written in the second half of the 15th century [137] but it remains to be deciphered. The text and the illustrations in the book remain a mystery. Just recently, Montemurro analyzed the word distribution and statistical structure of the text in the manuscript, and showed that the patterns within the text resemble those of real languages [94]. Is this proof that the manuscript is real and just not deciphered? Montemurro’s study clearly shows that a heuristic analysis can lead to insightful results in the domain of information retrieval.

With heuristics, it is possible to detect patterns in texts. Patterns may indicate information change or degree of informativeness. This idea is the fundamental basis for a number of keyword extraction approaches. Keywords are very useful in a number of scenarios: they provide an interpretation of the content of a document, enable a more precise search, or serve the purpose of indexing and linking. Thus, it follows that authors would readily provide keywords for their texts.

Nowadays, it is commonplace for authors of scientific papers to assign a number of keywords or key phrases to their publications. I also mention *key phrases* because the assigned keywords sometimes consist of more than one word. Key phrases typically consist of five to fifteen noun phrases. In Section 8.3 I show that author-assigned keywords or tags sometimes do not suitably represent the content of documents. This may be the case when they are assigned based on other – sometimes organizational or social – constraints. Other texts, such as online documents, may not be presented with any keywords at all. Automatic keyword extraction methods are useful for the reader in these cases.

This chapter is structured as follows: first I outline in Section 1.1 the context and in Section 1.2 the classification of my work. In Section 1.3, I provide an overview of my approaches and in Section 1.4 my scientific contributions, and finally in Section 1.5 an outline of my thesis.

1.1 Context of this Work

The methods presented in this thesis derive information from texts and can be categorized as text mining (also text data mining) algorithms. Although text mining is not a subcategory of data mining both fields are closely related and share a number of approaches [57]. The purpose of data mining – finding valuable patterns in data – is an obvious response to the growing amount of data being produced. Most data mining methods expect a highly structured format of data and therewith necessitate an extensive data preparation. Text mining methods prefer a document format such as Extensible Markup Language (XML). Some methods that are used resemble those of data mining as they transform textual data to numerical data, such as counting the words in texts. Although texts seem to consist of highly unstructured data, at a certain point of the process, the data gets transformed into a classical data-mining encoding and becomes structured.

Marti Hearst clarified in her paper different aspects of information retrieval and text data mining [57]. The goal of data mining is not to identify relevant information, but also to discover or derive new information from existing data or across datasets. Patterns such as part-of-speech tagging, word sense disambiguation, and bilingual dictionary creation already exist in the field of computational linguistics. Hearst noted that text mining includes an interactive cycle in which the user investigates hypotheses suggestion. Kroeze makes user interaction a precondition for the creation of *novel knowledge* [57; 72]. The concept of *data* refers to raw facts and numbers that – upon analysis and interpretation – may become *information*. In his work, Kroeze refers to *knowledge* as information combined with context and experience [72].

In this work, the focus is information retrieval rather than knowledge discovery. The presented methods do not require user interaction or lexical analysis of the texts.

The word "information" is used differently in various disciplines, ranging from Computer Science to Neurology and thus numerous definitions exist. Robert M. Losee suggests a discipline-independent definition of information that I adopt in this thesis:

Definition 1.1.1. *Information may be understood in a domain-independent way as the values within the outcome of any process [80].*

This definition designates an information carrying process to be the source of information. This process mays be a mathematical function as well as a complex phenomenon beyond human comprehension. In this work, I present two different carriers: one is a set of algorithms that are composed in different ways in order to identify keywords, and a second analyzes the structure of a text in order to identify changes of information value. Both approaches are based on a source text as the input. Their outputs are directly related to the input and, at best, carry meaningful information about the text. Therefore, Losee's definition of information suits the content of this work very well.

1.2 Classification of this Thesis

In the past, information acquisition from texts was performed in collaboration with domain experts, which was time consuming and costly. Two major text-based automatic information acquisition areas can be distinguished [148]: Information Retrieval (IR) and Information Extraction (IE). The IR techniques enable the recovery of relevant documents from a collection of documents. The task of IE aims to extract the most relevant content of the texts identified by IR. As the approaches in this thesis are applied to single documents, there is no distinction between relevant and irrelevant documents. Nevertheless, some of the algorithms aim to identify relevant sections (paragraphs or sentences) within a single document. The keyword extraction algorithms presented in this work perform IR tasks as well as IE procedures, whereas the trie-based change-point detection (CPD) algorithm presented in Chapter 7 is a classical IR task.

1.3 Goal and Approach

Single documents and also whole libraries such as the Bayerische Staatsbibliothek are digitalized [99; 2] and require IE methods to automatically recover the most relevant content from documents in very short time. Most of these IE methods rely on large databases, knowledge of the specific language to process, or a training phase. Texts can appear in different shapes, document formats, and structures. They can be static or a constantly changing data stream. In this work, I aim to extract the most valuable information from texts with nothing but the text itself. Whereas in the past, information acquisition from texts was performed with domain experts, the methods in this work guarantee maximal independence from languages, reference data, and autonomy from time consuming training processes.

Keyword Extraction

A number of heuristics have already proven to be extremely successful in IE to determine keywords or index terms for small document collections or texts [7; 149; 156]. These algorithms are based on specific properties of texts that help to determine the most informative terms and phrases. These properties were thoroughly analyzed throughout my work and thereafter I formalized them according to the task of single document IE. Furthermore, formal studies showed that the individual algorithms satisfy only a fraction of all preferable constraints of a successful algorithm. To compensate for this shortcoming, I test and evaluate different combination methods for keyword extraction algorithms. Finally I propose combined versions of these algorithms to increase their performance in different ways.

Structural Analysis

Documents and also text streams can be structured beyond the concept of frequency distributions. They may contain additional information, that can only be observed by analyzing their degree of order and disorder. I propose to adapt the concept of entropy for textual data and develop a novel algorithm that provides a relative measure for information value over a time series. My proposed algorithm is mapped by a dynamic trie structure. The algorithm should be language independent, fast, and be able to perform in a real-time scenario. The ultimate goal here is to be able to perform an analysis of online social networking and also news service that are constantly producing content.

Heuristic Analysis

It is in the nature of texts to differ in language, structure and content. A scientific text differs significantly from a message within a social networking service. Therefore, each text requires specific algorithms to extract the most meaningful information. The algorithms for keyword extraction as well as the detection method for structural information build a framework for this purpose. In this thesis I focus on the essence – the text – and aim to show that these algorithms extract meaningful information without many restrictions and adjustments. I expected them to work well but did not foresee that they would outperform well-known algorithms. I aim to provide use-case-scenarios that demonstrate their behavior. They are not required to outperform all available algorithms in every situation, nonetheless they come with a high degree of flexibility, independence, and speed.

1.4 Scientific Contributions

This thesis provides the following scientific contributions:

- A formal definition of retrieval constraints for single document analysis which allows for a characterization of some of the most prominent heuristics for single document analysis.
- I compare six different combination methods for the presented retrieval algorithms and interpret their behavior, and then propose a parameter-free, PCA-based method to determine an optimal selection of retrieval heuristics for combination. Furthermore, I demonstrate the compelling performance of one particular composed algorithm with real-world examples.
- A novel measure of information value that detects structural changes within a text or text stream. This algorithms is flexible, language independent, and fast.

  I demonstrate the behavior and the performance of all approaches with real world examples. Furthermore, the integration of my newly composed keyword extraction algorithm into a social media framework shows the applicability of a flexible, fast, and independent IR approach.

1.4.1 Research Methodology

The goal of this thesis is to identify the most useful properties of a plain text for IR and to use them to compose successful retrieval algorithms. The analysis of the most prominent retrieval algorithms and the definition of formal constraints for a successful retrieval algorithm lead to the insight that a composition of algorithms is desirable. Different composition methods exist but they haven't been evaluated and compared in this context. An analysis of different compositions was the logical consequence. During this process, I discovered that PCA-based analysis can identify candidates for a successful combination.

The detection of structural changes within texts is inspired by the concept of entropy. With the help of an entropy-based compression algorithm I am able to develop a measure of information change within texts. It is important to recognize that this measure is solely based on the input text and it may also be used for keyword extraction.

My aim here, is to provide a framework of techniques to identify information within plain texts without the use of large databases, user input, or large sets of parameters.

1.5 Outline

This thesis is divided into nine chapters and two appendices. The first three chapters introduce the most important concepts for single document IR and discuss related work. Chapter 4 to 5 contain the information retrieval algorithms used in this work and their combination methods. A novel algorithm that detects information structures in texts and text streams is described subsequently in chapters 6 and 7. The remaining chapters contain the evaluations and the conclusions of this thesis.

Chapter 1 classifies the research described in this thesis and puts the work into context. It highlights the scientific contributions and details the structure of this thesis. In Chapter 2 I provide the preliminary concepts for IR and IE for single documents and text streams. I also mention related concepts and recent developments. Chapter 3 discusses related work regarding the main approaches of this thesis. Chapter 4 introduces the concept of term weighting and applies the retrieval constraints of a successful IR algorithm to the scenario of single document analysis. The most important IR-models are described in this chapter. The proposed combination methods for the retrieval algorithms are described in Chapter 5. A second approach to text analysis based on structural information is described in Chapter 6. Chapter 7 describes the trie-based CPD algorithm and its implementation in detail. An exhaustive evaluation of the proposed keyword extraction algorithms and the novel information measure are presented in Chapter 8. The CommunityMashup represents a suitable use-case scenario for single document analysis in a fast changing environment and is presented in Chapter 9. A discussion of the results and possible future research are presented in Chapter 10. This chapter also contains the conclusions of this thesis.

Chapter 2

Information Retrieval for Single Documents

This chapter lays the foundations for the approaches presented in this work, and introduces related concepts and models that will be used in the remainder of this thesis. Additionally, a separate Chapter 3 introduces related approaches with reference to the main findings of my work.

Section 2.1 introduces certain mathematical methods necessary for the approaches presented in this thesis. The analysis of single documents and written texts requires a concept of linguistics and preprocessing for this work, which is described in Section 2.2. In Section 2.3 I present different segmentation approaches for single documents. The numerous approaches for text analysis in IR are based on a number of fundamental language models. A detailed comparison of the most important models is provided in Section 2.4.

2.1 Mathematical Background

In this section, I explain the basic mathematic principles essential for the understanding of the algorithms presented later on in this work. Statisticians perform *sampling* on a portion of a *population* in order to estimate the characteristics of the whole population. In the remainder I refer to a sample sequence X with its elements:

$$X = x_1, \ldots, x_N \tag{2.1}$$

The element $x_i, i \in 1..N$ refers to the ith value of the sequence X.

Certain properties of the sample set X can be calculated: *Mean* and *variance* are two basic parameters in statistics. Equation 2.2 defines the mean, μ:

$$\mu = \frac{1}{N} \sum_{i=1}^{N} x_i \tag{2.2}$$

Individual values may deviate from the mean, so that the spread of the values can be described with the *variance*, σ^2:

$$\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2 \tag{2.3}$$

The estimator above tends to underestimate σ for samples and is therefore replaced by the unbiased sample variance with the denominator $N-1$. As the variance is not calculated for all texts of a certain topic but only for a selection, the unbiased variance of the sample s^2 and its sample standard deviation s. The expected value $E(s^2)$ equals the variance σ^2 of X:

$$s^2 = \frac{\sum_{i=1}^{N}(x_i - \mu)^2}{N-1} \tag{2.4}$$

The standard deviation σ is simply the square root of the variance and shows how spread out the data is or how much it varies from the mean:

$$\sigma = \sqrt{E(s^2)} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2} \tag{2.5}$$

The *sample* standard deviation and *sample* variance are corrected (Bessel's correction) versions of the standard deviation and standard variance that compensate for systematic underestimation of variance in samples. The correction is performed by using $\frac{1}{N-1}$ in Equation 2.3 and 2.5 instead of $\frac{1}{N}$.

2.1.1 Smoothing

In statistics smoothing is used to reduce noise and change individual data so that important patterns can be captured more clearly. The analysis of the data in Section 8.6.2 demands some sort of smoothing in order to clearly identify seasonal patterns in the development of the trie structure that is monitored there. One of these patterns will lead to the determination of a change-point in the data. A common technique to perform a local averaging of a data stream is the *moving average* smoothing, where each element of a time series is replaced by the simple weighted average of n elements. The *median filter* can be used alternatively to means. Medians are frequently employed in image processing because they remove outliers from within the sequence of n data elements well. The basic concept of the median is to replace each element in the data stream with the median of the neighboring entries. The following example will demonstrate the median filter process:

```
median [1 2 3] = 2
median [7 3 4 6 8] = median [3 4 6 7 8] = 6
```

Another filter method is the *minimum filter* that basically extracts the minimum value from the sequence of elements:

```
min [1 2 3] = 1
min [7 3 4 6 8] = min [3 4 6 7 8] = 3
```

Both filters have been employed in the reference software that has been developed in the course of this work.

2.2 What is a Document?

About 5000 years ago, the Sumerians realized that the preservation of written information for future generations is critical for the efficient use of information [127]. This problem is more topical than ever before as the amount of information is growing tremendously and it is unknown in what form digital media will be available in 50 years from now [20].

Throughout history, text has been written on numerous surfaces in many forms all over the earth. Nowadays, it exists in either a printed or digital form as units of texts. The size of a unit is not always trivial to distinguish, but a general approach is to address a single unit of text as a *document* [7]. This section provides a definition of the most basic text units and introduces a series of preprocessing steps applied in this work.

2.2.1 Definition of Term, Word, Document

Generally, a document is a structured segment of text that appears in different forms such as traditional book, paper, article, more recently e-mail, web page, or source code. In this work, a document is denoted with the symbol d.

Words are the smallest syntactic units that can not be broken into smaller segments [70]. They can be classified in part of speech (POS) such as verbs, nouns, adverbs and used in their root forms as well as in modified forms. Words may not conform to the units that are used for text processing in information retrieval. Therefore, a *token* represents a single unit of text that may consist of a letter, a word, or any string of consecutive alphanumeric characters, whereas a *term* can be a single word, a word pair, a phrase within a document – I define single words as terms. A term is denoted with the symbol t in this work. Sometimes a term is called *token*, which is not entirely correct [93; 86]. *Tokens* are basically the elementary units that remain when the text is decomposed into smaller units by white spaces and special characters:

Input: “I am not omniscient, but I know a lot.” [155]

Output: $\boxed{\text{I}}$ $\boxed{\text{am}}$ $\boxed{\text{not}}$ $\boxed{\text{omniscient}}$ $\boxed{\text{but}}$ $\boxed{\text{I}}$ $\boxed{\text{know}}$ $\boxed{\text{a}}$ $\boxed{\text{lot}}$

The term-weight measures the importance of a term with respect to an assigned text unit. The relevance of a term coincides with the relevance of the term for the summary. A weighting algorithm assigns each extracted potential keyword a term weight. The term weight of term t is denoted as $\mathcal{W}(t)$. Finding an adequate weighting algorithm is the main challenge in keyword extraction. In the remainder of this work I refer to a term as a preprocessed word of the input text.

2.2.2 Preprocessing of Text

In this work, terms and characters represent the smallest fundamental units for further analysis. In an information retrieval process, text pre-processing is an essential functional part that has to be performed before the fundamental units are passed to the following processing steps [101]. Preprocessing can be referred to as *tokenization* or *normalization* in literature. It can be performed step-by-step according to the scheme in Figure 2.1.

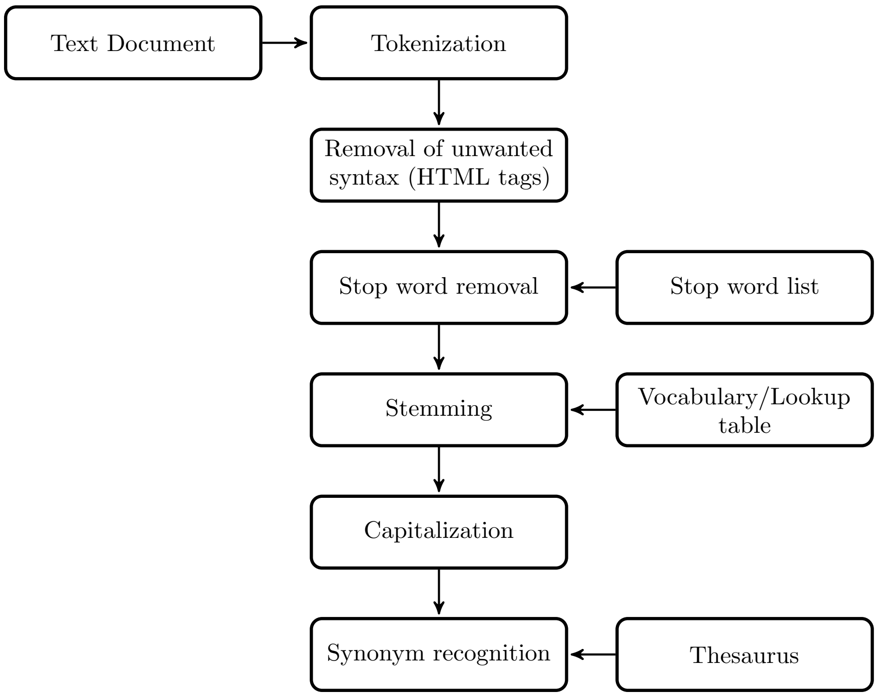


Figure 2.1: This scheme shows some of the most common preprocessing steps in a chronological sequence.

Some of the most common preprocessing steps and their dependencies are shown in Figure 2.1.

The removal of **special characters and unwanted syntax** is a language-specific and also a domain-specific task. In English texts, it makes sense to identify the term "parents" as equal in the following two cases:

"The parents eat out."

"The parents' dinner."

This example should probably be treated differently from words like "don't" or " McDonald's". In some domains, there are certain names that should be recognized, such as "C++", "C#" or "Jay-Z". When reading input from a HyperText Markup Language (HTML) page or XML file, the syntax has to be identified and processed, as well as certain character sequences like Uniform Resource Locator (URL)s, e-mail addresses, or numbers.

Stop words are words that appear in almost all documents of a document collection, providing limited information about the content. Usually high frequency words and function words such as conjunctions, prepositions, pronouns, etc. are identified as stop words and can be excluded from the list of potential keywords. Furthermore, stop words are content dependent, such as stop words for a web page differ heavily from stop words for emails or a news article [70].

Stop word removal aims to remove common words to prevent their identification as keywords and to reduce computation costs. The most common words in English based on the Oxford English corpus are "the, be, to, ..." [32]. Stop word lists can identify stop words in texts but they are language specific and usually not exhaustive. Furthermore, there exists no universal stop word list; each list has to be adjusted each time according to the input text, and the user runs into the risk of eliminating potential keywords that contain valuable information. For example, removing the most frequent English words from a text about the British rock band "The Who" or from a text about the song "Let It Be" by The Beatles is clearly not expedient. The trend in information retrieval systems changed from using large stop word lists (200-300 terms) to very small ones (7-13 terms) or even no stop word lists [86]. To remain language independent and to avoid removing potential keywords, I do not filter any stop words before the evaluation with the presented algorithms.

A *stem* is the root form of a word. Stems can be found in dictionaries though not all forms of a word need to be stored in a dictionary in order to identify a stem. The reason for **stemming** is to reduce a word to it's basic form and is very similar to **lemmatization**. Both preprocessing steps eliminate grammatical effects and derivations to increase potential matches [86]. An example of stemming is shown below:

"is", "are", "am" $\rightarrow$ "be"

"different", "difference", "differential" $\rightarrow$ "differ"

The example demonstrates a common problem that may occur with stemming: the words "different" and "differential" may be used in various contexts with a different meaning, such as "differential equation". The stemming result "differ" does not contain information about the context of the original word. With stemming the contextual information is lost.

Stemming basically cuts off the end of words whereas lemmatization uses a vocabulary and morphological analysis in order to remove inflectional endings. A *lemma* of a word is its base or dictionary form. The *Porter Stemmer* is a very popular stemmer for the English language [107] that is widely used.

Capitalization or case-folding can be performed to match terms in English that appear at the beginning of a sentence like "House" with terms that normally appear in lower case letters ("house"). This may also lead to problems with names such as "George W. Bush" or "US" for United States. Correct capitalization for only one language is challenging as it usually involves a sophisticated machine learning model.

Any one fact may be described with different words in different texts. **Synonym recognition** may help to identify factual similarities. It transforms words with the same meaning into a unique identifier using a thesaurus [22] and facilitates a subsequent analysis based on these common identifiers.

The pre-processing in this work includes removal of punctuation symbols, all relevant terms of the document are down-cased, and special characters are excluded. Since the examples in this work are based on the Latin alphabet, I filtered all special characters with a regular expression and excluded single characters from further processing. URLs, syntax, and e-mail addresses were filtered when necessary.

This pre-processing can actually be performed without knowing any details of the structure of the document. In addition, I exclude all terms with non-alphabetic characters as well as numbers because I empirically found that most numbers are used as page numbers or indices (e.g. in the appendix). Stemming and synonym recognition were not performed in order to stay language-independent and to conduct keyword extraction across several languages simultaneously. There is certainly room for improvement of the preprocessing steps in order to refine the results of the individual algorithms, but an extensive preprocessing is not the focus of this work.

2.3 Segmentation of Documents

The structure of written texts can be described as a sequence of subtopics that comprise a few main topics [56]. Sometimes these subtopics are noted by subheadings or separated in paragraphs. Topics and subtopics in a document are typically defined by the special words that characterize the document [112]. For example, articles about computer security would contain a lot of occurrences of words like: “phishing“, “intruder”, “malware”, “virus”. Topics can be inferred by a number of different models – topic models – that uncover the underlying semantic structure within a document (collection) [157; 35]. For example, the State of The Union Address of the President of the United States (US) in 2012 contains the special words: “Americans, United States, country, citizens”, whereas the subtopics of this document are characterized by words like “Iraq, war, troops”, or “jobs, manufacturers, hiring”.

The DARPA Topic Detection and Tracking (TDT) initiative addresses event detection in document streams and aims to group stories that arrive over time into *single-topic clusters*. Documents with temporal order (e.g. news stories) are therefore considered streams that are applicable to event detection. The developed methods aim to detect indications for new events. Topic Detection (TD) is a fundamental research question of TDT and different topic models can be obtained from a number of methods, such as *manual thesauri, term clustering, document clustering, Latent Semantic Analysis (LSA), relevance feedback* [54; 157; 3; 35].

Some of the presented information retrieval algorithms demand segmentation of documents in order to return meaningful results. Passage retrieval aims to retrieve text excerpts of a document that are relevant to a user query [121]. One could assume that passage retrieval algorithms can simply segment a document and return short passages in response to a user query [65; 170]. However, they actually require query terms and are not capable of determining the topic of a passage without these terms [90]. In fact, segmentation of a single document is a very challenging tasks, often not even human readers agree upon clear topic boundaries [56].

Most IR processes don’t address the segmentation issue because they focus on a document collection. They preprocess the contained documents and perform the retrieval process on a single document with respect to the whole collection. In this work, I partition a single document d into a number of windows N using different criteria. The retrieval is then performed on the individual windows with respect to the single document. The windows $w_1, \dots, w_N$ consist of terms, are non-overlapping, and contain all terms in d. All the segmentation approaches have to be performed before preprocessing.

An optimal segmentation algorithm would be able to identify all subtopics and partition a document into $N =$ *number of subtopics* windows, that comprise a single subtopic each. Words that describe a subtopic appear in close proximity to each other, whereas words that describe the main topics distribute across the entire document [58]. This information can be useful for term weighting. Accordingly, a well performed segmentation process may contribute to better keywords as the outcome. A book can be split into windows of constant length or alternatively the author-defined chapters can form windows of different size [114]. I propose the following three different criteria to determine the optimal window composition:

- Number of terms
- Logical document structure
- Number of extracted keywords

The most straightforward approach defines the size of a window as a fixed *number of terms* n that the window contains. This can be realized by setting $|w| = n$ beforehand and splitting the document into $\frac{|d|}{n} + 1$ windows, whereas the last window might contain less than n terms. Alternatively, I chose a segmentation into $\lceil \frac{|d|}{n} \rceil$ equally sized windows. This method is simple, fast, but does not split the document into subtopic-containing windows. An example is shown in Table 2.1.

Window 1	Window 2	Window 3
“Imagine what we”	“could accomplish if”	“we followed their ”

Table 2.1: The *number of terms* $n = 3$ determines the window size in this text sample.

With a varying window size, the number of terms in a window changes. Consequently, the risk of content-overlapping windows or content-splitting windows rises with increasing window size. I propose the use of document structure information to reduce that risk. Furthermore, I presume that there is no external corpus of documents available that could provide useful domain-specific information for the keyword extraction process.

In principle, every syntactically correct text contains sentences. I took advantage of the sentence punctuation and mark sentences as single units of the document. Instead of varying the window size by single terms, I increase or decrease the windows by single sentences to extract the claimed number of keywords (see Section 8.3.2). Table 2.2 shows an example of the sentence-based segmentation [135]. A similar segmentation approach creates windows that contain paragraph-sized units. A well-written text is supposed to contain a summary sentence at the beginning and / or at the end of each paragraph. Unfortunately these expectations are often not met in the real world. Paragraph markings may only be inserted by the author of the text in order to make it easier to read or just because of a page break [134]. During my experiments, the sentence and paragraph segmentation approaches contributed to good results but they were not the main scope of this work [19].

Window 1	Window 2	Window 3
“They focus on the mission at hand.”	“They work together.”	“Imagine what we could accomplish if we followed their example.”

Table 2.2: The document structure – here: sentences – form windows of different size.

Despite their usefulness, *linguistic features* are language-specific and domain-dependent. A scientific article contains an abstract at the beginning and a conclusion at the end. On the other hand, a news story or Web blog might not conform with that structure. Text itself can be classified in structured and semi-structured text. Web pages usually follow a very rigid format, such as the weather forecast on a news site. Semistructured text consists of sentences with fragments containing information that is following some kind of order [148]. Event announcements or restaurant menus usually show this kind of structure.

A more reliable domain- and language-independent source of additional information is the document itself. Documents contain many types of structured data. The structures of a document can be used to obtain maximum information for the keyword extraction process. I illustrate the usefulness of document structure with the following example:

The popularity of XML files rose significantly in the last years as they also contribute to the Open Document Format for Office Applications (ODF) document format. XML documents can contain meta-data, style-sheets and they also include content relevant information. XML content data is stored in leaves in the XML tree. Rönnau and Borghoff showed that the lowest two levels of the XML tree contain over 80% of all nodes [119]. Since text nodes are often just leaf nodes of the parent nodes, I propose a windowing technique that takes into account the tree structure of the XML document.

Another common approach is the *sliding window technique*, where a sliding window moves across the text. This approach uses overlapping windows – shared areas between the windows exist [90]. An example is shown in Table 2.3. As the window reads new terms, other terms are dropped. This technique includes three major steps [14]:

1. Insertion – a new run for each incoming term
2. Term reservation until the lifetime of the message expires
3. Deletion operation, where the term is deleted

Since this technique would require an analysis after each insertion and the analysis would be performed for each term multiple times, I did not consider

Window 1	Window 2	Window 3
`Imagine what we`	`what we could`	`we could accom-plish`

Table 2.3: The window of size $|w| = 3$ slides across the text.

this procedure for the term weighting algorithms described in Chapter 4, but rather for the trie-based CPD in Chapter 7.

Most keyword extraction approaches produce a variable number of keywords depending on the positive selection criterion. In Section 8.3 I empirically show that the number of extracted keywords strongly depends on the size of the windows that compose the underlying document. While experimenting with different window sizes, I used the number of the extracted keywords as a criterion for the optimal window size and developed the *self-regulation approach* that I describe thoroughly in Section 8.3.2.

2.4 Information Retrieval Models

Models are commonly used in science to simplify real-world scenarios or objects to make them more easily ascertainable for humans or to identify specific features. In IR, document models in particular associate the problem of retrieval with the issue of language model (LM) estimation [163]. The IR models account for specific features of a document. Therefore, I selected four very popular but also very diverse IR and language models for this work, which I introduce in this section. This selection is certainly not exhaustive as I do not consider the *logic-based models* or Latent Semantic Indexing (LSI) in this context. The selected models support a fast performing, language-independent algorithm that can perform on the text itself without user input and reference data.

2.4.1 The Bag-of-Words Model

The bag-of-words model is one of the most widely used document models in IR. The bag-of-words model of a document is like a mathematical multiset that allows duplicates containing the terms of the document as elements. It is a simplified representation of a text that does not take into account word order and grammatical structure. Consequentially, the bag $\{a, a, b, c, c, c\}$ and $\{a, b, c, a, c, c\}$ are equivalent considering that model. Most of the frequency-based IR algorithms are based on a bag-of-words document representation. One of the most popular of the frequency-based term-weighting algorithms is TF-IDF that I describe in Section 4.3.3. Consider the following two text snippets (windows):

Window 1: “Mike likes to run in the mountains. Adam likes to run too.”

Window 2: “Mike also likes to run flat.”

The terms of the two windows were preprocessed (here: down-cased) and the *term frequency* was put into the *term-window matrix* in Table 2.4. The word order is ignored in the matrix. Hence the matrix is not able to differentiate if it is “mike” or “adam” who “likes to run in the mountains.” It appears that windows with similarities in the term-window-matrix have a similar content [86].

Term	Window 1	Window 2
`mike`	1	1
`likes`	2	1
`to`	2	1
`run`	2	1
`in`	1	0
`the`	1	0
`mountains`	1	0
`adam`	1	0
`too`	1	0
`also`	0	1
`flat`	0	1

Table 2.4: This example is based on the Bag-of-Words model.

This representation used to represent the term frequency of the terms of the windows in a document is commonly used to represent the relative importance of terms of a document within a document collection and is denoted as *Vector Space Model (VSM)*.

2.4.2 Vector Space Model (VSM)

Salton et al. applied the VSM to information retrieval in 1975 [120]. They propose a window space where each window is represented by a vector. The dimension of the vector equals the number of different terms in the document. A vector represents the corresponding window as a bag-of-words column in a *term-window matrix* (Section 2.4.1). Each element of the matrix may be weighted according to its importance or assigned the frequency of the term's appearance in the corresponding window. The rows in the term-window matrix correspond to the *unique* terms in the document. The VSM is used for clustering, classification, or scoring windows (documents) on a query, whereas the query terms are treated as a pseudo window [86]. A measure of similarity is usually the cosine between two vectors in the vector space.

Let A be a matrix of n windows containing m different terms:

$$A = \begin{pmatrix} a_{11} & \dots & a_{1j} & \dots & a_{1n} \\ \vdots & & \vdots & & \vdots \\ a_{i1} & \dots & a_{ij} & \dots & a_{in} \\ \vdots & & \vdots & & \vdots \\ a_{m1} & \dots & a_{mj} & \dots & a_{mn} \end{pmatrix}$$

If Table 2.4 would be represented by A, it would contain 10 rows and 2 columns ($n = 2$ and $m = 10$). In general, if a term t_i is chosen randomly in a window w_j the value at position a_{ij} is zero because A is sparse. The vector $a_{:j}$ indicates the column vector of A and $a_{i:}$ the row vector of A. While the vector $a_{:j}$ contains the signature of the j-th window, $a_{i:}$ contains the signature of the i-th term [150]. According to this vector-based document model a query is is usually represented by a boolean query vector $\vec{q}$, containing true for the terms in the query and false otherwise.

The downsides of VSMs are the high dimensionality, the sparseness that comes with the large number of terms, and the lack of structure. A significant approach in IR that is based on the concept of VSM is LSA. LSA aims to overcome the high dimensionality of VSM by reducing the dimensions of the matrix in order to infer contextual usage of the terms and identify latent semantic relations [75]. This basically means, that the meaning of words is closely related the statistics of their usage. In 2010 Turney wrote an overview paper about VSMs, stating that they are a highly successful approach to semantics [150]. In this thesis, the VSM will be of great use for the analysis of the concepts of term weighting algorithms in Chapter 5. It is a basis for the described combination approaches.

2.4.3 The Probabilistic Language Model

There exist two probabilistic approaches that are mainly used in IR. A user that aims to find a section of text thinks of words that would describe this section and expresses a query – which is one or more words. The **Binary Independence Model (BIM)** aims to estimate the probability value $p(R|w, q)$ – the conditional probability of a window w being relevant (R) to a query q [115]. Imagine this user-generated query, containing n words, with $n \geq 1, n \in \mathcal{N}$. The windows in the document that are relevant to the query compose the *ideal* answer set [7]. So far it is unknown what *relevant* means. The meaning of *relevant* has to be guessed and is the reason for the probability $p(R|w, q)$. This allows the creation of a probabilistic answer set. The windows with the highest probability are assumed to answer the query best.

This model is closely related to the VSM as the windows in the document and the query q are represented by binary (boolean) vectors $\vec{x} = (x_1, ..., x_n)$ [162]. The value of x_i equals 1, if t_i is present in w, 0 otherwise; the query vector $\vec{q}$ is represented accordingly. Similar to the VSM, all occurring terms are represented independently in this model. There is also no relation between the windows, and duplicate windows would not be recognized. Additionally, two different windows may have an equal vector $\vec{x}$. This retrieval model is rather simple but has shown satisfying results and is widely used [86].

Besides the BIM, there is another probabilistic model, which refers to a probability distribution capturing the statistical regularities of the generating language [104]. This language model is the foundation for some of the most successful IR algorithms, that are analyzed in Section 4.3. In speech recognition, a LM would predict the probability of the next word. This principle can also be applied to IR. Consequently, Ponte and Croft state, that LMs assign probabilities to a single term (unigram) or a sequence of terms (n-gram). Then, a maximum likelihood estimate calculates which window $w_1, .., w_n$ in a single document d would most likely generate the query q: $p(q|M_w)$. The probability of a query being generated by a window M_w of the LM is denoted as $p(q|M_w)$. A relevant window basically maximizes the probability $p(q|M_w)$.

The *unigram model* is the most straightforward way to estimate the probability of a sequence of terms. Figure 2.2 shows the language being modeled as a finite

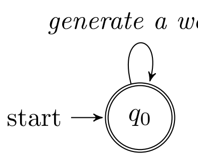


Figure 2.2: The finite single-state automaton representing the unigram language model.

single-state automaton producing a probability distribution for all the terms of the language $\mathcal{A}$ so that $\sum_{t \in \mathcal{A}} p(t) = 1$. This model can determine the probability for any sequence of any length as the single state which is also the end state of the automaton. Equation 2.6 shows that the probability of sequence $t_1 t_2 t_3$ equals the probability of the product of the probabilities of the individual terms.

$$p(t_1 t_2 t_3) = p(t_1) \cdot p(t_2|t_1) \cdot p(t_3|t_1 t_2) \overset{(unigram)}{=} p(t_1) \cdot p(t_2) \cdot p(t_3) \tag{2.6}$$

The complexity of the unigram model is quite low as it does not recognize any association between terms. It clearly does not model language correctly. It is an assumption that states: The probability of the terms t_1, t_2, t_3 appearing as a sequence is equal to the product of the terms appearing individually. It can be regarded as a first order approximation. Apparently, this assumption seems to suffice in most cases as it is widely used for IR applications and it generates good results [86]. This model totally neglects the influence of other words. It is basically a probabilistic version of the bag-of-words model. Considering the order of a sequence of terms, the n-gram model is a much better solution. The probability of the sequence $t_1 t_2 ... t_n$ is then approximated as follows:

$$p(t_1 t_2 ... t_n) = p(t_1) \cdot p(t_2|t_1) ... p(t_n|t_1 t_2 ... t_{n-1}) = \prod_{i=1}^{n} (p_i|p_1 ... p_{i-1})$$

The probability of the i-th term depends on the probability of the preceding sequence of the $i - 1$ terms (also called the *context)* which leads to an $(i - 1)$-th order Markov chain [33]. Due to the complexity of higher order Markov chains, the less sophisticated bigram and trigram models are preferred in most higher order approximations [91; 86].

In this work, I do not focus on the probability of a query fitting a window, but rather on the relevance of terms inside this window – the potential keywords. In order to extract keywords from a single document, the probability of a potential keyword term t within a document has to be translated into a measure of information. Which terms are relevant for the user? How much information does t provide to the content of the document? How are these probabilities determined? I will describe the most successful probability estimates and two possible approaches to determine the information value of potential keywords in Section 4.3.

2.4.4 Divergence from Randomness Model

Amati and Rijsbergen propose a probabilistic model that can also be interpreted as a language model [5]. Their basic concept states that terms occur in a document with a certain probability, that can be modeled with a probabilistic model. The term weights of the individual terms are then determined by measuring the difference between the probability process and the actual term distribution – *divergence from randomness*. This model is based on the following two components:

1. The essential assumption is: Terms that are randomly distributed across the entire document convey little information. The probability of the term t in the document d according to the chosen model of randomness is defined by $p_1(t|d)$. In his work, Amati provides a set of the most successful fundamental probability models to determine $p_1(t|d)$ [4]. A small probability $p_1(t|d)$ means a sparse distribution of t and therefore a high *informative content* of t:

   $$inf_1 = -\log_2 p_1(t|d) \tag{2.7}$$

   This first assumption closely resembles the Helmholtz principle in computer vision (see Section 4.3.4).

2. The second component of the model considers only the windows that contain the term t. According to the definition of Amati et al., this subset is referred to as the *elite set of windows* [5]. The second probability $p_2(t|w_t)$ is then determined with respect to the windows w_t containing the term t. If there has not been an appearance of t in a while and suddenly t occurs once, the expectation to find more occurrences of t rises [5]. It rises even more if more occurrences appear. This phenomenon is called an apparent *aftereffect of future sampling* and is very similar to the notion of burstiness [43]. The lower the expectation of t with respect to the elite set of windows is, the higher is the informative content that this term contains:

   $$inf_2 = 1 - p_2(t|w_t) \tag{2.8}$$

Amati et al. proposed a term weighting function that combines the two probabilities p_1 and p_2 as a product of the informative content and the apparent aftereffect of future sampling [5]:

$$\mathcal{W} = inf_1 \cdot inf_2 = -\log_2 p_1(t|d) \cdot (1 - p_2(t|w_t)) \tag{2.9}$$

The final term weight $\mathcal{W}$ is related to the whole document d as well as to the elite set of windows w_t. I utilize this this model in Chapter 5 to combine term weights of different IR approaches that represent the concepts of inf_1 and inf_2 in order to create more successful retrieval algorithms.

Nevertheless, this model does have some shortcomings. It does not take into account the length of the windows; therefore, normalization is necessary to avoid the effect of a higher weight of a term due to a larger number of occurrences in a long document. Normalization techniques are described in the next section.

2.4.5 Length Normalization

If the segmentation procedure for a document is not based on the number of terms comprising the window (see Section 2.3), the windows may be of different size. The term frequency of a term t in window w is defined as the number of appearances of t in w: $x_t^{w_1}$. In order to illustrate length normalization, let the term frequency of t in w_1 be $x_t^{w_1} = 5$ and the term frequency $x_t^{w_2} = 5$. If $|w_1| << |w_2|$, the term frequency $x_t^{w_1}$ should be accounted differently from $x_t^{w_2}$ although their numerical value equals. Amati and Rijsbergen named the normalization of the term frequencies by the window length the *second normalization principle* [5]. In general, language models have to account for the size of the windows in order to prevent bias towards longer windows.

Normalization of document length is a recurring topic in information retrieval [128; 15; 28]. Applied to the single document model, the windows should be re-scaled to the average window size in order to normalize the occurrences of t. This can be be implemented by multiplying the term frequency with the proportion of the window size of the average window size [5]:

$$xn_t^w = x_t^w \cdot \frac{|w|}{|w|_{\text{avg}}} \tag{2.10}$$

The size of a window is determined by the number of terms in the window. The normalized term frequency is denoted as xn_t^w and the average length of the window as $|w|_{\text{avg}}$.

The following example may illustrate this effect [5]: Let the average window size $|w|_{\text{avg}}$ be 2000 and the number terms in $|w_1| = 8000$ and $|w_2| = 200$. If a term t appears 8 times, I re-scale $x_t^{w_1}$ in w_1 to $xn_t^{w_1} = 2$ and in w_2 to $xn_t^{w_2} = 80$. In my experiments the proposed normalization reduced the extracted keywords of above-average length windows satisfactorily but performed poorly on short windows. The results seemed suboptimal and further investigation revealed the reasons.

The presented normalization formula postulates, that an increase of window length corresponds to a linear increase of the number of occurrences of a term t in the window. Recalling the properties of the BIM, a linear correlation is highly improbable. It is rather probable that new terms appear in the window as its size increases [128]. The term frequency density is a decreasing function on the window length and can also be normalized with the following equation [5; 28]:

$$xn_t^w = x_t^w \cdot log\left(1 + \frac{|w|_{\text{avg}}}{|w|}\right) \tag{2.11}$$

This equation fits the needs of a fast and parameter-free approach as it accounts for higher terms frequencies and a larger number of terms in long windows. Consequently, I apply this normalization of term occurrences in Equation 2.11 to the proposed weighting algorithms presented in Section 4.3 and replace x_t^w with xn_t^w in the length-driven approaches.

There exist a number of other normalization techniques that are mentioned in literature, such as: the *cosine normalization* for VSMs, the *maximum term frequency normalization* used in Inquery [104], or the *pivoted normalization scheme* [128]. None of these normalization methods resembles a decreasing term frequency function and fits the requirements for a fast and independent single document analysis. Therefore, I did not consider them for this work.

2.4.6 Smoothing Methods for Language Models

The aforementioned *probabilistic language model* does not take into account the probability of unseen words, and to assign a probability of zero to missing terms is generally regarded as radical. The purpose of smoothing is to assign a probability to unseen words that is different from zero [164; 104]. Most smoothing methods discount the probabilities of the words appearing in the document to assign an extra probability to the unseen words. The following example illustrates this effect:

query: “hurricane”, “season”, “USA”

The query contains three words and I aim to identify a window in the document d that is closely related to this query. If a window w contains only two of these terms, the missing query term would assigned a probability of zero for this window: $p(t, w) = 0$. Instead, the following smoothing can be realized [163]:

$$p(t,w) = \begin{cases} p_s(t,w) & \text{if } t \text{ appears in } w \\ \alpha_w \; p(t,d) & \text{otherwise} \end{cases}$$

The probability $p_s(t, w)$ is the probability after smoothing, and α_w is a coefficient ensuring that all probabilities sum to one. Further details on how to determine $p_s(t, w)$ can be found in the work of Zhai and Lafferty [163]. The following three smoothing methods were used to implement their approach: *The Jelineck-Mercer Method, Bayesian Smoothing using Dirichlet Priors,* and *Absolute Discounting* [164]. All three methods require parameters that must be precomputed beforehand. As I analyze parameter-free IR algorithms in this work, no such smoothing method is considered for my analysis. Moreover, a term that does not appear in a window is unlikely to be a promising keyword candidate.

2.5 The Key Adaptations

The specific task of single document analysis demands adaption of standard IR models and procedures.

Whereas various preprocessing steps are common in IR, implementation of only selected preprocessing steps is required. In this work, all words in the input texts are down-cased in order to increase potential matches. The advantage of down-casing is that most letters in the English and German language already appear in small case and some letters such as “ß” do not have an uppercase version. Besides, the readability and legibility of lower case text is higher than with all caps which facilitates the debugging of the samples used in this work [158].

Furthermore, some common preprocessing steps such as stop word removal and synonym recognition are omitted. This allows for emphasis of the characteristics of the presented approaches.

In Section 2.2 a document model is described that fits the specific purpose of single document analysis. *Windows* are defined as partitions of a document and are an essential element of that model. Suitable position and size of a window are crucial to the performance of the keyword extraction algorithms. Thus, different criteria for window size determination have been determined and presented. One of the criteria – the *number of extracted keywords* – is the foundation for my self regulation approach as presented in Section 8.3.2. Windows of different size require length normalization. Selected normalization models have been adapted to the document model.

My analysis is based on the IR models

- Bag of Words Model,
- Vector Space Model (VSM),
- Probabilistic Language Model, and
- the Divergence from Randomness model

presented in Section 2.4. These models have been adapted here to single document analysis and are applied to two different scenarios:

1. determination of potential keywords, and the
2. combination of keyword extraction approaches.

This chapter introduced several elementary models and methods, as well as their adaptations of these models for analysis of single documents.

Chapter 3

Related Work

IR from single documents is extremely popular these days and a large number of publications deal with that topic directly or indirectly. This chapter contains the state-of-the-art methods to extract the most meaningful information from single documents. The work presented in this chapter is relevant to the main approaches in this thesis.

As an overview, I introduce document summarization approaches with a focus on techniques applicable to single documents. As this work focuses on keywords, I will address the most relevant methods for keyword extraction subsequently. Combination methods can be applied to improve the performance of keyword extraction algorithms. In Section 3.2.1 I describe the most fundamental combination methods. One of my approaches is based on the concept of CPD. I present the most relevant approaches to change-point detection in Section 3.3.

3.1 Automatic Summarization for Single Documents

The sale of *Summly* generated an immense amount of attention by the world's news media in 2013. Summly is a software application that generates quick story summaries for news pages on iOS smartphones [123; 136]. The technology behind *Summly* has not been published in detail but in principle the application extracts the text from a website and applies a machine learning algorithm and natural language processing (NLP) to generate a short paragraph summary. Summaries do appear in various forms: sentences, paragraphs, n-grams, or even keywords.

In general, a *summary* is defined as a condensed version of the original document. The goal of automatic summarization is to extract the most important content from a text and present it to the user in less space. Different approaches exist to extract information from unstructured machine-readable documents, but they mostly conform upon creating semantic annotations. Summaries of documents can be classified into *generic summaries* and *query-relevant* summaries (also called *query-biased* summaries) [156]. Generic summaries are created independently whereas query-relevant summaries refer to a given query or topic. Summaries of documents are considered *extraction-based* when they consist of segments of the source document [156]. Teng et al. proposed a single document summarization by calculating sentence similarity in a document to define topics [141]. *Abstraction-based* summaries use natural language generation technologies to paraphrase sections of the source document. In general an abstraction-based summary can condense a text more strongly than an extraction-based summary. If one would consider the heuristic approaches in this work in a summary scenario, they would be classified as generic extraction-based summarization.

One of the most popular extraction-based text summarization tools is the aforementioned *Summly* [123; 136]. *TextTeaser* is a multilingual application programming interface (API) startup that follows a similar approach as *Summly* but offers a Web API instead [142]. Other competitors in the market of text summarization tools are Cruxbot, the Copernic Summarizer, and Topicmarks [34; 31]. As Summly was acquired by Yahoo in 2013, Topicmarks was acquired by the social discovery website Tagged in 2011 [146]. These acquisitions point out an addition application area: The generation of bi-directional recommendations, based on content in social media (see Section 9.1).

3.2 Keyword Extraction Algorithms

Qi He states in his review article that simple probabilistic models show best performance [55]. One of the reasons is that nonparametric algorithms do not require the use of a training set of documents. One of the most popular and widespread heuristic ranking functions for terms has been proposed by Jones [64] – it is known as TF-IDF. The definition of term specificity later became well known as Inverse Document Frequency (IDF) and in combination with term frequency (TF) it has proven to be extraordinarily robust. Numerous variants can be found, also outside the domain of keyword extraction [118]. The project Video Google aims to localize objects and scenes within a video [130; 129]. Therefore, matches on descriptors of objects are pre-computed as a visual analogy to words. Documents are represented by weighed vector of visual words. The ranking function applied to documents and a single query document (image) is TF-IDF. Subsequently, the visual frames are ranked by their normalized scalar product to retrieve matches between the documents in the collection and the query. Whereas some of the analogies from text retrieval could be applied to image retrieval, there were still some differences between the bag-of-words concept for documents or visual words [129].

Another fundamental approach to keyword extraction is based on the Bayesian Decision Theory. It is a fundamental statistical approach for classification – terms are classified as keywords or not. Turney [149] was one of the first who has defined keyword extraction as a supervised learning task. He combines the genetic training algorithm *Genitor* with the heuristic *Extractor* to create *GenEx*. The extractor filters stop words, performs stemming, and assigns a weight to the stems so that the output is an ordered list of mixed-case phrases. The Genitor optimizes a set of bit strings by randomly changing existing *individuals* and by combining substrings from *parents* to generate new *children*. Each individual is assigned a score, denominated *fitness*. Genitor is a steady-state genetic algorithm. It updates one individual at a time, resulting in a continuously changing population. Usually the least fit individual is being replaced by the new individual. Generational algorithms update their entire population in one batch hence creating a sequence of generations. GenEx has twelve parameters to maximize performance of the algorithm [149]. Tuning such a number of parameters is a sophisticated, domain-specific process.

There are similarities between keyword extraction and trend detection, especially when they face common constraints. The work of Schubert et al. faces the challenge of detecting emerging topics in data streams [1]. Their approach is three-fold: A significance measure in combination with hash tables, and clustering approaches in combination aim to detect emerging topics. The use of hashing and clustering techniques allows a high throughput of data and meaningful results at the same time. Although their objective is rather different from the one in this work, their proposed significance measure of a trend is closely related to the burst scores that I describe in Section 4.4. Contrary to their approach, the scores generated in this work do not require tuning parameters and a learning process.

Learning Algorithms

Similar to GenEx, the Keyphrase Extraction Algorithm (KEA) extracts keyphrases from a document collection. Instead of using a genetic learning algorithm for training, it is based on documents with author-assigned keyphrases [44]. These documents create a model for keyword extraction [159]. KEA chooses keywords based on this model employing the naive Bayes machine learning technique [159]. More pre-processing steps must be performed until the first keywords can be extracted and the TF-IDF weight is used to distinguish keyword candidates. For the speed of KEA, it is essential that documents belong to the same domain. Irrespective of the development of KEA, machine learning techniques have become more popular for keyword extraction in the recent years. This is due to the fact that improved machine learning techniques, large document collections, and enormous computational power have come together [86]

In contrast to the naive Bayes model, the Hidden Markov Model does not assume statistical term independence. Conroy et al. [30] have trained a Hidden Markov Model to assign a summary likelihood to each sentence of a text and create a summary composed by the sentences with the highest probability. A different approach based on the Hidden Markov Model is described in Section 4.4 [66]. A fundamental feature of Kleinberg's model is the applicability to indefinite data streams. The processing of fast-flowing, ever-changing text streams becomes increasingly important and the need to identify hot topics rises. The work of Schubert et al. presents a novel method for the detection of emerging hot topics in a data stream based on manually assigned keywords and clustering [1].

Graph-Based Algorithms

A graph-based document representation can be built upon lexical similarities or semantic chains in a document. The Google PageRank algorithm [100] and Kleinberg‘s HITS algorithm [67] are the most prominent link-based search algorithms that take into account graph information. Litvak et al. [79] apply HITS to document graphs to determine the top-ranked nodes in the graph in order to identify the most significant keywords. Link-based algorithms require a link structure between individual texts, and their application is expedient in large collections such as the internet or a library.

Linguistic Features

A number of methodologies use linguistic features in combination with filtering and lexical analysis to identify relevant terms in a document [127]. Kumar et al. use an n-gram filtration to extract potential keyphrases [73]. The LAKE algorithm is another approach that couples linguistic analysis with a learning algorithm that uses features such as TF-IDF for scoring [6]. Ontology-based approaches are usually very complex and content specific as ontologies can't be transferred from one content to another. Gao et al. aim to identify Nigerian fraud e-mails with a linguistic model [47]. In general, algorithms for keyword extraction based on NLP require an extensive linguistic knowledge of the language used [148].

3.2.1 Combination Approaches

Heuristics used in combination have been shown to have supplementary properties that improve retrieval results [5; 76]. Only a few approaches for ranking aggregation can be found in the literature. One method of combining retrieval results is feature ranking aggregation [108; 81]. Prati investigated four different ranking aggregation methods, and demonstrated the compelling performance of combinations towards single heuristics. I describe his most successful approaches in Section 5 and further discuss their performance in Section 8.4. Another approach presented by Li et al. suggests the application of PCA on the result space of term weights [77]. With PCA they create a weighed linear combination of retrieval heuristics that results in a final weight. While the method produces good results, the theoretical foundation of this approach shows some difficulties that I discuss in Section 5.3.1.

A similar application of Singular Value Decomposition (SVD) is LSA, also known as LSI [75]. A vector-space representation of documents and queries is used to identify *latent semantic associations* with co-occurrences of terms. With Singular Value Decomposition (SVD) the dimensionality of a corpus can be reduced in order to expose a limited number of topics. LSA uses a very similar mathematical technique, but the objective of this method is entirely different. It requires a large document collection in order to identify latent semantic associations and the computational cost is significant [86].

3.3 Change-Point Detection Literature Review

The detection of abrupt changes in statistical processes dates back to the first half of the 20th century when the quality in manufacturing processes had to be monitored. Poor and Hadjiliadis mention in their book a range of fields where CPD is used [105]: climate modeling, econometrics, environment and public health, finance, image analysis, network security, and historical text analysis. This list is definitely not an exhaustive list of applications but it shows the versatility of CPD. Depending on the specific application field, CPD may be denoted as *statistical change detection* or *disorder detection*.

Network performance is usually predicted with stationary models because it is often characterized by periods of stationary – a time span with a constant amount of network traffic. These intervals of stationary can be interrupted by abrupt transitions such as a change from a period with high network traffic to a period with low throughput or the detection of denial of service attacks [140]. CPD was applied to network traffic analysis in order to model statistical changes in networks and to improve traffic predictions. With CPD it is possible to estimate the parameters of the stationary model after a change and generate more accurate predictions [38]. It is generally difficult to detect change-points because of their uncertainty but it is an additional challenge to estimate the location of a change-point.

A very simple check for a change-point occurrence can be performed with *likelihood-ratio* tests and a cumulative sum (CUSUM) algorithm [10]. An approach based on likelihood was presented in [59] by Hinkley and Bhattacharyya and Riba and Ginebra [114]. Johnson published a parameter-free approach in 1968 [13]. A CUSUM algorithm monitors the cumulative differences of samples and a "quality number", which could be the mean of the probability distribution. CUSUM is based on a threshold that has to be estimated beforehand.

Another way to address the CPD problem is to use Bayesian approach by calculating the probability of a change-point, based on the posterior probability that a change-point exists. Girón et al. apply a Bayesian approach to the word lengths and word appearances in a book [50]. They aim to show the existence of two authors in the Catalan book *Tirant lo Blanc* by assuming that the stylistic boundary indicates multiple authorship. Besides, the same book was analyzed with a likelihood-based CPD analysis by Riba and Ginebra with very similar results [114].

Downey proposed an algorithm for the detection of the location of change-points in a time series, based on the calculation of the probability for each index i, that the last change-point occurred at position i [38]. This Bayesian approach is based on the work of Chernoff and Zacks [24] who designed a Bayesian estimator for a process with changing means. Downey's approach requires subjective prior probabilities, and is time consuming proportional to the complexity of $\mathcal{O}(n^2)$. The denotation n refers to the data points or time steps, which leads to a particularly negative impact for large gaps between change-points.

Non-Parametric Approaches

A non-parametric CPD algorithm for written texts has been presented by Johnson et al. [62]. They utilize an entropy estimator to estimate statistical changes in documents and perform analysis of concatenated texts to demonstrate that their algorithm works well. This algorithm is based on ideas from information theory and forms the foundation for the algorithm of this work (see Section 6.1). To highlight differences between this algorithm and the original work presented in this thesis, some of my experiments are based on the same examples as in [62] (see Section 8.6).

Change-Point Detection Based on Entropy Estimators

A number of entropy estimation measures have been employed in other research areas such as neuroscience to describe the amount of information processed by the human brain [48]. Similarly, entropy serves as a universal measure in information theory to quantify statistical effects of word order in language [93]. Montemurro and Zanette compared the entropy of human written texts of eight different languages with those of random texts. They showed a significant difference between the entropy of the random texts and the human written texts but also observed a variability among the entropy of the texts of the eight languages. Their approach is very similar to the problem of CPD, as they already aim to compare texts with different statistical properties while knowing the beginning and the end of the text. I mentioned in a publication that the relationship between uncertainty of terms and their meaningfulness can be beneficial for information extraction approaches [19]. The Gestalt theory in Computer Vision and the probabilistic model Divergence from Randomness are likewise based on this principle, and were utilized in information extraction approaches [8; 5].

Online and Offline

It is possible to classify CPD algorithms as *offline* or *online*, which basically indicate the way they are applied rather than the nature of the algorithm itself. In many cases, CPD is performed as an offline analysis because the main goal is to identify a single change-point in a data sample and characterize a statistical behavior before and after the change. In some applications such as network analysis, it may be beneficial to detect changes in real-time. Online algorithms usually monitor new data and raise an alarm as soon as the probability for a change-point reaches a certain threshold. Some online approaches continuously read the input data and process it incrementally.

Compression-Based Similarities

The measurement of similarity between texts based on entropy has been applied to research areas such as *language recognition, authorship attribution, classification of sequences,* and *plagiarism detection* [143; 11; 9]. A common approach is to train compression algorithms with a collection of texts of a specific category in order to optimize their compression rate for this category of texts. The subsequent compression of an unknown text with the optimized algorithms allows for conclusions on the texts similarity. LZ is often the algorithm of choice.

Detection of Hidden Passages

A closely related scenario with rising significance is the detection of hidden passages within written texts. Governmental, as well as corporate organizations, are exposed to the potential risk of losing information that may be sent outside of the intranet as a hidden passage within text [89]. The detection of hidden passages is also of critical importance in the finance sector when an insider attempts to leak insider information [90]. Mengle et al. proposed an algorithm for the detection of hidden passages in texts based on classification and supervised learning [90]. His method requires an extensive amount of training as one of his training dataset comprised 18 000 documents. Furthermore, the category prediction result depends on the number of training documents of that category [89].

Recent plagiarism detection methods aim to identify short passages that have been copied into a long document by using structural information [132]. Stamatatos showed that by using structural information based on stop-words, he can capture local similarities between texts and identify boundaries of plagiarized passages. His method is based on a list of stop-words that has to be prepared manually and comes with all the deficiencies of stop words that I described in Section 2.2.2.

The distribution of symbols and words in a text is modeled by numerous information extraction approaches that identify structures within a text [17; 93]. In Chapter 7, I introduce a word-based entropy estimation algorithm based on the principle of the Lempel-Ziv compression that pursues a single-pass strategy. Word-based text compression algorithms were evaluated in previous studies were found to perform well compared to character-based approaches that are more common in literature [60]. In fact, the Lempel-Ziv algorithm performed the best out of all that were tested because it was able to take long-range correlations into account.

Chapter 4

Term Weighting Algorithms for Single Documents

The weight of a term represents its significance within its context. There are remarkable algorithms available that aim to determine term weights for keyword extraction or indexing. With no more than a plain text, they are able to identify the most meaningful words for us humans. Unfortunately, none of them is optimal and clearly superior to the others. Optimal means that it is faster, more reliable than the other algorithms, and its resulting keywords are outstanding in all possible scenarios.

First, I will briefly introduce the general concept of keyword extraction with the heuristics presented in this Chapter. In Section 4.2 I will clarify why none of the algorithms is superior to the others. Subsequently, I briefly describe the different reference algorithms used to create well performing retrieval algorithms. Based on the analysis of the retrieval constraints described in Section 4.2 and the reference algorithms, it is possible to combine well-performing algorithms. In Chapter 5, I present such an algorithm. This combined algorithm even outperforms the well-known TF-IDF algorithm (see Section 8.3).

4.1 Keyword Extraction Procedure

The keyword extraction approaches presented in this chapter follow the same general procedure. All of the algorithms will generate a term weight $\mathcal{W}$ for each term of the document. If $\mathcal{W} > 0$, the term is considered a keyword with the weight $\mathcal{W}$. The following algorithm describes the iterative steps for keyword extraction of a plain text.

Algorithm 1 Keyword Extraction for a Single Document

```
Input: $d$ = text document {the source text}
 1: preprocess(d) {perform necessary preprocessing steps}
 2: segmentation(d) {segment $d$ if necessary}
 3: normalization(d) {normalize $d$ if necessary}
 4: for all $t$ in $d$ do
 5:   generate weight $\mathcal{W}$ for $t$
 6:   if $\mathcal{W}_t > 0$ then
 7:     t is keyword
 8:     $t.weight = \mathcal{W}_t$
 9:   end if
10: end for
Output: write(keywords)
```

After performing this algorithm, all terms in the document contain a term weight that is either zero or greater than zero. This allows the creation of a weighed list of terms – the list of keywords. Algorithm 1 gives a basic idea of how keyword extraction can be performed. The reference implementation of the approaches presented in this thesis follows this algorithm and is described in Appendix A.

4.2 Analytical View of Retrieval Constraints

The quality of the results of keyword extraction algorithms is closely related to the properties of the retrieval heuristics used to determine weight of the potential keywords. In this section, I adapt existing retrieval constraints for IR to term weighting algorithms for single documents and extend them. The retrieval constraints for single documents differ significantly from the well known IR scenario.

Fang et al. formally defined and characterized a set of desirable constraints that retrieval heuristics should meet [42]. They concluded, that none of their analyzed retrieval formulas satisfies their formal constraints which comprise the first three of the following four characteristics:

1. *term frequency*,
2. the effect of *window frequency*,
3. *length normalization*,
4. *burstiness*.

A high *term frequency* does not mean that the term is a keyword, but the term frequency definitely has an impact on the importance of a term in a document. In Section 4.3.2, I will show that there is a characteristic relationship between position and frequency in human written documents.

The effect of *window frequency* is closely related to the IDF-part of TF-IDF; terms that are evenly spread across a document should be penalized. Conversely, terms that appear in a very limited space should receive a higher weight because they are probably important. Additionally, texts of different length should be normalized according to the constraints described in Section 2.4.5.

The three characteristics described above aim to characterize the properties of a successful keyword extraction algorithm. They are all based on the bag-of-words-model described in Section 2.4.1. A phenomenon that is discounted by the bag of words model and that raised attention in the field of text data mining in the 1990s is the appearance of *bursts*. Church and Gale first mentioned the phenomenon of burstiness in their publication [25]. The research group He et al. define burstiness as follows:

“A word in a news stream is *bursty* if it appears in a large number of documents over a finite time window.” [55]

According to this definition, a burst is characterized by a sudden unexpected rise of term frequency in a short period of time. This behavior clearly distinguishes terms from low-frequency words and stop-words with a consistently high frequency. Considering a document as a stream of content, the appearance of bursts may indicate significant topic changes. Consequently, a successful retrieval algorithm should also satisfy the characteristic of *burstiness*.

If the search for a term t in a text has been unsuccessful for a long time and suddenly t appears once, the expectation to find more terms rises. This effect of rising expectation is called the *aftereffect* of future sampling and similar to the notion of *burstiness* [43]. The probability of an observed term t contributing to the discrimination of a window is assumed to correlate to the probability of another appearance of t [5].

Formal Definition of Retrieval Constraints

I applied the formal constrains proposed by Fang et al. to the scenario of keyword extraction from single documents [42]. As keyword extraction is considered a specialization of an IR scenario, where the potential keywords equal the search terms of a query, these constraints continue to be valid without further proof. In the following, I adapted the more general IR related constraints to the special case of keyword extraction for single documents [42].

For this purpose, I refer to the document model described in Section 2.2 and Section 2.3. Here, I refer to a document d that has been segmented into windows $w_1..w_n$. The symbol $|w|$ refers to the size of the window w and the weight of the term t in window w is designated $\mathcal{W}(t,w)$. The term frequency of t in w is given by x_t^w.

The following two conditions describe the expected behavior of a retrieval algorithm with respect to the term frequency of the term t. Condition 4.2.1 describes the fact that windows with a higher score must result in a higher weight of t. If a term appears more frequent in w_1, there should be a higher weight assigned to t in w_1. Condition 4.2.2 ensures that the weight increase is smaller for larger term frequencies. The weight difference between a frequency of 1 and 2 is larger than the weight difference between a frequency of 100 and 101.

Condition 4.2.1. *Let term t be the potential keyword and t be a unigram ($|t| = 1$). Assume $|w_1| = |w_2|$. If $x_t^{w_1} > x_t^{w_2}$, then $\mathcal{W}(t, w_1) > \mathcal{W}(t, w_2)$.*

Condition 4.2.2. *Let term t be the potential keyword and t be a unigram ($|t| = 1$). Assume $|w_1| = |w_2| = |w_3|$ and $x_t^{w_1} > 0$. If $x_t^{w_2} > x_t^{w_1}$ and $x_t^{w_3} > x_t^{w_2}$, then $\mathcal{W}(t, w_2) - \mathcal{W}(t, w_1) > \mathcal{W}(t, w_3) - \mathcal{W}(t, w_2)$.*

Condition 4.2.3 states the following: if two terms appear in a document with the same frequency, the term with the lower window frequency (higher iwf) receives the higher term weight – less evenly spread terms receive higher term weights.

Condition 4.2.3. *Let term t_1 and t_2 be two potential keywords and document $d = \{w_1, w_2\}$ contains two windows. Assume $|w_1| = |w_2|$ and $x_{t_1}^{w_1} + x_{t_1}^{w_2} = x_{t_2}^{w_1} + x_{t_2}^{w_2}$. If $iwf(t_1) > iwf(t_2)$, then $\mathcal{W}(t_1, d) > \mathcal{W}(t_2, d)$.*

The following constraint describes the condition of an optimal normalization property. Condition 4.2.4 states that the weight of a term t in w_1 is greater than the weight in w_2 if the number of t are the same in w_1 and w_2 but w_2 is longer than w_1 – it basically penalizes long windows.

Condition 4.2.4. *Let term t be the potential keyword and w_1, w_2 two windows. Assume $|w_2| > |w_1|$ and $x_t^{w_1} = x_t^{w_2}$, then $\mathcal{W}(t, w_1) \geq \mathcal{W}(t, w_2)$.*

Condition 4.2.5 prevents that a term in window w_1, that is a concatenated multiple of w_2, receives a lower score than t in w_2. This condition prevents over-penalization of long windows.

Condition 4.2.5. *Let term t be the potential keyword and w_1, w_2 two windows. $\forall k > 1$, if $|w_1| = k \cdot |w_2|$ and for all t, $x_t^{w_1} = k \cdot x_t^{w_2}$, then $\mathcal{W}(t, w_2) \leq \mathcal{W}(t, w_1)$.*

Condition 4.2.6 regulates the relationship between the length of a window and the number of occurrences of t. If w_1 equals a version of w_2, which contains a few more occurrences of t, then t should receive a higher weight in w_1.

Condition 4.2.6. *Let term t be the potential keyword and w_1, w_2 two windows. Assume $x_t^{w_1} > x_t^{w_2}$. If $|w_1| = |w_2| + x_t^{w_1} - x_t^{w_2}$, then $\mathcal{W}(t, w_1) \geq \mathcal{W}(t, w_2)$.*

These constraints comprise the first three characteristics for a successful algorithm for single document IR. A formal (probabilistic) definition of burstiness was provided by Clinchant and Gaussier [29]:

Condition 4.2.7. *A distribution P is bursty iff the function*

$$f_\epsilon(x) \quad = P(X \geq x + \epsilon | X \geq x), \quad \forall \epsilon > 0 \tag{4.1}$$

is a strictly increasing function of x. A distribution which verifies this condition is said to be bursty [29].

A bursty probability distribution occurs when a few outcomes of a term have been observed and the distribution has passed a certain threshold, the probability of more outcomes of the term rises. It is important to emphasize that burstiness does not refer to a large number of occurrences at a certain point – a single state – but rather to a behavior with a period of increase and decrease. Applying this constraint to standard probability distribution, one can conclude that the binomial distribution as well as most of the standard probability distributions are not bursty. Then again, the Laplace law of succession and the Negative Binomial Distribution are bursty but the Negative Binomial Distribution is not applicable for single documents [27]. In Section 4.4, I describe the most important algorithms that can detect bursty behavior in texts.

I reason that a successful retrieval heuristic for keywords should meet **all** the above described constraints. Unfortunately no such retrieval heuristic exists [27]. In Section 4.5, I will show that the retrieval heuristics used in this work do not meet all the requirements but can be categorized according to them.

Therefore, I propose that a combination of multiple heuristics can provide improved and more reliable results than an individual algorithm. In Chapter 5, I introduce a selection of the best combination approaches. One of the key findings of my work is the analysis of the performance of different combinations for different scenarios and the characterization of combinations with respect to the properties introduced in this section.

4.3 Frequency-based Information Measures

In this section, I introduce five different term weighting algorithms that can be used to extract keywords from texts. I selected these algorithms because they are well known in IR and they behave differently in different scenarios. All of these algorithms consider term frequency in their calculation of the term weights but do not consider burstiness.

4.3.1 On Information Content and Informativeness

A keyword extraction algorithm aims to identify terms that are important for the content of the document and terms that describe the topics of the document best. The works of Zipf, Luhn, Harter, Amati, Edmundson, and Jones assume that the difference between high term frequencies and low term frequencies of a term allow for conclusions to be drawn regarding the informativeness of the term [40; 83; 168; 5; 53; 64]. Terms that are likely to be informative are also referred to as "speciality" words, whereas terms that are usually randomly distributed in a document are referred to as "non-specialty" words [53]. As I aim to identify speciality words, the following question arises: how can informativeness of terms be measured?

Tomokiyo and Hurst provide one answer to this question in their paper [145]. They determine an *informativeness score* and a *phraseness score* of a term. These scores are generated with a binomial log-likelihood ratio test as well as with the point-wise Kullback-Leibler (KL) divergence between two language models. Their LMs are based a foreground and a background distribution, which refer to a single document and a document collection. Their KL-based approach creates reasonable results, but it is not a good measure of dependence as it favors low-frequency scores over high frequency scores [85].

Another approach is proposed by Amati [4]. He calculates a term weight based on the product of two factors, where one of these factors is denominated *informative content*. In his thesis, he defines the informative content of a term as the negative logarithm of the probability of the term occurrence. This probability is based on a model of randomness that is shown in Section 2.4.4 in Equation 2.7. This idea was first introduced by Popper under a different mathematical representation [106]. The smaller the probability according to the model of randomness, the less conform is the term distribution with the model of randomness and the higher is the informative content of the term. In my work, I utilize this model to calculate the informative content (the weight) of the potential keyword for term probabilities generated by following language models: Helmholtz Approach, the Bernoulli Model of Randomness, and the Poisson distribution. With this definition of informativeness, it is possible to translate the probability of these language models into a term weight or a measure of informative content. Another application of this definition of informative content can be found in Section 5.1.

4.3.2 Analysis of Term Frequencies

The term frequency of a potential keyword is one of the essential properties described in Section 4.2. It is considered in almost every term-weighting algorithm. Luhn published one of the first papers about keyword extraction based on term frequency [83]. *Luhn's Assumption* states the following [7]:

Definition 4.3.1. *The value, or weight, of a term t that occurs in a document d is simply proportional to the term frequency x_t^d. That is, the more often a term t occurs in the text of the document d, the higher its term weight $\mathcal{W}$ is.*

In 1932, Zipf recognized that term frequencies in texts of natural languages are inversely proportional to their rank in the frequency table [167; 168]:

$$x_i \sim \frac{1}{i}$$

Hence, they follow a power-law distribution. Count all the terms in a document, sort them by number of occurrences and write them in a table in decreasing order. Zipf states, that the frequency x_i of the i-th most term t_i in an English text is $\frac{1}{i}$ times the frequency x_1 of the most frequent term in the text [7]. The parameter α characterizes the distribution and was originally set to $\alpha = 1$ by Zipf for the English language.

$$x_i = \frac{x_1}{i^\alpha}$$

The parameter α has to be determined empirically and characterizes the distribution. Zipf's distribution holds for many other ranking scenarios such as the population ranks of cities in China or the magnitude of earthquakes [96]. A generalized version of this law is known as Zeta distribution.

The example in Figure 4.1 uses the top 14 conference paper abstracts from Engineering of Computer Based Systems (ECBS) 2012 to illustrate Zipf's law by plotting the terms after ordering them in descending order according to their frequency.

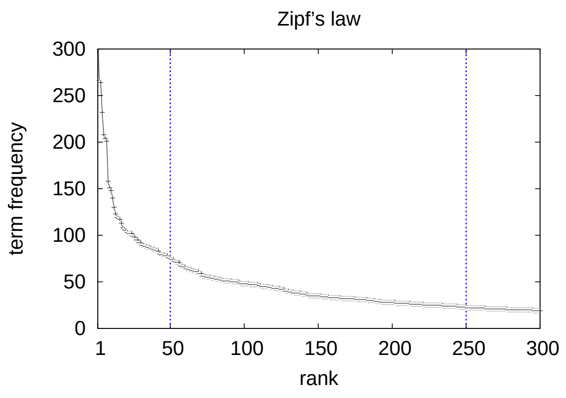


Figure 4.1: The graph shows the term distribution of the terms in the top 14 ECBS 2012 conference paper abstracts after preprocessing.

Luhn stated that the most frequent words in a document are usually the most common ones [83]. They are not containing a lot of information. Also the very rare ones should not be considered as potential keywords. Besides that, it is assumed that the author does not intend to mislead the reader by using words that are unrelated to the content.

This means that the keywords should be found on neither side of the blue dotted lines in Figure 4.1 but somewhere in between. This statement unfortunately does not hold for all texts, as Ventura et al. describe in their publication [154]. Additionally, it is difficult to determine the threshold (the exact position) for the *very frequent* and the *very rare* terms in a document.

But indeed, the number of topics in a document and how closely terms match the topic they represent can be measured. The amount of information covered by a document description and the amount of information provided by keywords can be defined as follows:

Exhaustivity and Specificity

Exhaustivity is a property of keyword descriptions, *specificity* is a property of keywords. The term exhaustivity describes the coverage of a document description for the main topics of the document. Specificity describes the degree of correctness with which a keyword describes a topic of a text. [64; 7]

Karen Spärck Jones stated in her seminal paper that the exhaustivity of a document encompasses the various topics that are covered by the extracted keywords [64]. Statistical exhaustivity is then the number of terms a document contains and an essential factor of the well-known TF-IDF term weighting algorithm [64]. And that is the basis for the TF-IDF approach where the term frequency is only a factor of the final term weight, which is also determined by the *specificity* of a term.

4.3.3 Term Frequency - Inverse Document Frequency (TF-IDF)

One of the most popular term weights in IR is the TF-IDF algorithm. Karen Spärck Jones concluded that popular words distributing equally over the whole document should be penalized [64]. Therefore, Jones defined the specificity as the number of documents that contain the term – the collection frequency. In 1972 she converted the specificity of a term in a document into a simple equation which later became famously known as IDF.

The IDF-score is used to scale the TF of a term. Its purely heuristic nature has led to many theoretical explanations [118]. The use of the logarithm in IDF by Jones seems rather intuitive, and although she used a logarithm base 2, the base of the logarithm is not really important [118; 86].

In the case of a single document, I apply IDF as a measure for the number of windows that contain the term – the Inverse Window Frequency (IWF). Let the number of windows in the document be N, where the term t occurs in $w(t)$ of them. The Inverse Window Frequency (IWF)-weight for single documents for t is adapted as follows:

$$iwf(t) = \log \frac{N}{w(t)} \tag{4.2}$$

The original proposal of IDF from Spärck Jones[64] does not make use of within-window TF information. Consequently, there was no combination of TF and IDF. I recall, the *term frequency* is the raw count of the number of occurrences x_t^w of the term t within the particular window w.

$$\mathcal{W}_{TF-IWF} = x_t^w * iwf(t) \tag{4.3}$$

Apart from this very basic equation, a number of variants of the original TF-IDF weighting scheme are common [7]. To account for longer windows, the term frequency x_t^w could be normalized as in Equation 4.4 [154]. I describe further normalization techniques in Section 2.4.5. In this work, I refer to the above variant of Term Frequency - Inverse Window Frequency (TF-IWF) with the normalization described in Section 2.4.5.

$$xn_t^w = \frac{x_t^w}{x_t^d} \tag{4.4}$$

The TF-IWF weight $\mathcal{W}_{TF-IWF}$ increases with a high term count in a single window and a generally low appearance of the terms in the whole document. The original TF-IDF weighting scheme has proven to be extraordinarily robust and difficult to beat [118]. A serious problem with TF-IWF is, that only the single potential keyword terms are taken into account and all the other vocabulary is ignored (see BIM).

4.3.4 Helmholtz Approach

The Helmholtz principle is defined as the following statement: meaningful events appear as large deviations from randomness. This concept was first stated by Lowe and is valid for a huge range of applications [82]. In other words: every structure that appears with too much regularity as it is very improbable to occur in noise, calls for attention and may be of importance. Figure 4.2 illustrates that principle with a black rectangle within a rectangle of noise. If the black and white pixels appear with the same, equal probability, the black rectangle is rather improbable. Therefore, the black rectangle clearly stands out.

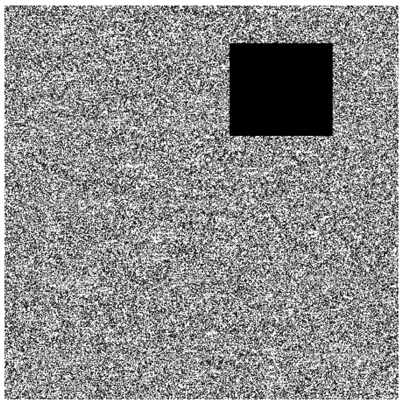


Figure 4.2: An example for a deviation of randomness: a black rectangle within a rectangle filled with noise (black and white pixels)

The Helmholtz Principle is defined as follows:

Helmholtz Principle:

"Gestalts are sets of points whose (geometric regular) spatial arrangement could not occur in noise." [37]

Balinsky et al. present the Helmholtz Principle in the context of automatic keyword extraction [8]. I applied their approach to the domain of keyword extraction for single documents. Let N be the number of windows in a document, whereas the windows $w_1, ..w_N$ are of the same length and the term t occurs in one or more of them. x_t^d is defined as the sum of occurrences of t in the document d. The set $S_t = \{t_1, t_2, ..t_m\}$ contains all of these occurrences of t, whereas $m = x_t^d$.

$$\underbrace{t \quad tt \qquad}_{w_1} \quad \underbrace{\qquad t \qquad}_{w_2} \quad \underbrace{ttt \quad t \quad}_{w_3} \quad \ldots \quad \underbrace{tt \qquad\qquad}_{w_N}$$

Figure 4.3: Sample term distribution modeled as m-tuples in the windows of a document.

Figure 4.3 shows a sample distribution for term t in the windows $w_1..w_N$ of a document. If the significance of a term in the document is related to its appearance, the question must be: Is it an unexpected event if t appears m times in w? Let $x_t^w = m$ and the random variable X_m represents the m times occurrence of t in w. Balinsky et al. propose the expected value $E(X_m)$ as [8]:

$$E(X_m) = \binom{x_t^d}{m} * \frac{1}{N^{m-1}} \tag{4.5}$$

The exponent $m-1$ results from the fact that the probability of m occurrences of t in one window is $\frac{1}{N^{m-1}}$.

Furthermore, Balinsky et al. define $E(X_m)$ in Equation (4.5) as the *number of false alarms* $NFA_T(m, x_t^d, N)$ of an m-tuple of a term t.

$$\mathcal{W}_{NFA} = \begin{cases} E(X_m) & \textit{if } E(X_m) < \epsilon \\ 0 & \textit{else.} \end{cases}$$

If $NFA_t < \epsilon < 1$ it is called *ϵ-meaningful*. The set of extracted keywords consists of words with $NFA_t < \epsilon$, whereas $\epsilon = 1$ in the reference implementation. If the input stream does not consist of windows of equal size, the windows have to be normalized according to the specification described in Section 2.4.5. It may be noticed, that this algorithm does not consider the window frequency (and IWF).

4.3.5 The Bernoulli Model of Randomness

Written text consists of symbols from a finite alphabet. The symbols are assembled in a way so that they compose and separate words and generate a certain structure in a text. Symbols as well as words are not uniformly distributed over a text (see Section 6.2). In fact, they depend on the symbols and words that appeared before. A probability model should therefore consider the previous "draws", as it is done by the Markov Model [111]. A probabilistic model can then be classified by the number of terms k it takes for generating the next term – speaking of a k-order model. The Bernoulli trial is a very basic experiment, that is considered a 0-order model.

A classic Bernoulli experiment is the coin toss, where a coin is repeatedly tossed with the outcome heads or tails. The two main characteristics of Bernoulli trials are:

- The individual trials are independent of each other.
- The probability for each event (success or failure) remains constant.

A Bernoulli trial that has a success value of 1 with probability p and failure probability $1-p$ is defined as:

$$P(X = 1) = p$$
$$P(X = 0) = 1 - p$$

Carried over to information retrieval, the probability p of window w containing the term t can be defined as $p = \frac{1}{N}$, whereas N equals the number of windows in the document d. The complexity of the Bernoulli model is the same as the complexity of the multivariate model [86].

The Binomial Distribution

The difference between a Bernoulli Trial and the basic binomial model is as follows: In a binomial distribution the Bernoulli experiment is conducted in a sequence of trials (multiple sampling). The *binomial distribution* is the discrete probability distribution of the number of successes $p(k)$ from n independent trials. The term *binomial* is based on the binomial coefficient that is related to the number of experiments n and their outcomes k [70]. In case of $n = 2$ the possible combinations of outcomes of an experiment with two trials are the following: two times *success*, two times *success*, and *failure* and two times *failure*.

The probability mass function for a binomial distribution is defined as follows:

$$\binom{n}{k} p^n * (1-p)^{(n-k)} \tag{4.6}$$

Imagine a document d with 100 windows $w_1, .., w_{100}$ and the term t appears 10 times in the d. The probability of 4 occurrences of t in a specific window would then be calculated as follows:

$$P(x_t^w = 4) = \binom{10}{4} \left(\frac{1}{100}\right)^4 \left(\frac{99}{100}\right)^6$$

The binomial distribution in IR is based on the BIM, which assumes no relation between individual terms (see Section 2.4). The general assumption of the Bernoulli Model of randomness is: a term t is spread across a document d with N windows $w_1, .., w_N$ according to the binomial law [5]:

$$P_{Binom}(t|d, w) = \binom{x_t^d}{x_t^w} p^{x_t^w} q^{x_t^d - x_t^w} \tag{4.7}$$

The probability of t appearing once in a specific window is $p = \frac{1}{|N|}$ and the probability of failure is set to $q = \frac{|N|-1}{|N|}$.

Contrary to the TF-IWF heuristic, the Bernoulli model does not take into account the *window frequency* of t. The probability of success can be transformed into a term weight as suggested in Section 2.4.4. In this work, I follow the proposal of Amati et al. and determine the *informative content* $\mathcal{W}_{Binom}(t)$ for each term in the document with the following logarithmic representation:

$$\mathcal{W}_{Binom}(t) = -\log P_{Binom}(t|d, w) \tag{4.8}$$

Accordingly, the terms with the highest informative content (weight) $\mathcal{W}_{Binom}(t)$ are considered the most probable keywords in the document. Furthermore, the logarithmic representation also reduces computational costs.

4.3.6 Poisson Distribution

If the values of $n = x_t^d$ in a binomial distribution become large, the results become computationally expensive and an approximation for the binomial distribution is desirable. The *Poisson* distribution can be used as an approximation if $n \to \infty$ and np converges towards a constant value λ, whereas the probability p equals the probability of the binomial distribution. This behavior can be observed when analyzing very large literary texts such as books with thousands of terms that repeat frequently.

$$P_{Poisson}(t) = \frac{\lambda^{x_t^w} e^{-\lambda}}{x_t^w!}, \tag{4.9}$$

$$\lambda = p * x_t^d \tag{4.10}$$

The mean and the variance of the distribution $P(t)_{Poisson}$ is defined by λ. Using the Stirling formula, Amati et al. define the informative content $\mathcal{W}_{Poisson}$ as an approximation of the Poisson distribution as follows [5]:

$$\mathcal{W}_{Poisson} \sim x_t^w * log_2 \frac{x_t^w}{\lambda} + \left(\lambda + \frac{1}{12 * x_t^w} - x_t^w\right) * log_2 e + 0.5 * log_2(2\pi * x_t^w) \tag{4.11}$$

The informative content $\mathcal{W}_{Poisson}$ is based on the term t, its window frequency and the number of occurrences in the document. In this work, I calculate $\mathcal{W}_{Poisson}$ for each potential keyword in the document. The notion of informative content is introduced in Section 2.4.4.

4.3.7 Okapi BM25

The BM25 model is one version of a series of experiments, whereas the term "Okapi" refers to the retrieval system developed by the London City University [117]. The formula was widely used with great success during the Text REtrieval Conference (TREC) evaluations [116]. The aim of the experiments was to design a formula that comprises the first three properties mentioned in Section 4.2. During the development of this system, a number of models were tested and denominated with the prefix "BM". The BM25 model contains a number of parameters that must be set beforehand.

$$\mathcal{W}_{BM25}(t) = \log\left(\frac{N}{w(t)}\right) * \frac{(k_1 + 1) * x_t^w}{k_1 * ((1 - b) + b * (\frac{|w|}{|w|_{avg}})) + x_t^w} \tag{4.12}$$

The term weight $\mathcal{W}_{BM25}$ is based on the definition of informative content described in Section 2.4.4. The first part of the right side of Equation 4.12 equals the original iwf-weight. Sometimes a slightly different IWF-version is used for the BM25 formula [5]. The number of windows containing the term t is represented by the denomination $w(t)$.

$$iwf(t) = \log \frac{N - w(t) + 0.5}{w(t) + 0.5} \tag{4.13}$$

This alternative iwf-version may lead to negative results if the term appears in more than 50% of the windows. Therefore, the original version of the iwf-score is preferable for the purpose of this work. I use the original iwf-version as it is introduced in Equation 4.2.

The second part of the right side of the formula represents a combination of term frequencies and window lengths. Parameter k_1 calibrates the x_t^w-scaling on the weight. If $k_1 = 0$, then the right side reduces to $\frac{x_t^w}{x_t^w} = 1$ and the term frequencies become ineffective on the term weight. In my reference implementation, I set $k_1 = 1.2$ as it is recommended in the work of Amati [86; 4]. The parameter b ($b \in [0, 1]$) scales the term weight with respect to the window length. No length normalization is performed for $b = 0$. And for $b = 1$ the term frequencies are normalized by the average window length. In the reference implementation, I set $b = 0.75$ as it is recommended in the work of Amati[86; 4]. As the focus of this work is keyword extraction and term weighting for unigrams, the extension for *query-scaling* of the BM25-formula may be dispensed.

4.4 Burstiness

The microblogging platform Twitter allows their users to send short text messages of up to 140 characters – so called *Tweets* – instantly. The users use this method to update their personal status, their experiences, or emotions more frequently – in almost real time. In the domain of microblogging services, a *burst* is a well-known scenario. In social media, bursts are represented by a dramatic change of the activity rate during a limited time period [165; 46]. These high peaks of activity are usually triggered by a real world event and collective excitement, and usually take place in real time. During the Olympic Games in London the 100 meter final victory of Usain Bolt was accompanied by over 80 thousand Tweets per minute (TPM) [152].

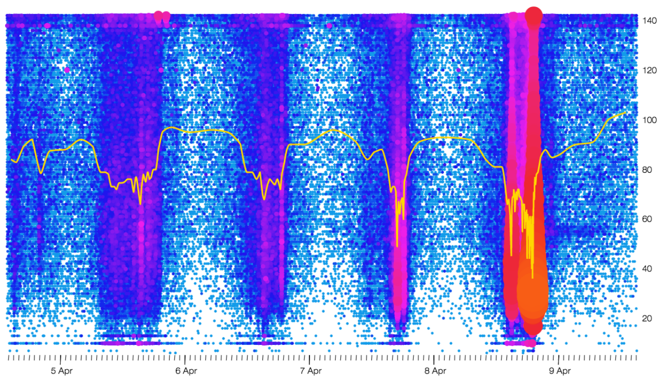


Figure 4.4: This graph shows the number and length of tweets during the Golf Masters tournament in Augusta, Georgia.

Figure 4.4 displays the user activity on Twitter during the Golf Masters tournament in Augusta, Georgia [92]. Every dot is a tweet ranging from 1 to 140 characters, whereas long tweets are at the top and short tweets are at the bottom of the graphic. The yellow line displays the average length of the tweets. During the five-day event more than 40 Million tweets were posted. In Figure 4.4 it can be observed that the tweets burst during the competitions throughout the individual five days, and peak during the final game on the last day. The principle of bursts can be observed in documents as well.

Following the premise that a topic shows bursty behavior in a document or in a text stream, the terms in the text are expected to represent bursty behavior as well. In information retrieval, a number of methods aims to identify bursty behavior [139]. Nevertheless, most frequency-based heuristics consider a document as a *bag-of-words-model* and assume independence between the term occurrences. In Section 4.2, I state that burstiness is one of the properties of a successful keyword extraction algorithm, but burstiness can not be measured with a bag-of-words-model as the (temporal) word order is an essential property of a human written text and an essential criterion for burstiness [93]. Quite a few metrics assume that the word order in a text plays a crucial role and that the *neighborhood* of a term can be used to measure the relevance of a term. In the remainder of this section, I present some of the most successful approaches for modeling burstiness in texts. I later utilize these approaches to create and analyze different combinations of term weighting algorithms and show that certain combinations leads to improved results and meaningful keywords.

4.4.1 The Laplace Law of Succession

The rule of succession was introduced by Pierre-Simon Laplace in 1814. In the context of IR, it states the following [36]:

If the term t has not occurred for a long time and suddenly t appears once, the expectation to find further occurrences of t rises.

This effect of rising expectation is also called the *aftereffect of future sampling* and is similar to the notion of *burstiness* [43]. The probability of an observed term t contributing to the discrimination of a window is assumed to correlate to the probability of another appearance of t [5]. This probability is obtained by the conditional probability $P(X_{x_t+1}|x_t)$ of the term t by the aftereffect model. This probability is only related to the *elite* set of windows – the set that contains the term t. As the Laplace law of succession reduces the term frequencies to $[0, 1]$, it may also be used as a normalization method (see Section 2.4.5).

Amati et al. applied the Laplace Law of Succession to account for burstiness in their proposed algorithm [5]. They first surveyed the performance of several aftereffect models, and found the Laplace Law of Succession provides reasonable results despite its simplicity and linear complexity. I include this method in the set of selected algorithms for combination because it accounts for term frequency of a term t across the whole document and differs significantly from other algorithms modeling burstiness.

Based on the assumption of term independence and Laplace's Law of Succession, the probability of $x_t + 2$ appearances is close to $\frac{x_t+1}{x_t+2}$. Assuming that $x_t^w - 1$ occurrences have been observed in the window an additional appearance of t is approximated with the following equation [4]:

$$P_{Laplace}(x_t) = \frac{x_t^w + A}{x_t^w + A + B} \tag{4.14}$$

The parameters A and B are constants and were set to $A = B = 0.5$ in [4]. This simple formula comes with linear complexity and is independent of the window length. Amati et al. applied the Laplace Law of Succession to account for burstiness in their proposed probabilistic model [5]. The information content associated with this probability distribution is denoted as:

$$\mathcal{W}_{Laplace} = \frac{1}{x_t^w + 1}$$

Besides its simplicity and linear complexity the Laplace Law of Succession has provided reasonable results. Amati proposes a second probability distribution to model the *aftereffect of future sampling*. Therefore, he adopted the ratio of two Bernoulli processes, but the results of this ratio were generally worse than the Laplace Law of Succession [5]. Consequently, the Bernoulli model for modeling the aftereffect is not considered in this work. Moreover, I include this method in the set of selected algorithms for combination because it accounts for term frequency of a term t across the whole document and differs significantly from the following Γ-Metric.

4.4.2 The Γ-Metric

The Γ-metric is based on the assumption that the terms in a document are distributed inhomogeneously and relevant terms appear concentrated in limited areas. This assumption is very similar to the definition of bursts and fits the basic assumption of topics and subtopics in a written text (see Section 2.3). Zhou and Slater developed a metric that is independent of raw term frequency but analyzes term sequences [166]. The difference between a term sequence and the raw term frequency is that a sequence accounts for order, whereas the raw term frequency only tells about the numerical presence of terms. Their Γ-metric is computationally simple, robust, and accounts for term order and long-range correlations in texts. Zhou and Slater define an imaginary time-series of term occurrences, where t_i , with represents the i-th occurrence of the term t:

$$\{t_{begin}, t_1, ..., t_{x_t}, t_{end}\}.$$

Contrary to previous approaches, this one includes the space before the first term occurrence t_{begin} and the distance after the last term occurrence t_{end} [166]. Since common words are evenly distributed, they should appear closely to the beginning and the end of the document. Zhou and Slater define the *separation* around term occurrence t_i with $sep(t_i) \equiv \frac{t_{i+1}-t_{i-1}}{2}$, with $1 \leq i \leq x_t$. The separation represents the median distance of a single occurrence of t_i in the document. The mean waiting time $\hat{\mu}$ between consecutive occurrences is calculated as follows:

$$\hat{\mu} = \frac{|d|+1}{x_t^d+1}$$

The symbol $|d|$ accounts for the total number of terms in the document d. This metric differs between large collections of terms in a small area – a cluster – and isolated pairs of terms. In order to capture this difference and to identify clusters of terms, they introduce the notion of a *cluster point*. The term occurrence at position i is a cluster point, if the separation of t_i is less than the mean waiting time:

$$t_i \text{ is } \textit{cluster point}, \text{ if } sep(t_i) < \hat{\mu}$$

The individual cluster points represent the local excess of term t in the text. The quantity $\Gamma(t_i)$ measures the Γ-value of t_i at position i [166]:

$$\Gamma(t_i) = \begin{cases} \frac{\hat{\mu}-sep(t_i)}{\hat{\mu}} & \textit{if } t_i \textit{ is a cluster point} \\ 0 & \textit{else.} \end{cases}$$

To calculate the spread of t across the whole document, a normalized average of the Γ-value of the cluster points of t has to be calculated. The Γ-metric for a term t in d is defined as follows:

$$\mathcal{W}_\Gamma = \Gamma(t) = \frac{1}{x_t^d}\sum_{i=1}^{x_t^d}\Gamma(t_i)$$

$\mathcal{W}_\Gamma$ is the term weight for the terms in the document. The normalization factor $\frac{1}{x_t^d}$ is necessary to avoid preferential weighting of common terms. Terms with a higher term frequency would score higher, otherwise. The Γ-metric basically counts the *additional* words in a cluster. Zhou and Slater claim that this Γ-metric is able to measure clusters of relevant terms in a text. They also claim that the metric is stable towards term repositioning, does not artificially increase term weights, and remains computationally simple. Unfortunately this metric does not account very well for relevant terms with a low frequency and relevant words that appear in more than one local area of the text.

4.4.3 BursT

Derived from the bursty appearance of topics in microblogging streams, the *BursT* term weighting scheme reflects changes over time and assigns term weights in a dynamic environment. Lee et al. presented this term weighting method for text streams that focuses on online burst analysis by considering the arrival rate of terms as a global baseline [76].

They define the weight of a term as a product of *Burst Score* and the Term Occurrence Probability (TOP). The TOP is a very basic probability, based on the term occurrences in the window. However, the factor burst score is a novel approach for measuring bursts. It is defined by the arrival rate of the terms of the text. The denotation $ar_{t,i}$ in Equation 4.15 describes the reciprocal of the arrival gap between two arrival times t_i and t_{i-1}.

$$ar_{t,i} = \frac{1}{t_i - t_{i-1} + 1} \tag{4.15}$$

While $ar_{t,i}$ is a measure of a single arrival gap, the development of arrival gaps over time may indicate bursty behavior. Therefore, the mean value of the arrival rate has to be calculated. Instead of calculating the mean for all appearances of t at once, an incremental calculation of the mean is chosen. The mean value of the arrival rate of a term t at time i is then calculated with Equation 4.16:

$$\mu_{t,i} = \mu_{t,i-1} + \frac{1}{n_{t,i}} \cdot (ar_{t,i} - \mu_{t,i-1}) \tag{4.16}$$

This allows for a mean calculation while new text arrives and makes this approach applicable for text streams. The number of occurrences of t up to arrival time t_i are denoted by $x_{t,i}$. Lee et al. proposed that the *burst score* $\mathcal{W}_{BS}$ of the current observation result of t at time i directly depends on the distance between the incremental mean and the current arrival rate. The score accounts for shorter arrival intervals than $\mu_{t,i}$.

$$\mathcal{W}_{BS}(t,i) = max\left\{\frac{ar_{t,i} - \mu_{t,i}}{\mu_{t,i}}, 0\right\} \tag{4.17}$$

In case of negative values of the part of the equation $ar_{t,i} - \mu_{t,i}$, the term is considered a *falling word* [76]. Only *rising words* – term appearances with shorter arrival intervals – are taken into account by this algorithm. The authors referred to a sliding window in their work that is being analyzed as it slides across the incoming messages of a microblogging platform. In this work, I refer to the window approaches as described in Section 2.3.

4.5 Analysis of the models based on the retrieval constraints

In the first part of this chapter, I formally defined retrieval constraints for a successful information retrieval algorithm. Subsequently I introduced eight state-of-the-art models for term-weighting. Here, I state how well the above mentioned algorithms fit the retrieval constraints. Table 4.1 contains the weighting formulas of the described algorithms in the rows and the criteria in the columns. The table summarizes the formal criteria as the following four properties:

1. *term frequency*,
2. the effect of *window frequency*,
3. *length normalization*, and
4. *burstiness*.

Table 4.1 illustrates the general properties of the weighting formulas, which can be later utilized to create beneficial combinations. The properties are denoted with *yes* if the behavior of the weighting formula fits one or more formal conditions describing the property. If the weighting formula follows the conditions partially, based on other conditions, it is denoted as *conditional* (cond.) and if it does not follow the property it is denoted with *no*.

term weight $\mathcal{W}$	**TF**	**window freq.**	**norm**	**burstiness**
$\mathcal{W}_{TF-IWF}$	yes	yes	no	no
$\mathcal{W}_{NFA}$	cond.	yes	no	no
$\mathcal{W}_{Binom}$	cond.	yes	no	no
$\mathcal{W}_{Poisson}$	cond.	yes	no	no
$\mathcal{W}_{BM25}$	yes	cond.	yes	no
$\mathcal{W}_{Laplace}$	no	no	no	yes
$\mathcal{W}_{\Gamma}$	no	no	no	yes
$\mathcal{W}_{BS}$	no	no	no	yes

Table 4.1: Properties of the individual term weighting algorithms.

Table 4.1 shows clearly that none of the algorithms meets all the constraints of a successful retrieval algorithm. It can also be observed, that the different weighting formulas meet different properties. All the properties are met at least once by one of the selected algorithms. This, the assumption that a combination of these algorithms might be beneficial follows. Therefore, the next section introduces several combination methods.

4.6 Performance Analysis of the Heuristic Algorithms

The heuristic models described in this chapter do show differences in their properties for extraction, which may affect their performance in a software application. In this section, I analyze the performance of the heuristics described in this chapter. I use an implementation that is further described in Appendix A. The implementation follows the steps described in Section 4.1 for each algorithm.

The performance of a Java application is can be divided into two properties:

- the runtime of the application and
- the memory consumption of the application.

In the following, I compare the runtime performance and the memory consumption of the heuristics described in this chapter.

Runtime Analysis

I performed an analysis of ten text documents with all the weighting algorithms described in this chapter. The smallest text document has a size of 6 kilobyte and the largest file 217 kilobyte. The texts were preprocessed and segmented into 30 almost equally-sized windows in order to avoid very small and very large window sizes. A normalization is performed according to the description in Section 2.4.5. The runtime measure includes the calculation of the term weight for each term and the assignment of that weight to the term. The output of the keywords to a file is not measured here. The runtime was measured according to the description in Section 8.1.2.

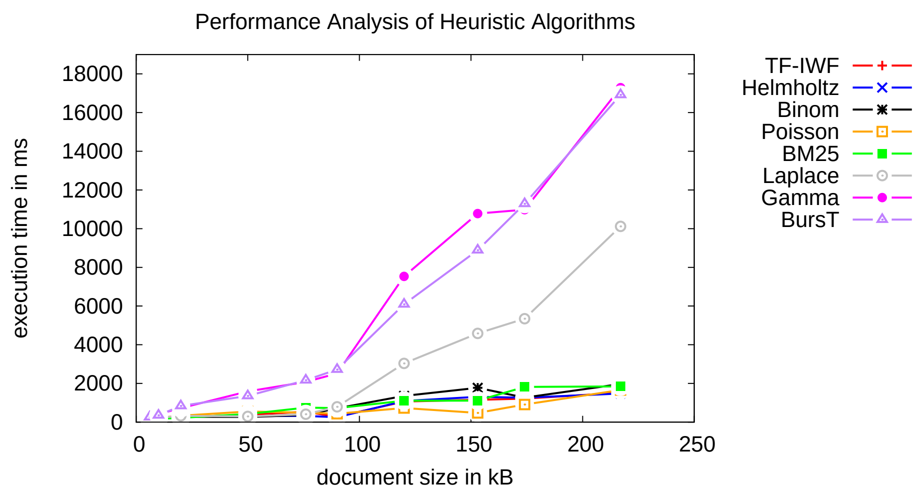


Figure 4.5: The graph shows the runtime of the described weighting algorithms for ten different documents.

Figure 4.5 shows the results of this analysis. The runtime clearly increases with the size of the documents that have been analyzed. The values of the runtime range within a few seconds and allow for scenarios, where recalculation is needed in a short time. Nevertheless, the absolute values are not very indicative as the software is not optimized for runtime performance. Additionally, the heuristics modeling burstiness seem to run longer with increasing document size. The reason for that is related to the current implementation.

The calculation of the term weight for $\mathcal{W}_{Laplace}$ is rather quick, but each time a weight for a term t is calculated, it is assigned to all occurrences of t in d. This process could be optimized in a future implementation. Furthermore, the amount of processing for $\mathcal{W}_{BS}$ and $\mathcal{W}_{\Gamma}$ grows with the number of new terms and also with the number of occurrences of a term.

Memory Consumption

I measured the memory consumption of the Java application, including all operations described in Section 4.1. The eight algorithms analyzed two documents of different size. Figure 4.6 shows the results of this measurement:

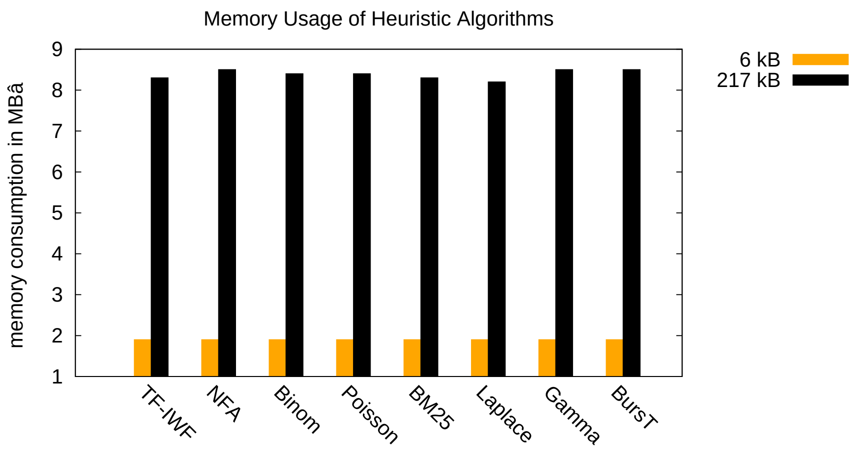


Figure 4.6: The graph shows the memory consumption of all algorithms during an analysis of two documents of different size.

All algorithms had a very similar memory consumption with marginal differences. A 36-times larger document requires a 4.5-times larger memory for the application.

In summary, the runtime of all presented algorithms increases with document size but allows for quick recalculation as it does not exceed a few seconds. The memory consumption behaves similarly as it also increases with document size but does not exceed 9 Megabyte (MB) for the chosen samples.

Chapter 5

Combination of Heuristic Measures

At present, a large number of different term weighting algorithms are used in various IR-scenarios. However, there is no universal algorithm that outperforms all the others. In Section 4.2, I stated that no retrieval algorithm meets all of the formal constraints. Therefore, I decided to combine multiple (≥ 2) heuristics and evaluate different combination methods with a selection of weighting algorithms. I can show that the formal retrieval constraints allow for the combination of a successful retrieval algorithm.

A combination can be realized by either mathematically combining the functions of the weighting algorithms or combining the generated term weights instead. Here, I introduce and analyze both methods. The combination of data from multiple sources or systems is generally defined as *data fusion* [81]. In the specific application for keyword extraction, it refers to the merging of term weights of multiple term weighting algorithms. I chose five of the most successful combination approaches of three different types to combine the algorithms described in Chapter 4. First, I introduce the Divergence from Randomness approach in Section 5.1. In Section 5.2 I review three different ranking aggregation methods and in Section 5.3 I present a combination approach based on PCA.

5.1 The Divergence from Randomness Framework

Amati proposed a probabilistic framework that can be interpreted as a language model (see Section 2.4.4) as well as a term weighting formula consisting of two components [4]. He states that a proper term weight must combine the *eliteness* and the *randomness* of a term in a distribution and and therefore functionally combines the *informative content* and *the aftereffect of future sampling* to generate term weights [5]. A similar approach is presented in [76] where the authors define a weight, based on *burstiness* and *term frequency.*

The Divergence from Randomness framework combines two heuristics as a parameter-free product of term weights. After combination, terms with a weight

$\mathcal{W} > 0$ are considered keywords. Since the aftereffect of future sampling is closely related to bursty behavior, I select heuristics for the second component inf_2 accordingly.

5.1.1 Improving Retrieval Performance

The Helmholtz approach (Section 4.3.4) is language-independent, performs with a quick runtime, and exhibits meaningful keywords when applied to various kinds of documents though it does not consider the window length and the window frequency. Furthermore, the automatic removal of stop words is a valuable criterion for preference of the Helmholtz algorithm over a comparable algorithm such as TF-IDF. Hence, I aimed to combine this algorithm with a second algorithm in order to compensate for these deficiencies and generate a well performing algorithm that eliminates the need for automatic stop word removal. This approach has already been published in [19].

The algorithm is composed as follows: I chose the combination of $\mathcal{W}_{NFA}$ with the Laplace law of succession, based on the Divergence from Randomness Framework. The Laplace law of succession models burstiness to some degree and it performs faster than the two other algorithms that model burstiness as shown in Section 4.6. The runtime is important for that approach because it is supposed to work in real-time scenario. The term weight based on that combination can then be calculated based on the following equation:

$$\mathcal{W} = \frac{1}{x_t^w + 1} \cdot (-\log NFA_t) \tag{5.1}$$

In Equation 5.1 the number of occurrences of t depends on the length of the window. In order to compensate for different window lengths, I rescale the window length with the decreasing function presented in Section 2.4.5 on the window length. The occurrences of a term t in the window x_t^w are replaced with the normalized number of occurrences xn_t^w. This normalization allows the analysis of documents with windows of different size because it ensures that term occurrences in larger windows are not erroneously weighted higher than term occurrences in smaller windows. The evaluation of this approach is presented in Chapter 8.

5.2 Ranking Aggregation Methods

All term weighting heuristics provide a weighted list of n terms that can be transformed into a ranked list of potential keywords, whereas the top ranked terms may be selected as keywords. The ranked list of terms is an ordering of the terms according to their term weights $[\mathcal{W}(t_1) > \mathcal{W}(t_2), ..., \mathcal{W}(t_n)]$, where "$>$" is an ordering relation. The order is linear and strict. The position or rank of the term t_i is denoted as $\rho(t_i)$.

Feature ranking is a principle often used in many research areas because of its simplicity, scalability, and good empirical success [108]. Rankings are solely order-based, therefore ranking aggregation is naturally calibrated and scale-insensitive. In contrast, weight-based aggregation requires some sort of normalization (e.g.

weights between 0 and 1) but these scores may still represent different relative scales. Here, I present three different ranking aggregation techniques that combine independent term rankings in order to create a more stable ranking that meets the constraints of a successful keyword extraction algorithm (see Section 4.2). The techniques presented meet the requirements of my applications: they are simple to implement, computationally cheap, and parameter-free.

5.2.1 Minimum Ranking Method

The Minimum Ranking Method was proposed by Louloudis et al. [81], and despite its simplicity it outperformed most of the state-of-the-art ranking aggregation methods. Consider a sequence of terms that has been weighted by three different algorithms $\mathcal{W}_1, ..., \mathcal{W}_3$. The weights have been converted into three ranked lists of terms $\rho_1, ..., \rho_3$. The results of this procedure are depicted in Table 5.1 for a selection of four terms.

term	ρ_1	ρ_2	ρ_3
“they”	5	3	4
“focus”	3	1	2
“mission”	8	9	11
“hand”	2	7	4

Table 5.1: A selection of four terms and their ranks, generated from the weights of three different weighting algorithms.

The final ranking score of a term t is the minimum rank position on **all** the different retrieval rankings $\rho_1, ..., \rho_m$:

$$\rho_{min}(t) = \min_{0 \leq k \leq m} (\rho_k(t))$$

In the case of the examples in Table 5.1, the minimum rank ρ_{min} for the term “they” is 3, for “focus” $\rho_{min} = 1$, for “mission” $\rho_{min} = 8$, and for “hand” $\rho_{min} = 2$. The implementation of this algorithm is extremely simple and its execution time is quite short. Nevertheless, it requires the sorting of all ranks for all terms. In my reference implementation (see Appendix A.1), I utilize the sorting algorithm *merge sort* with a complexity of $\mathcal{O}(n \log n)$.

5.2.2 Borda Count

The Borda Count (BC) determines the rank of a term t by the *number of points* according to its position in the different retrieval rankings $\rho_1..\rho_m$. The term with the highest term weight receives the highest number of points, which is $(n-1)$ – the maximum number of rank positions decreased by one. The final rank of t represents its mean position over all the input rankings $\rho_1..\rho_m$ [108; 81].

$$score_{BC}(t) = \sum_{k=1}^{m} (n - \rho_k(t))$$

Alternatively, the the terms can be assigned a score equal to the number of terms ranked lower than them. Assume a maximum number of rank positions $n = 15$

for the sample terms shown in Table 5.1. As a result the term "they" would be ranked according to the BC-score as follows:

$$score_{BC}(\text{"they"}) = 10 + 12 + 11 = 33$$

A large number of variants of the original BC ranking aggregation method exist. At present, derivates of this method are used to determine winners in elections with preference lists. All candidates on the list have to be ranked according to the preference of the voter. Such lists are used in political elections in several countries including Slovenia and small Pacific republics as well as for sports events and public competitions such as the Eurovision Song Contest [113; 147].

5.2.3 Schulze Method

The Schulze method is an election method, that has been presented by Markus Schulze in 1997 and it is based on the *Condorcet* method [122; 108]. A Condorcet algorithm performs pairwise comparisons of the input ranks of all candidates (here: terms $t_1..t_n$). The winner of a Condorcet method is the candidate (only one winner!) which is preferred over all other candidates – the candidate that wins most of the pairwise comparisons.

The Schulze method first counts how many times the rank of term t_i ranks higher than the rank of t_j and vice versa across all rankings $\rho_1, ..., \rho_m$. If t_i ranks higher than t_j in one comparison, then t_i wins this comparison, and the number of wins $d[t_i, t_j]$ is increased by one. The results of all comparisons can be stored in an $n \times n$ matrix or in a directed graph, where the terms represent the nodes. If $d[t_i, t_j] > d[t_j, t_i]$ (t_i defeats t_j) I draw an edge from the node t_i to the node t_j. The output of the Schulze method is then determined by computing the *strongest path* between all candidate pairs in the graph [122]. The *strength* p of a path between the nodes t_i and t_j is an ordered set of candidates $[C(1), ..., C(n)]$ with the following properties:

1. $t_i = C(1)$ and $t_j = C(n)$
2. $\forall k \in \{1, ..., n-1\}$:

   $d[C(k), C(k+1)] > d[C(k+1), C(k)]$
3. $\forall k \in \{1, ..., n-1\}$:

   $d[C(k), C(k+1)] \geq p$

The strength p of a path between two candidates t_i and t_j means, that there are at least p voters who prefer t_i over t_j. If there does not exist a path between the nodes t_i and t_j, the *strongest path* equals zero. Otherwise, the strongest path $p[t_i, t_j]$ represents the maximum value, such that there is a path of the accumulated number of wins from t_i to t_j. Term t_i wins over t_j, if $p[t_i, t_j] > p[t_j, t_i]$. The term t_i is a potential *Schulze winner* iff $p[t_i, t_j] > p[t_j, t_i]$ for every other term t_j:

$$t_i \ wins \iff p[t_i, t_j] \geq p[t_j, t_i] \text{ for every other candidate } t_j$$

The Schulze method is a single-winner election method. The winner term might not be top-ranked by a single input ranking but may be the most preferred term of all the rankings. In the case of ties, Schulze proposed a tie breaking method in his paper [122].

The Schulze method can be efficiently implemented with the Floyd-Warshall algorithm reducing its complexity to $\mathcal{O}(n^3)$, where n is the number of terms. This method is widely used by organizations including Debian, Ubuntu and the German political party Pirate Party Germany for the determination of candidate rankings on election lists. If several candidates have to determined for a ranked list, the last winner is taken out of the list and the Schulze method is applied again.

5.3 Principal Component Analysis

PCA is a statistical technique used to structure data and identify patterns in a multivariate dataset. It is used to highlight similarities and differences between variables in a dataset. An orthogonal transformation is used to reduce a highly dimensional dataset of possibly correlated variables into low dimensional data without a significant loss of information. PCA has been introduced by the English mathematician Karl Pearson in 1901 [103] and extended by Hotelling in 1933 [61].

My motivation to introduce PCA is twofold: I will first analyze this method for term weight combination, also used by Li et al [77], and compare it to other combination approaches in Section 8.4, and then I will demonstrate how PCA can provide valuable information for the selection of algorithms for combination.

The algorithms presented in Chapter 4 generate term weights for each term $t_1..t_n$ in the analyzed document. To generate a single list of ranked terms – potential keywords – the term weights generated by m different algorithms for each $t_1..t_n$ have to be combined. The resulting lists of term weights of the m different algorithms applied to d are the m different data *dimensions* $\delta_1..\delta_m$. Li et al. [77] used PCA to combine the term weights for each term $t_i \in \{t_1..t_n\}$ and dimension to a final weight a_i for the term t_i. His approach is utilized in Section 5.3.1.

To perform PCA, the mean $\bar{\delta}$ is subtracted from each single value – the mean is the average across the terms $t_1..t_n$ of the corresponding dimension:

$$\bar{\delta_j} = \frac{\sum_{i=1}^{n} \delta_{j,t_i}}{n}$$

Term t_i in dimension δ_j is assigned the value $a_{ij} = \delta_{j,t_i} - \bar{\delta_j}$, whereas δ_{j,t_i} is the weight of t_i in dimension δ_j. The resulting data set has a mean of zero and the mean-adjusted dimensions can be written as matrix A:

$$A = \begin{pmatrix} a_{11} & \dots & a_{1j} & \dots & a_{1n} \\ \vdots & & \vdots & & \vdots \\ a_{i1} & \dots & a_{ij} & \dots & a_{in} \\ \vdots & & \vdots & & \vdots \\ a_{m1} & \dots & a_{mj} & \dots & a_{mn} \end{pmatrix}$$

Matrix A is made up of m rows and n columns with $m \ll n$. The rows of the matrix represent the weights for each algorithm and the columns represent the weights for each individual term.

In order to determine how much the term weights of the m dimensions $\delta_1..\delta_m$ vary from their mean with respect to each other, a covariance matrix has to be created:

$$C^{m \times m} = (c_{i,j}, c_{i,j} = cov(\delta_i, \delta_j))$$

Covariance is a measure between two random variables that analyzes their behavior with respect to each other:

$$cov(\delta_j, \delta_k) = \frac{\sum_{i=1}^{n}(\delta_{j,t_i} - \bar{\delta_j})(\delta_{k,t_i} - \bar{\delta_k})}{n-1} = \frac{\sum_{i=1}^{n} a_{ij}a_{ik}}{n1}$$

If the covariance is positive, an increasing term weight of δ_j corresponds with an increasing term weight of δ_k and decreasing term weight of δ_j corresponds with a decreasing term weight of δ_k. The dimensions δ_j and δ_k increase and decrease together. If the covariance is negative, an increase of the values of δ_j means a decrease of the weights of δ_k. If there is no relation between the dimensions, the covariance is zero. In the field of IR from single documents a positive covariance translates: if a term is weighted high with algorithm j, it should also be weighted high with algorithm k.

The covariance matrix is a square $(m \times m)$ matrix that is symmetrical about the main diagonal. The entries of $C^{m \times m}$ capture the covariance between all possible pairs of dimensions: the entry $c_{i,j} = c_{j,i}$ contains the covariance between dimension δ_i and δ_j and the diagonal entries $i = j$ equal the variance of the corresponding dimension.

The covariance matrix C_A can also be written as

$$C_A = \frac{1}{n}AA^T$$

because A consists entirely of real numbers. The analyzed document is a sample of the entire population of documents. Therefore, the sample covariance can be an unbiased estimator of the covariance matrix of the matrix A and is $\frac{1}{n-1}AA^T$. The denominator changes to $n-1$ du to a variant of Bessel's correction [63; 126].

The covariance matrix measures the strength of linear relation and minimizes redundancy. Any data contains noise and redundancy. No valuable information can be extracted from noise, whereas redundancy might indicate the same information captured by different keyword extraction algorithms. In order to further minimize the redundancy, measured by the magnitude of the covariance, and to maximize the signal of the variance (the diagonal entries of C_A), the matrix C_A has to be diagonalized [126]. The matrix C_A is diagonalizable if it exists an invertible matrix Q, such that QC_AQ^T is a diagonal matrix. The following describes how this leads us to the eigenvectors and eigenvalues of C_A.

The motivation for doing this is to identify the most important characteristics of the data by transforming it into independent new variables sorted by variance – the components. The eigenvectors provide the new axis of the dataset and define the orientation, whereas the eigenvalues express their significance. The components represent the new values of data in each direction and they are usually very significant characteristics of the data. The new linearly uncorrelated components allow for a new interpretation of the data. Maybe two IR-algorithms show the same behavior so that a combination of them is not beneficial. The following procedure is able to detect that redundancy.

A good way to diagonalize the sample covariance matrix C_A is PCA, because it is a simple, non-parametric method. The procedure acts as a generalized rotation of data in order to align it with the components.

PCA can be performed in different ways. The following version is the standard method for PCA. Another approach is described in Section 5.4. The $m \times n$ matrix A contains m dimensions and n term samples for each dimension. The goal is to find an orthogonal transformation matrix Q so that $B = QA$ such that $C_B = \frac{1}{n}BB^T$ is a diagonal matrix. The following mathematical conversion shows that Q is the matrix of eigenvectors of A because it is able to diagonalize C_A.

$$\begin{aligned} C_B &= \frac{1}{n}BB^T \\ &= \frac{1}{n}(QA)(QA)^T \\ &= \frac{1}{n}QAA^TQ^T \\ &= \frac{1}{n}Q(AA^T)Q^T \\ &= Q\frac{1}{n}(AA^T)Q^T \\ C_B &= QC_AQ^T \end{aligned}$$

Furthermore, it is important to state that any symmetric matrix C can be diagonalized by an orthogonal matrix of its eigenvectors – *eigen-decomposition*:

$$C = Q\Lambda Q^T$$

C can be factorized in a column-eigenvector matrix Q, where the column q_i is i-th eigenvector of C and the diagonal matrix λ containing the eigenvalues $\lambda_1, \ldots, \lambda_n$ as its diagonal elements. Given that C_B is the diagonal matrix containing the real eigenvalues of C_A, the columns of the orthogonal eigenvector matrix Q^T contain the eigenvectors of C_A and subsequently the rows of Q contain the eigenvectors of C_A. The eigenvalues and eigenvectors are ordered and paired.

$$\begin{aligned} C_B &= QC_AQ^T \\ &= Q(Q^T\Lambda Q)Q^T \\ &= QQ^T\Lambda QQ^T \\ &= (QQ^T)\Lambda(QQ^T) \\ C_B &= \Lambda \end{aligned}$$

The equation above shows that Q diagonalizes C_B, whereas the eigenvalues of

C_B form a $m \times m$ diagonal matrix Λ with $\lambda_1..\lambda_m$:

$$\begin{pmatrix} \lambda_1 & 0 & \dots & 0 \\ 0 & \lambda_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \lambda_n \end{pmatrix}$$

The eigenvalues $\lambda_1..\lambda_m$ in descending order account for the amount of variation in the components (eigenvectors) of A.

The PCA procedure is defined in such a way that the first of those resulting components comprises the largest amount of variance – the *principal component* of the data set. Each subsequent component has the next highest variance and is orthogonal to all preceding components. An advantage of PCA is that these discovered patterns allow for dimension reduction, e.g. if three out of four components already reveal the internal structure of the data to a sufficient degree, the dimensionality of the data can be reduced to only the first three components.

Three major assumptions are made with PCA to derive good results [126]:

- *Linearity*
  Linearity is the basis of any linear transformation involved in PCA.
- *Large variances represent important structure*
  This basic assumption relates to the assumption that components with large variance represent properties of the text that were successfully identified by an algorithm.
- *Orthogonality of the principal components*
  This assumption simplifies the PCA decomposition and allows for the solution presented in the following paragraphs.

5.3.1 PCA-based weighting

Li et al. [77] took the eigenvalues into account to determine the *contribution rate* α:

$$\alpha_i = \frac{\lambda_i}{\sum_{k=1}^{m} \lambda_k}$$

The contribution rate α is then used to create a weighted combination of keyword weights for terms $t_1 \dots t_n$ in the dimensions $\delta_1 \dots \delta_m$:

$$weight_{t_j} = \alpha_1 a_{1,j} + \alpha_2 a_{2,j} + \dots + \alpha_m a_{m,j}$$

This approach accounts for **all** eigenvalues $\lambda_1..\lambda_m$ and does not exclude any components. The goal of this approach is not to reduce dimensionality but rather to achieve a weighted combination of the dimensions based on the variance in the result space. It does not seem plausible to multiply the contribution rate with the original data set. Li et al. apply their approach to several keyword "properties", these include: *word length, contribution rate, stop word rating, word frequency* [77]. They extract keywords from 300 chosen samples of three different groups and and received high precision ($> 80\%$) for the top 20 keywords of each sample. However, the authors do not provide details about their evaluation or the exact description of their test samples, nor do they explain how the precision has been measured.

The PCA-based combination of term weights appears promising because it is a novel way to combine term weights. Therefore I used it to combine weighting algorithms presented in Chapter 4 according to the formal retrieval constraints presented in Section 4.2. As the evaluation of such an approach is a very challenging and subjective task, the results in this work might differ from the results in the paper of Li et al. [77]. In order to analyze this approach in detail, further evaluations have been performed. I provide an evaluation of this approach in Chapter 8.

5.4 The relation between Principal Component Analysis and Singular Value Decomposition

PCA is closely related to SVD. Both eigenvalue methods are able to reduce a high dimensional dataset to a dataset with fewer dimensions while retaining valuable information. For an implementation of PCA, it is important to understand the relationship between PCA and SVD. This Section refers to the procedure of PCA as it is described in Section 5.3 and presents an alternative and more general method for the calculation of PCA.

By definition of SVD [138], any real $m \times n$ matrix A of rank r can be decomposed as follows:

$$A = U \Sigma V^T$$

with the $m \times m$ matrix U denominating the *left-singular vectors* and the $n \times n$ matrix V^T denominating the *right-singular vectors*. The diagonal $m \times n$ matrix Σ contains the ordered singular values $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_r$ on the diagonal axis.

$$\begin{pmatrix} \sigma_1 & & & & & & \\ & \sigma_2 & & & \Large 0 & & \\ & & \ddots & & & & \\ & & & \sigma_r & & & \\ & & & & 0 & & \\ & & \Large 0 & & & \ddots & \\ & & & & & & 0 \end{pmatrix}$$

The singular values σ equal the square root of the non-zero eigenvalues of AA^T

and A^TA. The covariance matrix C_A of A is then defined as:

$$\begin{aligned} C_A &= \frac{1}{n}AA^T \\ &= \frac{1}{n}U\Sigma V^T(U\Sigma V^T)^T \\ &= \frac{1}{n}U\Sigma V^T V\Sigma^T U^T \\ &= \frac{1}{n}U\Sigma^2 U^T \\ &= U\Lambda U^T \end{aligned}$$

Since V is an orthogonal matrix, its transpose equals its inverse and entails $VV^T = I$. It can be observed that the positive square roots of the eigenvalues Λ equal the *singular values* Σ:

$$\lambda_i = \frac{1}{n}\sigma_i^2$$

Since Λ is the diagonalized eigenvalue matrix of C_A, it is known that $\Lambda \equiv C_B$ and therewith:

$$\begin{aligned} C_B &= U^T C_A U && (5.2) \\ &= QC_AQ^T && (5.3) \end{aligned}$$

Equation 5.2 and 5.3 show that the column eigenvector matrix Q is equivalent to the transposed left-singular vectors U^T. Consequently, SVD can be used to calculate the eigenvectors and eigenvalues for PCA without the construction of a large covariance matrix. Since the dimensions of A can be very large, the SVD-based calculation method of the eigenvectors and the eigenvalues is the preferred method. The programming language R use the SVD computation as their standard technique.

Chapter 6

Structural Information in Textual Data

Some people say that politicians talk a lot but essentially they say nothing. What they really mean is: they use a lot of words but provide very little of information. Conversely, very short messages in the form of abbreviations such as “IMHO u r gr8!”, or “tl;dr” contain only a few characters and may convey a lot of information. Even languages may vary in the size of their expressions, when describing the same facts. For example the Chinese proverb

揠苗助长。

can be written in simplified Chinese (Pinyin) as

Yà-miáo-zhù-zhǎng.

In English, this sentence is translated as:

You won't help the new plants grow by pulling them up higher.

The Chinese language appears to be very concise and a few words in Chinese may convey much more information than a few English or German words. Is it possible to measure the amount of information a sentence or a sequence of elements contains? Is it even possible to distinguish between paragraphs that convey little information and paragraphs that convey a lot of information?

In Chapter 4.4 I have described the notion of burstiness in the context of word ranking. Burstiness is often based on term order of single terms and accounts for entropic properties in a text. In this chapter, I introduce a novel method that accounts for statistical properties of a whole document or text stream. I developed this algorithm to identify information structures that can possibly be used to segment the document subsequently (see Chapter 2.3) to conduct keyword extraction afterwards. This new algorithm is language independent and it can be applied to single documents as well as text streams of indefinite length.

First, I introduce the basic concepts behind this algorithm. In Section 6.2 the notion of entropy is introduced as the fundamental measure of information for texts. In Section 6.3 Grassbergers concept of match lengths is described. These match lengths form the theoretical basis for the well known LZ77 compression algorithm, which is described in Section 6.4. In Chapter 7 I utilize the LZ77-algorithm for CPD in texts and text streams.

6.1 Basic Idea

Suppose you want to model statistical data from an experiment or that appears during a time series. You would have to assume stochastic homogeneity or at least identify sequential homogenous passages that can be modeled individually. The detection of statistical changes in the probability distribution is an essential criterion for modeling a stochastic process properly. Brodsky et al. define CPD as: "the problem of detection of changes in the characteristics of random sequences..." [21]. A more formal definition of a change-point is provided by Girón et al. [50]:

Definition 6.1.1. *Let X be an observed sequence of conditionally ordered random variables with the outcomes $x_1, ..., x_n$. The sequence has a change-point at t if it has a probability distribution function $P_a(X)$ for $i \leq t$ and $P_b(X)$ for $i > r$, and $P_a(X)$ is different from $P_b(X)$.*

In this chapter I apply CPD to written texts in order to identify structural changes within documents or text streams. This application is based on the fact that human language is the main contribution of mankind to evolution that allows the exchange of complex information. Human languages can be learned by the human brain and can also be analyzed structurally and mathematically [97]. Written human text consists of sequences that appear at certain different levels: Characters form phonemes, which combine with other phonemes to words, word sequences create sentences and sentences form paragraphs. All these sequences are methodologically assembled in a way to encode information and are not random collections of data. Finite rules specify the grammar of a language and provide the framework for organized patterns that permit for individuality and versatility in communication. Whereas individual languages follow different grammatical rules, the structural, and mathematical principles remain similar [97].

Johnson considered written text as the output of a data source drawing a string of symbols from a finite alphabet $\mathcal{A}$ [62]. Based on that assumption, I consider written text a piecewise stationary source of data and determine the positions at which the source model changes its probability distribution. My assumption is that the detection of statistical changes within a text may reveal valuable information about its structure or its composition.

A number of different approaches exist to analyze the statistical properties of a text. Zipf's law captures the distribution of term frequencies at the first organizational level in a text. It articulates that terms are inversely proportional to some power of their ranking in a frequency table (see Section 4.3.2). This correlation is a key for many information extraction approaches, but it does not consider the **word order** in a text – which can be considered a second organizational level in a text. Since word order plays a crucial part of information composition of a text, it is necessary to take it into account when analyzing the information structure of a text.

Montemurro et al. showed that the degree of word order in a random text differs significantly from the degree of word order in written human texts [93]. His approach is based on entropy – a measure of uncertainty of a random variable. The Shannon Entropy [124], however, is a measure of order and the expected value of information in any symbolic sequence. Consequently, the Shannon entropy contains information about the degree of order in a text.

Based on this assumption, I developed a novel, language-independent CPD algorithm, that can detect statistical changes in texts. The fundamental principle of this algorithm is based on word order in a text and can be an estimator for entropy. This algorithm is language-independent, fast, and applicable to single documents as well as to indefinite text data streams. For this approach, words are considered the most elementary units of written information. In order to measure structural changes in a text, a trie-based data structure is introduced in Chapter 7.2. It represents the entropic properties of a written text at any one time. My method allows for constant analysis and detection of information change of a text stream instantly and continuously. The experimental results of this approach and a comparison to a similar method are presented in Chapter 8.6. The application requires a few preprocessing steps that are described in Chapter 2.2.2. To avoid language dependencies only minimal preprocessing of the text was performed.

6.2 Entropy – An Information Measure

In this section, I introduce the concept of entropy which reveals information about the distribution of words in a written text. The fundamentals of the concept of entropy go back to the 19th century when the German physicist Rudolf Clausius provided a first definition in the field of thermodynamics where he and his colleagues studied the dissipation of energy in a thermodynamic system [26]. Entropy measures the amount of disorder in a physical system. A trivial analogy for entropy is the physical form of water: Hot water can be considered in a state of high entropy because the molecules are moving randomly with a high kinetic energy. In contrast, the water molecules in ice are arranged in fixed positions within the crystal structure of ice and move only very little – ice represents a state of low entropy. Boltzmann later investigated statistical systems and formulated the probability equation $S = k \cdot \log W$ for entropy S, where k denotes Boltzmann's constant and W the number of microstates consistent with the given macrostate. Gibbs formula considers thermodynamic systems with microstates of different probabilities: $S = -k \sum_{i=1}^{W} p_i \log p_i$ [88]. Based on this concept of entropy, Claude Shannon established an entropy measure for information theory [124]. How does the entropy definition in physics relate to language?

A random sequence of words resembles the example of the hot water as it is chaotic and unordered. Shannon recognized, that in a random text sequence it is difficult to predict the next word, such as:

"carries heart man in world what A in sees his the he."

The same words compose one of the most famous quotes of Goethes Faust (First Part), when they follow the structure of the English language:

"A man sees in the world what he carries in his heart. [155]"

Shannon noted that a random text with high entropy contains a lot of information. This does not seem very logical as this means that the second sentence of the examples given above contains less information than the first. In fact, Shannon's definition is a of mathematical nature which differs from the intuitive definition of information. Shannon states that a sequence of elements that is entirely predictable does not convey a lot of information. A more random sequence has a higher information content since it is unexpected.

Shannon performed experiments with people, asking them to predict letters in the English texts. If most of the people were able to predict a certain letter, the letter would be marked as predictable, otherwise less predictable. This experiment allowed him to measure the probability distribution of the characters of the English language and consequently the entropy of the English language.

Closely related to the definition of Shannon entropy is the *Kolmogorov complexity*, expressing that the information content of a string s being the size of the smallest nonhalting program that produces s as its output. This algorithmic counterpart of Shannon entropy was presented by Kolmogorov in 1965 [69] and can be interpreted as algorithmic information of s. The Kolmogorov complexity is computer independent and of fundamental importance in information theory. The Kolmogorov complexity is not the focus of this work.

It is generally difficult to capture the amount of information contained in a text, but it is obvious that it is somehow related to the distribution of characters in a text: For example, if one symbol appears almost all the time in a text, the text probably does not contain a lot of information. Different characters do not appear with the different probabilities in natural languages. There have been studies on English language that proved that vowels appear generally more frequently than consonants [78]. Figure 6.1 shows letter frequencies for the English [109] and the German [12] language. The numbers of the English frequencies are based on the letters occurring in the main entries of the Concise Oxford English Dictionary [131]. The numbers for the German character frequencies in the figure also contain the umlaut characters *ä, ö, ü* which are treated as *ae, oe, ue* [12]. The difference between uppercase and lowercase letters was ignored. These letter frequencies not just vary by language but also by author, style, and topic. In German the letter "e" has a frequency of about 17 %, whereas the least common letters "q" and "j" only appear with a frequency of about 0.2%.

Figure 6.1 shows, that the characters in our alphabet are not statistically independent and do not appear with the same probability. Shannon applied this idea with his source code theorem and showed that a character, that appears with the probability p, should be encoded by a codeword of length $\log_2 \frac{1}{p}$ bits to generate the best possible average length of lossless encoding [7]. For example, a fair coin toss can be encoded with one bit. The entropy, or *information content*, represented by a fair coin toss is one bit. A written text can then be considered

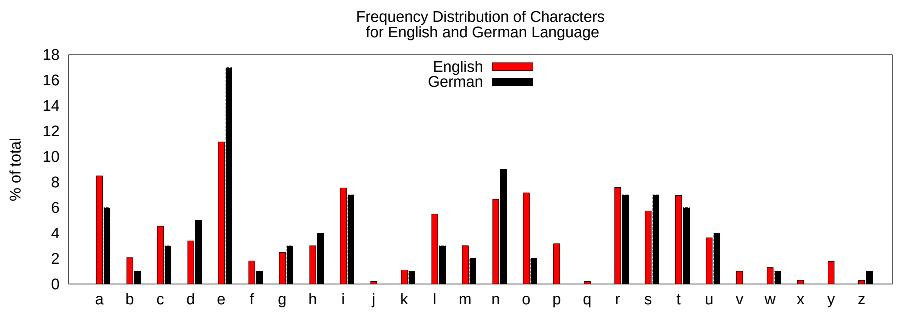


Figure 6.1: The figure shows the average usage frequencies of the characters in the alphabet in German language [12] and English language [109]. The frequencies are neither equally distributed nor is their distribution identical for the two languages.

as a statistical model with a discrete random variable X, an alphabet $\mathcal{A}$, and a probability mass function $p(x) = Pr(X = x),\ x \in \mathcal{A}$. Each symbol in the text occurs with a certain probability $p(x)$. The Shannon Entropy H can then be defined as follows [33]:

Definition 6.2.1. *The entropy $H(X)$ of a discrete random variable X is defined by*

$$H(X) = -\sum_{x \in \mathcal{A}} p(x) \log p(x) \tag{6.1}$$

The logarithm in this formula is base 2; and hence the entropy is measured in bits. The entropy $H(X)$ measures the information uncertainty or information content – amount of information in the text. If a possible outcome of an experiment is certain with probability $p(x_1) = 1$, the entropy is zero. The entropy is maximized for two outcomes $X = \{x_1, x_2\}$ if $P(x_1) = P(x_2)$, such as for the fair coin toss.

A very basic entropy estimation method is the *maximum likelihood* (also referenced as *plug-in*) estimator [48; 95]. The process X has to be stationary and ergodic. The document d contains $|d|$ terms and the term t appears x_t^d times in d. If the text is sufficiently long, the empirical probability of t is $\hat{p}(t) = \frac{x_t^d}{|d|}$. The plug-in estimator $\hat{H}(X)$ is close to $H(X)$ for a large number of samples.

The term frequencies of a large sample of N observations allow the calculation of term probabilities and the estimated entropy of the source.

The computation of entropy in a written text is impracticable with Equation 6.1 because not only is the frequency of letters and letter pairs far away from uniform, but also due to the existence of *long-range correlations* in texts [93]. Long-range correlations in language emerge when sentences cohere in a higher semantic structure beyond itself. They often originate in the fact that a text is written by one author in a unique style according to a plan [39]. In addition, the letter frequency distribution in a text is significantly affected by so called *content words* – words that have a statable lexical meaning – and their lexical composition amongst *function words* [84]. These complex dependencies in written texts, DNA sequences, or TV images lead to extremely difficult probability computations and an exponential explosion of parameters [93]. In a Markov chain you visit the states:

..., new, plants, grow, by, pulling, ...

After visiting state “grow”, a third order Markov model calculates the probability for the next term on the recent history: “new, plants, grow”. In order to predict the next term, a third order Markov model requires the probabilities of the last two states: $p(x_{i+1}|x_i, x_{i-1}, x_{i-2})$. For the 27 characters of the alphabet this leads to 19.683 states and a transition matrix with $27^4 = 531.441$ entries. These numbers point out a serious limitation of Markov models for probability estimates in written texts. Another way to estimate the entropy of a text is to compress the text with an optimal algorithm. My algorithm is based on a version of the Lempel-Ziv compression algorithm, which is a robust and highly efficient compression algorithm that converges towards the entropy of the source. In the following, I describe the algorithm in more detail. In Chapter 8.6 I evaluate the results of the structural analysis with several examples.

6.3 Grassberger’s Match Lengths

In Section 6.2 I stated that entropy is a universal measure of information content, but it is difficult to estimate the entropy within a text with Markov models. An alternative approach can be parameter-free models such as an optimal compression algorithm that converges towards the entropy of the text. The working principle of this algorithm can be explained with an easy example of two binary sequences:

1. 01010101010101010101
2. 00101101011000101011

The two sequences contain the same numbers of ten 0s and ten 1s but their composition is entirely different. The first sequence is periodic, whereas the second sequence appears to be random. In order to differentiate the two sequences, determine the length of the longest substring that starts at position 1 and appears again within the sequence. In sequence 1 the longest substring has a length of 18 bits and starts again at position 3. Whereas the longest substring starting at position 1 in sequence 2 is only 5 bits long and starts again at position 13.

1. **010101010101010101**01
2. **00101**101011000101011

In the year 1989 Grassberger defined these “match lengths” in his seminal paper [51] and showed that his match lengths converge towards entropy and were computationally superior to previous algorithms. To provide a formal definition for a written text, one has to assume the text a piecewise stationary data source X. Let X_m^n denote the finite sub-sequence $(x_m, x_{m+1}, ..., x_n)$, drawn from a finite alphabet $\mathcal{A}$. The match length for a term sequence drawn from $\mathcal{A}$ is then defined as:

$$L_i^n = L_i^n(X) = min\{L : X_i^{i+L-1} \neq X_j^{j+L-1}, \forall\ 1 \leq i, j \leq n, i \neq j\} \tag{6.2}$$

The formula above allows the calculation of the matches for the two sequences at position 1:

Sequence 1: $L_1^{20} = 19$

Sequence 2: $L_1^{20} = 6$

These match lengths were derived from LZ compression algorithm and from the paper of Wyner and Ziv who showed that the LZ data compression algorithm converges towards entropy [160]. Additionally, Shields states that match lengths can serve as an entropy estimator for a stationary ergodic source:

$$\lim_{n\to\infty} \frac{\sum_{i=1}^{n} L_i^n(X)}{n \log n} = \frac{1}{H} \tag{6.3}$$

Johnson et al. utilized Grassbergers match lengths to determine change-points in literary texts – times when the source model changes [62]. Their approach is presented in Section 7.1 and will serve as a reference approach for the comparison with my method in Chapter 8.6. Considering a text as an indefinite stream of data containing multiple change-points I utilize the widely used LZ compression algorithm for CPD [169]. A detailed description of the LZ algorithm can be found in Section 6.4.

6.4 Lempel-Ziv'77

One of the most successful compression algorithms, that takes long range correlations into account, is the LZ compression algorithm [169]. A number of estimators based on the algorithm of Lempel-Ziv have been published, showing that the compression ratio of the LZ compression can be used as an upper bound to the entropy of the stationary ergodic source H [71; 98; 161; 9]. A stochastic process is stationary if it does not change its statistical properties. If its statistical properties can be deduced from a sufficiently long sample, it is considered ergodic.

The main advantage of the LZ algorithm over the presented match lengths in Section 6.3 is its speed rather than the achieved compression rate. The LZ algorithm requires a time of the order of $\mathcal{O}(N^2)$ for N input elements [51]. The compression rate on the other hand tends asymptotically to its maximum with increasing text size. In other words: LZ does not encode a text sequence optimally but does it better with increasing size. I utilize this convergency property to detect statistical changes in a text sequence, and benefit from the performance of this algorithm that is applicable to streams [161; 9].

Here, I refer to the core of the original LZ77 algorithm, that is the base for many variations of this algorithm. LZ77 is the base for the commercial compression software *gzip, zip, pkzip,* and *winzip*.

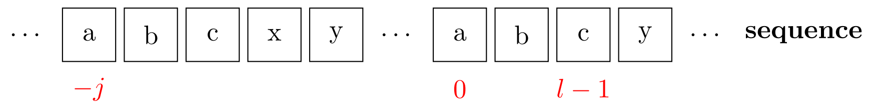
For all $n \geq 1$ let $X_1^n = (x_1, ..., x_{i-1}, x_i, x_{i+1}, ..., x_n)$ be an input sequence of length n, where x_i is the ith input element in the sequence. In the original compression-based applications, binary values are used as input elements. Here, I use characters and terms instead. The basic functionality of LZ77 can be described with the match lengths: as the algorithm reads new terms of the input sequence, it aims to identify the shortest sequence that has not been seen before [71]. Shields states in his work, that the simplest form of the algorithm can be summarized with the following sentence [125]:

The next sequence is the shortest new sequence.

Equation 6.4 contains a formal definition of the match lengths determined by the LZ77 algorithm [125; 62]:

$$\begin{gathered} L_n = 1 + max\{l : 0 \leq l \leq n, X_0^{l-1} = X_{-j}^{-j+l-1} \\ \text{for some } l \leq j \leq n\} \end{gathered} \tag{6.4}$$

The denotation l refers to the longest match with a sequence that has been seen before and L_n adds one to the longest match that starts at position 0 and does not appear in the past X_{-1}^{-n}. The small example below shows a character sequence with a match of “a,b,c” and $L_n = 4$.



With probability 1 the source entropy is the quotient of $\log n$ and $L_(X)$ as shown in Equation 6.5 [160].

$$\lim_{n \to \infty} \frac{\log n}{L_n(X)} = H \tag{6.5}$$

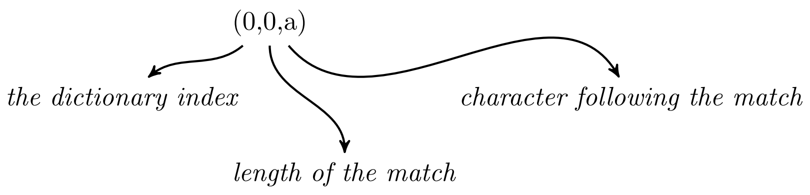
To identify new matches, the algorithm maintains a dictionary that is empty prior to the start of the compression procedure. The dictionary may vary in size and is referred to here as the *sliding window*. Then, the input is encoded continuously: to encode the **next** segment at position 0, the algorithm reads the longest contiguous sub-sequence X_0^{l-1} that comprises an entry in the dictionary and adds the next input element to it. LZ77 emits a triple, containing the dictionary index for X_0^{l-1}, the length l of the longest match, and the term X_{l-1}^l following the match. Subsequently, the new sequence X_0^l of length l is written to the dictionary.

The following example with the input stream "abrakadabra" will show the working principle of LZ77. Each character of the string is read individually.

dictionary	input stream	output triples
	abrakadabra	→ (0,0,a)
a	brakadabra	→ (0,0,b)
ab	rakadabra	→ (0,0,r)
abr	akadabra	→ (3,1,k)
abrak	adabra	→ (2,1,d)
abrakad	abra	→ (4,4,-)
abrakadabra		→

Table 6.1: The character-based compression sequence of the term "abracadabra".

At the beginning of the process, the dictionary is empty and the input stream comprises the sequence "abrakadabra". The algorithm is looking for the longest match with the dictionary entries and the input sequence. Since the dictionary is empty, no match can be found and the output triple is:



For the next two steps, the input characters cannot be found in the dictionary and the output contains no reference to the dictionary. In step 4, the input "a" can be found in the dictionary at a distance of 3 with a length of 1 – here the distance of the character behind the input stream represents the index position in the dictionary. The longest match can be found in the 6th line of the table, where the 4-character input "abra" matches the dictionary entry at position 4 and creates to the output (4,4,-). Since there is no further input, the compression is finished at that point. The compression depends on the repetition rate of the input sequence and LZ77 compresses better with increasing input.

This strategy of adding character or a term sequence to the dictionary guarantees that the dictionary already contains all prefixes of the next string. Tries are commonly used for dictionaries due to their quick modification operations and space efficiency for subsequent terms. Therefore, I implemented this dictionary as a trie data structure, which I describe in Section 7.2.

Chapter 7

Change-Point Detection for Textual Data

This chapter describes the novel trie-based CPD algorithm in detail. In Section 7.1 I present a reference algorithm, that is also based on match-lengths. In Section 7.2 I describe the trie structure, used to store the dictionary during the CPD process. Update procedures of the trie require restructuring and are presented in Section 7.3. In order to enable auto-adaptation of the algorithm and to avoid a preset window size, an adaptive window approach is suggested in Section 7.5. Finally, I provide an overview of the reference implementation for this trie-based CPD approach. Parts of this approach have been pre-published in [17; 18].

7.1 A Match-Length-Based Approach

In this section, I present Johnson's state of the art CPD algorithm [62] that is based on string matching and Grassbergers match lengths (see Section 6.3). I present this approach to complement my work because this estimator is parameter free and does not require any assumptions about the source distribution. This algorithm is particularly interesting because it is not language specific and serves as a reference for my algorithm, that is presented in Section 7.2.

Let s be a string of n characters that consists of the two substrings s_1 and s_2. The purpose of Johnson's algorithm is to detect a change-point in between those two substrings. An example could be two literary texts where one text is attached behind the other, as shown in Figure 7.1.

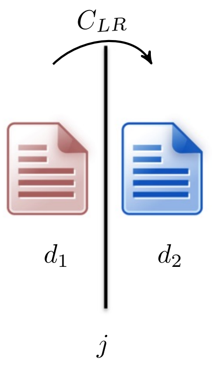


Figure 7.1: Text document d_2 is attached behind text d_1 with the change-point j between the two texts. A crossing C_{LR} is found when a substring in d_1 matches a substring in d_2.

The crossing functions C_{LR} and C_{RL} count the number of matches that cross the potential change-point j between the two texts of length n [62]. C_{RL} counts the number of *right to left* crossings of the potential change-point j and C_{LR} counts the number of *left to right* crossings respectively.

$$
\begin{aligned}
C_{LR}(j) &= \#\{k : k < j \leq T_k^n\}, \quad 0 \leq j \leq n-1 && (7.1)\\
C_{RL}(j) &= \#\{k : T_k^n < j \leq k\} && (7.2)
\end{aligned}
$$

In Equation 7.1 and 7.2 the potential change-point is at position j and T_k^n denotes the position of the longest match that equals the substring at position k. If k is left of the position j and T_k^n appears right of j, we count a left-right crossing. In case of two matches T_k^n of the same length, the one that is chosen is the match closer to j. The count of the number of elements in a set is denoted with $\#$ – the cardinality.

The idea behind this approach is: the substrings in d_1 match substrings in d_1 and the substrings in d_2 match substrings in d_2. If $k > j$ then T_k^n will tend to be $> j$ and if $k < j$ then T_k^n will probably be $< j$. The following equations are the normalized versions of the the crossing functions [62]:

$$
\begin{aligned}
\psi_{LR}(j) &= \frac{C_{LR}(j)}{n-j} - \frac{j}{n} && (7.3)\\
\psi_{RL}(j) &= \frac{C_{RL}(j)}{j} - \frac{n-j}{n} && (7.4)\\
\psi(j) &= max(\psi_{LR}(j), \psi_{RL}(j)) && (7.5)
\end{aligned}
$$

The crossing function $\psi(j)$ represents the maximum of the normalized versions of C_{RL} (7.4) and C_{LR} (7.3) [62]. The aim of Johnson et al. is to minimize the crossing function $\psi(j)$ (Equation 7.5) and to assume the change-point at its minimum. This approach provides no absolute measure of entropy but it accounts for the entropic properties of the source. The authors applied their approach to different concatenations of literary texts and showed the successful detection of change-points.

I applied my trie-based approach (see Section 7.2) to the same examples to demonstrate that my method works correctly (see Section 8.6). The problem with this approach is that its runtime increases with n as its complexity is $\mathcal{O}(n \log n)$. That leads to long runtimes for large texts and renders it inapplicable for indefinite text streams. With my algorithm, I do consider scenarios of large texts and indefinite texts streams. A runtime comparison between the two algorithms is provided in Section 8.6.3.

7.2 Trie-based Text Analysis

This section contains the core of my CPD algorithm. Here, I describe the basic structure of the trie that implements the dictionary of the LZ-algorithm – which is described in Section 6.4. The trie transformation operations required are described in Section 7.3.

The word trie is derived from the word re*trie*val by Edward Fredkin [45]. A trie is a M-ary tree data structure, which means that the number of child nodes of a node is $\leq M$ [68]. The nodes are M-place vectors with components corresponding to digits, characters, or as in this work: terms. The nodes do not store keys, instead each node on a certain level l represents the set of all keys that begin with a sequence of l components – its *prefix* [68]. The *level* of a trie is defined as the number of parent nodes a trie node has. The root node of the trie is located at level zero.

In general, tries are used to store repeated patterns because they inherit positive attributes such as fast lookup and insertion time, space efficiency, and ordered iteration. Here, I use a trie as a database for the entries of the LZ-dictionary. The motivation is to store structural information within that trie that represents the entropic property L_n of the text. The denotation L_n is described in Section 6.4. With this approach, I focus on a relative measure of entropy as the trie structure does not lead to absolute number of entropy but rather indicates changes with respect to previous states.

While a moving window slides across the text, the trie represents the term sequences within that window. The algorithm is described in the next section. In this work, the input elements are either single characters or terms, depending on the specification of the actual scenario. In other words: the trie stores n-grams, whereas each prefix is represented only once.

Trie Structure

Nodes in the trie can be classified into three categories: *root node*, *leaf node*, and *inner node*:

- The **root node** is the starting and top node of the trie.
- A **leaf** node is a node without children.
- An **inner node** is a node with at least one descendant.

The root node is can also be a leaf node if the trie is empty. The trie is built up, starting at the root node, as the sliding window processes the input text. An exemplary trie structure is depicted in Figure 7.3.

A single node of the trie is presented in Figure 7.2. The node contains the element, that has been read on the path to the node, and the birth-time of the node. The element in the node is not associated with the node itself, but rather with the path that leads to the node. Character "a" means that this node can be reached by reading character "a" in the input sequence. The detailed steps that

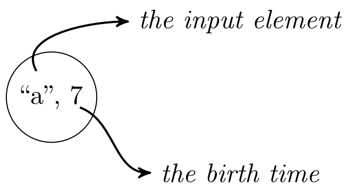


Figure 7.2: This Figure shows a single trie node.

lead to this trie are described in Algorithm 2 and depicted in Appendix A.2. Let the sample "abrakadabra" be the input sequence. The first character 'a' is read by the algorithm and added to the helper list *sequence*. Since the trie T consists only of the root node, the character 'a' is added as a child of the root node to the trie. The next character of the input sequence is 'b'. The same input procedure happens to 'b' and 'r' so that there are three nodes at level one. When the next 'a' is read and put to the helper *sequence*, it can already be found in the trie. The character 'k' is added to *sequence* and a node with character 'k' is added as a child node to the node at level one with character 'a'. The trie in Figure 7.3 represents the structure of the dictionary window after having read the input "abrakadabra".

Algorithm 2 Trie Construction Algorithm

```
Input: input ← the source text
 1: character $c$ ← ' '
 2: trie $T$ ← new empty trie
 3: trie node node ← root node of $T$
 4: for $c$ in input do {take next character of the input}
 5:   if $c$ in node.children then
 6:     node = node.getChild(c)
 7:   else
 8:     node.addChild($c$) {add new leaf node}
 9:     node = root node of $T$
10:   end if
11: end for
```

The root node in Figure 7.3 is marked with the term *root* and contains an additional numerical value that is described in Section 7.3. The paths to the children of the root node represent the character sequences as they appeared in the text. A path along the nodes starting with the root node and ending with a leaf or an inner node represents a character sequence that has been observed in the input stream. Not all sequences in the text can be observed in the trie.

The input sequence of the trie in Figure 7.3 differs slightly from the example in Section 6.4. Here, the last two trie inputs have been "ab" and "ra" instead of "abra". Accordingly, the nodes with the content "a", "b", "r" are the child-nodes of the root node and are therefore not recognized as subsequent input data for future matches. The keys are only read from the root node to the leaf nodes and not across sibling nodes at any level.

This change to the original LZ77 algorithm results in a speed gain but also means that some patterns will be dismissed (similar to LZ78). This adaption of LZ77 is necessary, but it does not have a huge impact on the performance of this algorithm for the purpose of CPD (see Section 8.6).

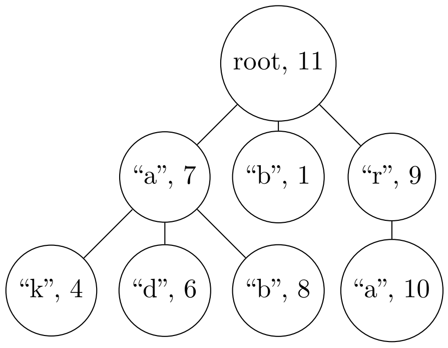


Figure 7.3: The trie structure that is created by the LZ77 algorithm after processing the input sequence "abrakadabra".

7.3 The Trie Transformation Procedure

The trie dictionary represents a sliding window, that comprises a certain amount of structural information. It it necessary to limit the amount of data analyzed at any one time and to discard terms from the dictionary after a defined period. This can be achieved with a sliding window technique. The sliding window technique requires a trie structure that can be constantly modified upon every update procedure of the dictionary. One way to update a trie is to completely rebuild the trie with each new node that is added. This requires a rebuild of the trie for each new dictionary input. Since the complete rebuild is time-consuming, I propose an alternative: a trie transformation procedure that retains the entropic properties of the text. How is that guaranteed?

In Section 6.4, I showed that LZ77 is a lossless compression algorithm that is based on match-lengths, which represent the entropic properties of a text well. The trie that I introduced in Section 7.2 stores most of these match lengths within its structure. Since not all of the matches are stored, the trie can not be used to completely reconstruct the original text as can be done with a LZ77 dictionary. The key is that Algorithm 2 does not change the order of the input sequence and thus ensures that the trie contains a certain amount of the entropic properties of the text. How much order of the text the trie represents, depends on the text itself. In the following, I present two extreme cases to emphasize the amount of order represented by the trie structure.

Another limitation is the total amount of text that is stored in the trie. A time value for each node except the root node limits the size of the trie.

Therefore, the creation time $\tau_i,\ i \geq 0$ of each individual node i is displayed next to the character in Figure 7.2 and inside the nodes in Figure 7.4. When a new character or term (as in Figure 7.4) is read from the input stream, the time value of the created node is set to the actual trie time τ. For each node inserted in the trie, the time value of the trie τ is increased by one. If a new term sequence is read and one of the existing nodes has to be updated, the time value of the node is updated as well. The node with the term "to" in Figure 7.4 contained the time value 0 before it was updated to 4 when the sequence "to be" was inserted into the trie the second time. The most recent node that has been added to this trie is the node with the term "be" because its time value is 5.

As the window progresses and slides across the text, the trie will change its structure and eventually become wider and deeper. The window size $|w|$ determines the maximum size of elements represented by the trie.

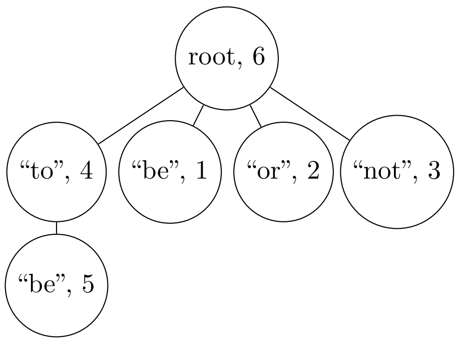


Figure 7.4: The trie structure that is created by the LZ77 algorithm after preprocessing the input sequence "To be or not to be" from Shakespeare's Hamlet.

In order to describe the composition of the trie, I depict two extreme cases:

1. the input sequence X contains a multiple of one element
2. the input does not repeat at all

Figure 7.5 contains an example for the character sequence containing 10 times the character "a". Although this sample indicates an estimated entropy of zero of the source, the trie still grows as it processes the input. However, the trie only grows in depth and the number of nodes is limited. The minimum number of nodes for input $x_1, ..., x_n$ can be calculated by the reverse of the triangular number: *number of nodes*$= \lfloor \frac{\sqrt{8n+1}-1}{2} \rfloor + 1$. Case two leads to a very flat and wide trie structure with a maximum number of nodes that equals $n + 1$.

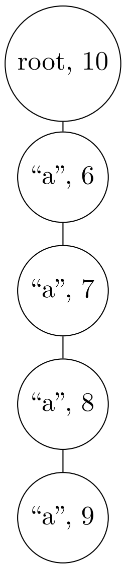


Figure 7.5: The trie structure that is created by the LZ77 algorithm after processing ten times the character ’a’.

The trie contains the most recent input elements and its structure represents the entropic properties of the input sequence, but the size $|w|$ depends on the following trie transformation procedure:

Algorithm 3 Insertion of a new symbol into the trie

```
{insert a new input element $x_i$}
if node with $x_i$ exists then
  update $\tau_i$ of the node
else
  insert a new node with element $x_i$ and creation time $\tau$
end if
$\tau = \tau + 1$
{Decay Procedure}
for all nodes of the trie do
  if node exceeds its life time then
    delete the node
  end if
end for
```

A new node can only be inserted at the correct level in the trie. Therefore the insertion procedure of a sequence of elements $y_1, ...y_m$ always starts at the root of the trie and the first element y_1 is then inserted into the level of the children of the root node, y_2 at the next level and all subsequent elements are inserted accordingly. Figure 7.5 contained the root node and three nodes with the element “a” before the sequence “aaaa” was inserted. Since the first three elements already existed as nodes, their creation time was just updated. Only for the last input character of the sequence “aaaa”, no node existed and a new node with element “a” and $\tau_i = 9$ had to be created.

If a new node has to be inserted at a certain level into the trie and other nodes already exist at that level, it is always inserted behind the existing nodes. Figure 7.4 shows that the terms “to, be, or, not” were inserted subsequently as they appeared in the input sequence “to be or not to be”.

After each input of an element, the overall time τ of the trie increases. After each increase, all the nodes of the trie have to be validated. The lifetime of a trie node is determined as follows:

$$\tau_{lifetime} = \tau_{creation} + decay(\vec{\Delta\tau})$$

$\tau_{lifetime}$ of a node limits how long that node can exist. If $\tau_{lifetime}$ of a single node has expired, that is: $\tau_{lifetime} < \tau$, the node is deleted.

The $\tau_{lifetime}$ depends on the creation time of the node and the function $decay(\vec{\Delta\tau})$. The vector $\vec{\Delta\tau}$ contains the deltas of the creation times of the node and its past instances. The following equation calculates the deltas for the creation times of a trie node other than the root node.

$$\begin{aligned}\vec{\Delta\tau} &= [\Delta\tau_1, ..., \Delta\tau_n] \\ &= [\tau_{creation}(2) - \tau_{creation}(1), ..., \tau_{creation}(n) - \tau_{creation}(n-1)]\end{aligned}$$

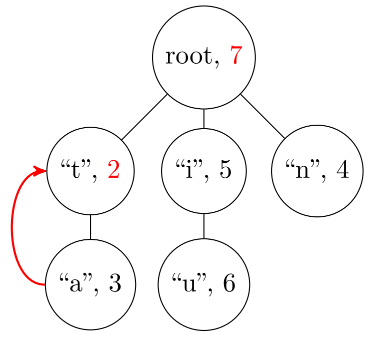
The vector $\vec{\Delta\tau}$ of the first child of the root node in Figure 7.5 contains the following values after processing the input procedure:

$$\vec{\Delta\tau} = [1, 2, 3]$$

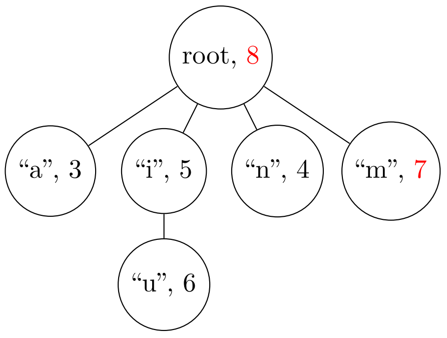
I tested different versions of $decay(\vec{\Delta\tau})$ and describe the evaluation results in Section 8.6.

The delete function for a single node is defined as follows: If $\tau_{lifetime} < \tau$ then delete the node from the trie. If this node has child nodes, insert its child nodes recursively as sub-nodes of the father of the deleted node at the position where deleted node was before. This ensures that the information of the child nodes does not get lost and the impact on the whole trie structure is minimized.

The children are inserted at the position of the father node in the same order as they were before. Figure 7.6 shows the deletion of the node with character “t” as the last letter of the input sequence “Titanium” is read by the algorithm. The value oft he function *decay* for node was 5 before it got deleted.



(a) Trie after 7 input elements are processed.



(b) Trie after 8 input elements are processed.

Figure 7.6: This figure shows the deletion procedure of the node with the character "t" as the last letter of the input sequence "Titanium" is read.

These transition operations ensure that the trie never grows indefinitely, although the input stream might be of indefinite length.

7.4 Trie-Based Change-Point Detection

The trie-based CPD is based on the structure of the trie itself as it represents the entropic properties of the text. In order to detect change-points, the following properties of the trie are measured:

- maximum depth
- total number of nodes
- total number of leaf nodes
- input position

The maximum depth of the trie represents the number of nodes along longest path from the root node to the farthest leaf node. Since the trie structure represents the entropic properties with respect to a distinct window, I expect these measures to vary over time and to indicate fluctuations in entropy as the window progresses along the stream. The total number of nodes as well as the total number of leaf nodes is expected to decrease with recurring term sequences.

The *input position* is related to the nodes at level one of the trie – the children of the root node. The children of the root node are numbered according to their position in the trie from left to right. Each node in the trie has a designated input position, that equals the input position of its parent nodes, except for the nodes at level one. The input position may change with trie transformation procedures, that are described in Section 7.3. The possible input positions of the nodes in the trie in Figure 7.7 are $1, 2$, or 3.

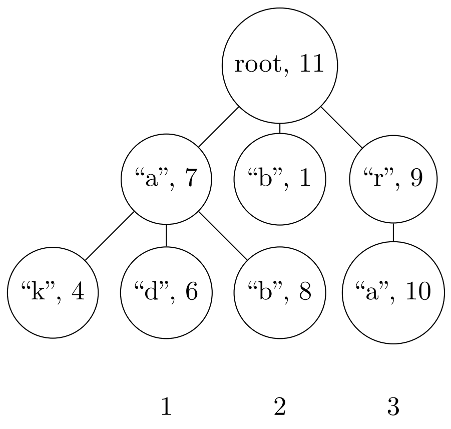


Figure 7.7: The children of the root node of the trie are numbered from 1 to 3.

Based on the trie structure, the following assumptions can be made:

- If the trie processes a single text that does not change style, language, or topic, the measures should adopt a somewhat steady state.
- As the algorithm processes new terms, that have not been seen before, the number of nodes and the number of input positions should increase.
- If the algorithm hits a change-point, where the statistical properties of the text change abruptly, measures such as total number of nodes and input position should indicate this clearly by a sudden rise.

In Section 8.6.2 I measure these properties of the trie as the algorithm slides along several sample texts. The measures are observed over time while the input text is processed and sudden changes of the values may indicate a change-point. I also describe how this method reveals some unexpected change-points in sample texts and that the input positions are the most appropriate measure to detect change-points.

7.5 Adaptive Window

Similar to the original LZ77 compression algorithm, the trie-based algorithm is based on a sliding window. The window does not contain a fixed number of elements, but it allows for the analysis of indefinite streams. A crucial parameter for the CPD is the window size $|w|$, which dictates the number of terms to be considered at any one time.

Long-range correlations can only be detected if the window size exceeds a multiple of sentence length. Only then is the likeliness of finding more sequential matches within the text increased. The window size can be considered as a parameter specifying the resolution. A larger window size accounts for more matches as well as more noise, whereas small window sizes only accounts for very serious changes of the distribution of the source. Here, I do not focus on finding an optimal value for $|w|$ as I did in 2.3 with the introduced *self-regulation approach*. I rather present an adaptive window approach, based on different properties of the text.

The performance of trie-based approach depends on the length of the document $|d|$ and the window size $|w|$. An increasing document size leads to a linear increase of the number of match lengths and also a linear increase of the number of trie operations. The trie operations *insert*, *search*, and *decay* require runtime proportional to the following:

- *insert*: $\mathcal{O}(key\ length)$
- *search*: $\mathcal{O}(key\ length)$
- *decay*: $\mathcal{O}(number\ of\ nodes)$

The operation *decay* checks for every node of the trie to determine if the lifetime of the node has expired. The number of nodes of the trie never exceeds the window size. All trie operations depend on the window size $|w|$. Consequently, the runtime for the trie operations grows slowly as the document size increases. The complexity of the trie-based change-point algorithm is $\mathcal{O}(n^2)$ as its three main operations *insert, search, decay* depend on the input length and the constant number of nodes. In case of a data stream, the algorithm can run continuously and can continuously produce results.

The *lifetime* of a node in the trie is based on the *creation time* and the function $decay(\vec{\Delta\tau})$ (see Section 7.3). The following four variants of *decay* have been realized in this work:

$$decay_1(\vec{\Delta\tau}) = |w|, \qquad |w| \in \mathbb{N} \tag{7.6}$$
$$decay_2(\vec{\Delta\tau}) = \Delta\tau_n, \qquad \Delta\tau = \Delta\tau_1, ..., \Delta\tau_n \tag{7.7}$$
$$decay_3(\vec{\Delta\tau}) = median(\vec{\Delta\tau}) \tag{7.8}$$
$$decay_4(\vec{\Delta\tau}) = min(\vec{\Delta\tau}) \tag{7.9}$$

The function $decay_1$ simply returns a fixed value $|w|$, determining the decay time of a node, whereas $decay_2$ always refers to the latest delta of this node. If there has not been updated, $decay_2$ is set to the initial value $|w|$. The versions $decay_3$ and $decay_4$ are based on the median, respectively the minimum element of vector $\Delta\tau$. The different decay functions allow for a flexible window size, based on the deltas of the appearances of the nodes. The only parameter that has to be set beforehand is the initial window size $|w|$. I conducted experiments with this adaptive window approach and present the results of these in Section 8.6.

7.6 Reference Implementation

During the development of the CPD algorithm, I developed a reference implementation to verify and evaluate the behavior of my algorithm. The implementation is based on Java, using the version Java SE 1.7. Java code is portable, allows for cross-platform development, and comes with a large collection of plug in libraries.

The CPD algorithm has been implemented iteratively and its architecture is based on two independent packages: *Entropy* and *Trie*.

In a first step, I implemented the LZ77 algorithm (compression and decompression) in the package *Entropy* in order to compare its results to the estimated entropy of sample texts. The input texts were read from UTF-8 plain texts and preprocessed according to the descriptions in Section 2.2.2. My first version of the trie-based algorithm builds up a trie that grows according to the size of the input text.

An overview of the program structure of the package *Trie* is provided by the UML class diagram in Figure 7.8. An instance of the generic class *Trie* can have only one root node, whereas the root node can have zero or more child nodes. The child nodes are identified as *INNER* or *LEAF* nodes.

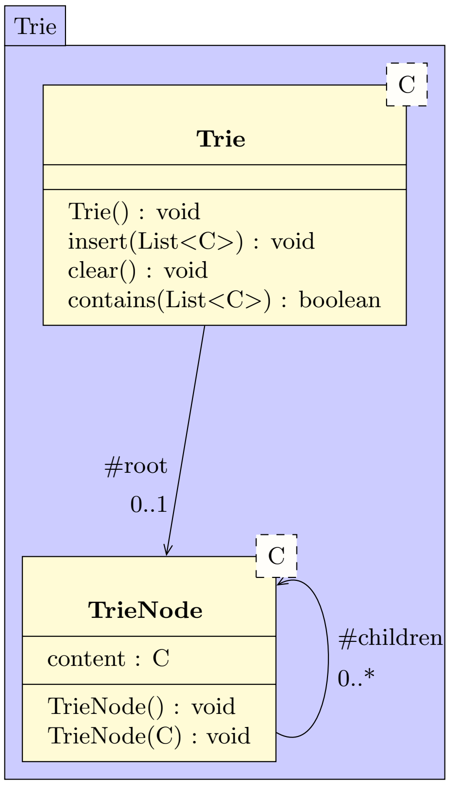


Figure 7.8: This UML Class diagram provides an overview of the structure of the reference implementation of the trie without transformation operations.

The next version of the reference implementation included the trie transformation procedures as described in Section 7.3. A more exhaustive UML class diagram for this implementation is attached to this work in Section A.3. The trie transformation procedures were implemented and realized with the classes *Trie* and *AdaptiveWindowTrieNode* – referring to the adaptive and moving window capabilities of the trie. In order to instantiate the generic trie package *TrieNodeType* with characters and terms, the class *Term* and *Preprocessor* were added to the package *Entropy*. The class *Preprocessor* extracts text from Webpages and Documents of different kinds and performs all the necessary preprocessing. It also provides information about the text, such as: the number of terms in a window as well as the number of terms in a document. The package *Entropy* contains all classes for the input data processing and CPD and compression operations. It also contains reference implementations of related projects for comparison (see Section 8.6).

Finally, visualization of the trie was necessary for debugging purposes and I chose the open source graph visualization software *Graphviz* [144] for this purpose. However, this implementation is yet a prototype and neither optimized for fast runtime nor space efficiency. It serves as a proof of concept and as a first reference for an evaluation.

Chapter 8

Evaluation

Even though the theoretical foundations of an algorithm may sound compelling, the true nature of its behavior can only be revealed throughout evaluation. In this chapter, I demonstrate the most interesting experiments that I conducted throughout this work and present their results. Some of the results in this chapter have been pre-published [19; 17; 18; 16].

In the first part of this chapter, I describe the experimental setup of my experiments in detail. Subsequently, I demonstrate how my combination of two term weighting approaches outperforms one of the most widely-used keyword extraction approaches. This example serves as a prototype. The number of possible combinations of algorithms is fairly large and may be composed according to the use case scenario. The adjustments of these combinations may differ in the number of algorithms in their kind, but also in the way they are combined. Therefore, in Section 8.4 I compare the different combination approaches that I proposed before in Section 5.

As the structure of a text may contribute to the heuristic analysis, I dedicate Section 8.6 to the proposed trie-based structural analysis. I analyze some compelling real-world examples with the trie-based CPD algorithm and demonstrate its superior performance with respect to a state-of-the-art algorithm.

8.1 Experimental Setup

This section describes the setup as well as the methodology of the test environment in detail. The experiments were conducted on commercial-off-the-shelf hardware, which is concisely described in the next section. Execution time measurements in a Java environment demand for special tools and precautions, which are briefly described in Section 8.1.2.

8.1.1 Test Environment

All experiments are conducted on a MacBook Pro furnished with a 2,66 GHz Intel Core 2 Duo Processor and 4 GB random access memory. The operating system is an OS X 10.8.5. The Java environment dates version 1.7.0_25 and the heap size is set to 1024 MB. The Java code development has been conducted with the development toolkit Eclipse integrated development environment (IDE) version Kepler (4.3).

8.1.2 Methodology

This section provides a detailed explanation of the performance evaluation applied to the methods in this work in order to allow others to better understand the results and enable them to reproduce the evaluation results presented here.

The performance evaluation of a Java application is not a trivial task. One of the challenges that comes with a Java system is its non-determinism. A Java Virtual Machine (JVM) uses Just-In-Time (JIT) compilation which may lead to different execution times of the same code. A multi-processor machine may execute different thread schedules which may also lead to different execution times and also affects the garbage collection behavior and thus the overall system performance [49]. Various system effects such as interrupts and DMA transfers may also be a source of non-determinism.

In this work, the performance of the algorithms is measured by executing relatively short running java applications. Consequently, the analysis is focused on *startup* performance of these applications rather than *steady-state* performance. A recommended "statistically rigorous performance evaluation" approach is to measure the average or mean execution time instead of the fastest or slowest execution time [49]. Therefore, the following two steps were conducted:

1. Measure the execution time of $n = 6$ Virtual Machine (VM) invocations running a single benchmark iteration.
2. Calculation of the confidence interval for the confidence level as described below.

Each application must be executed at least seven times. Since the first VM invocation will load certain data, such as libraries, persisting in memory or data persisting in disk cache, the subsequent VM invocations are not entirely independent as one would require for a well-defined mean. Therefore, the first VM invocation is being discarded and only the subsequent measures are retained for further analysis. The mean of the $n = 6$ measurements $y_i, 1 \leq i \leq 6$ can be calculated as follows:

$$\bar{y} = \frac{\sum_1^6 y_i}{6}$$

The measured execution time is the time between the specific task such as CPD or keyword extraction and the beginning of the output operations. The initialization time, I/O time, and preprocessing time is not measured because document parsing and content loading may take a serious amount of time. To accomplish that, I use *VisualVM*, a profiling software tool that provides a visual interface for Java applications that are running on JVM. VisualVM allows for a monitoring and performance analysis at method level. The measurement of the execution time is conducted according to the following procedure:

1. set breakpoint within *main*-method
2. run the application
3. wait until VisualVM has properly launched
4. when application has finished, open *Profiler* → *CPU*
5. resume the application
6. read the execution time after the app has finished

This procedure ensures that *VisualVM* is properly loaded and the execution time of the specific method can be measured correctly. During the time measurements, most other processes are inactive but operating system processes may still affect the results.

8.2 Evaluation of Keyword Extraction Approaches

The evaluation of the performance of a keyword extraction approach provides insight pertaining to its usefulness in a real time scenario. The performance of an algorithm is crucial as it may influence the response time and therefore the usability of an application. The following measures are commonly used in information retrieval to determine the quality of the retrieved data:

Compression Ratio

The research areas indexing and summarization apply keyword extraction methods to generate indices or summaries based on keywords. A substantial criterion of evaluation in these research fields is the *compression ratio* [86; 153]. The compression ratio quantifies the reduction of data that can be achieved with keyword extraction. It is possible to calculate the *compression ratio*, which depicts the savings of space obtained with the extracted keywords in contrast to the original text [153]. Equation 8.1 shows how to determine the compression ratio for a text.

$$\textit{compression ratio} = \frac{\textit{length of the summary}}{\textit{length of the full text}} \tag{8.1}$$

Meaningfulness

The second fundamental property measures the *meaningfulness* of information that is retained. The evaluation of the meaningfulness of keywords is very challenging because it is a very subjective task. Whereas a term appears to be a keyword for one reader, it might not be a meaningful term for another one.

Moreover, single terms may become meaningful only in combination with another term, or they may take up another meaning in different context. Furthermore, various definitions of meaningfulness exist: Wan and Xiao state that a meaningful term highlights the major points of a document and contributes to the summary [156], whereas the Gestalt Theory in computer vision defines meaningful events as a meaningful deviation of randomness – an event is ϵ-meaningful if the expectation of number of occurrences of this event is less than $\epsilon \leq 1$ under the a-contrario uniform random assumption [37].

Precision

I chose telling real-world examples as source documents and evaluated all the results by hand. This means, each extracted keyword was categorized into *meaningful* or *not meaningful* – keyword or no keyword. This approach allowed the use of the classic evaluation methods: *precision* and *recall*. Here, I did not consider *accuracy* (F-measure) due to the high number of negatives. Precision is the fraction of extracted keywords, that are actual keywords:

$$precision = \frac{|keywords\ extracted \cap keywords\ identified|}{|keywords\ extracted|}$$

Recall

Not all rankings exclude terms from the set of potential keywords but rather provide a ranked list of potential keywords. Hence, I selected the top ten keyword candidates for the precision determination. Recall is the fraction of extracted keywords out of all keywords in the whole document:

$$recall = \frac{|keywords\ extracted \cap keywords\ identified|}{|keywords\ identified|}$$

It is very subjective to determine the exact number of all potential keywords in a document and for a keyword extraction approach it is not necessary to identify all potential keywords in a documents. It is rather preferable if the extracted keywords comprise the topics of a text than all the meaningful words. Therefore, I did not consider the measure recall in this evaluation.

8.3 A Specific Combination Approach

In this section, I present an evaluation of the combination of two specific heuristics for keyword extraction. This combination had been described in Section 5.1.1. This particular combination is a prime example of a successful combination for a specific purpose. Here, I also show how the number of extracted keywords changes with the size of the document for this particular combination. This observation is the foundation for my self regulation approach, that regulates the window size automatically. Moreover, I evaluated the meaningfulness of the extracted keywords and I analyzed the compression ratio in dependence of the window size. Finally, I evaluated the runtime efficiency of this particular combination approach.

8.3.1 Compression Ratio and Window Size

In order to analyze the compression ratio, I performed keyword extraction with the algorithm presented in Section 4.1. My combined heuristic described in Section 5.1.1 was used to generate the term weights for all terms in the sample, with $0 < NFA_t < \epsilon$, whereas $\epsilon = 1$. The ϵ criterion allows to limit the number of extracted keywords, which is a prerequisite for a count of extracted keywords. Subsequently, I accumulated the number of extracted keywords of all windows of the single document. The analysis has been performed on more than 200 State of the Union Addresses of the Presidents of the USA individually [135; 19]. The size of the documents varied between 8 KB and 217 KB with more than 27 000 words. Furthermore, the use of language and style of writing varied for each document.

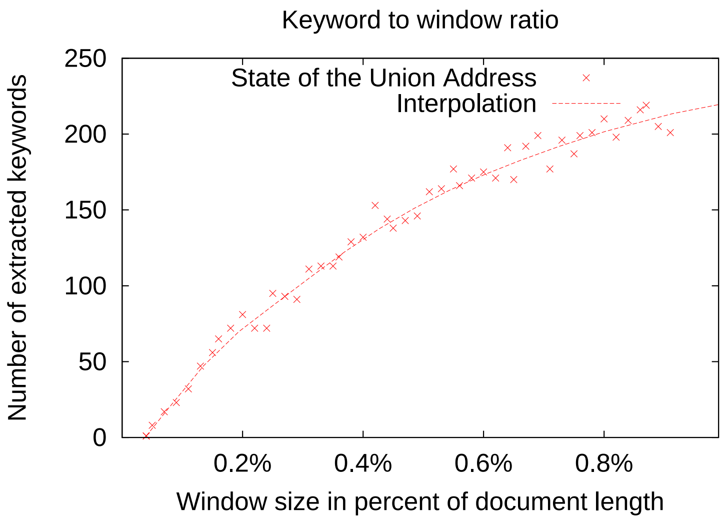


Figure 8.1: The compression ratio directly relates to the window size.

After preprocessing, I performed term weighting with NFA_t for each single term of the individual documents with a varying window size. The window sizes ranged from 0.25% up to 50% of the document length. The windows were non-overlapping and of equal size. For this experiment, I did not consider document structure. Figure 8.1 shows the results for the State of the Union Address in 2012. The graph shows the typical curve for a window size smaller than 1% of the document length. In Figure 8.1, it can be observed that the algorithm extracted about 250 potential keywords at a window size of 1% of the document length. Therefore, the compression ratio at 1% is about 4%. For the window size below 1%, the increase of extracted keywords is significant, especially for larger documents.

A window size of about 1% of the document length leads to an average of 2.5 keywords per window. This allows the user to capture the content of each window, and with that window size it is unlikely that one window comprises several subtopics of a single document. This window size seems appropriate for the chosen samples. A window size of more than 30% of the document size presumable leads to topic-overlapping windows as stated in the pre-published paper [19]. In Section 2.3, I stated that keyword extraction performs best, if each window comprises a single subtopic. Subtopic-overlapping windows are likely to affect the specificity of a term – how specific one term is to a concept than another. For this heuristic, it may lead to a reduction of the term weight for $\mathcal{W}_{NFA}$. The number of extracted keywords stagnated with a window size of more than 10% and even decreased abruptly in larger documents with a window size of more than 30% of the document length [19].

To summarize, the window size is crucial for the extraction of the appropriate number of meaningful keywords. In this case, it should be set between 1% and 30% of the document length. This is only a rough estimate and may not work for other samples as the window size also depends on the kind of literary work – the number of meaningful words differs significantly between a novel and a technical document. The CPD approach presented in Chapter 7 may help to identify a beneficial window structure. Subsequently, the self-regulation approach presented in the next Section may provide a way to determine an optimal window size.

8.3.2 Self-Regulation Approach

During my test runs with the described algorithms, I experienced large differences in number of extracted keywords per window. Some algorithms extracted more keywords from larger windows and less keywords from smaller windows. Consequently, the window size had to be fitted to a size so that a reasonable number of keywords per window was extracted. However, the number of keywords varied significantly for documents of different author, style, and topic and the window size had to be adjusted for each document to extract a reasonable number of keywords. A small window size of 20 terms may fit a short document but it does not fit a novel such as Treasure Island by Robert Louis Stevenson. A too small window size may split topics and generate not enough keywords whereas a too big window size may not satisfy the intended focus and leads to content-overlapping. To automate this process, I developed a self-regulation approach that determines the window size automatically by running the keyword extraction algorithm with different window sizes.

This algorithm can only be applied if the number of keywords positively correlates to the window size. In order to obtain a fixed number of keywords k from the window w, I introduce a self-regulating algorithm for an adaptable window size. This algorithm automatically determines the window size necessary to extract the claimed number of keywords c and does not require any further adjustments.

Algorithm 4 Bisection method for window adaption

```
1: c = claimed number of keywords
2: left = 0
3: right = maximum window size
4: found = false
5: while left $\leq$ right and not found do
6:   size = (left + right) / 2
7:   k = extracted keywords with window size size
8:   if k > c then
9:     right = size - 1
10:  else if k < c then
11:    left = size + 1
12:  else
13:    found = true
14:  end if
15: end while
```

Algorithm 4 repeatedly bisects the interval of number of terms that compose a window until the claimed number of keywords per window is reached. The algorithm performs several subsequent keyword extractions for windows of different size until an optimal window size is reached. I set the maximum window size for the initialization of the algorithm to half the number of terms in the document: $right = \frac{|D|}{2}$. The window size determined by the algorithm is only an estimate of the optimal window size to retrieve a reasonable number of keywords per window. It has proven to be extremely fast and its adaptiveness allows for analysis of documents of different size. This approach has been pre-published in [19].

8.3.3 The Meaningfulness of Extracted Keywords

In Section 8.2, I stated that it is difficult to evaluate the meaningfulness of the extracted keywords. In order to show the meaningfulness of the extracted keywords, I chose documents of different domains and different sizes with an easily deducible content. During my studies, I analyzed a number of different algorithms and combinations of algorithms. Not only did I experience varying results depending on the algorithms and their combinations, but also on the analyzed texts.

I demonstrate the meaningfulness using two examples (pre-published in [19]). First, a keyword extraction on all abstracts of all Association for Computing Machinery (ACM) Symposium on Document Engineering (DocEng) submissions has been performed. Subsequently, I have compared the extracted keywords to the author-defined keywords to estimate their meaningfulness. In the second example, I have extracted keywords from President Obama's State of the Union Address in January 2011. Here, my goal was to bring out the benefits of inner-document keyword extraction.

Title (Year)	**Author-assigned keywords**	**Extracted keywords**
Vector Graphics: From Postscript and Flash to SVG. (2001)	svg, flash, swf, pdf, postscript	svg, vector, graphics
Fast Structural Query with Application to Chinese Treebank Sentence Retrieval. (2004)	treebank, structural query, xml	pcrf, query, chinese, structural, flexible, corpus, search
Towards XML Version Control of Office Documents (2005)	version control, office applications, xml diffing	office, openoffice, diff, version, control, state-of-the-art, binary, xml, versioning, documents
A Document Engineering Environment for Clinical Guidelines. (2007)	clinical guidelines, xml, deontic operators, gem	computerization, markup, recommendations, guidelines, clinical, operators, medical
Logic-based Verification of Technical Documentation. (2009)	model checking, document verification	checker, specification, technical, documentation
Semantics-based change impact analysis for heterogeneous collections of documents. (2010)	document collections, document management, change impact analysis, semantics, graph rewriting	changing, documents, other, different, collections

Table 8.1: The author-assigned keywords of all submissions of ACM DocEng are compared with the automatically extracted keywords. Here, I show six results.

The first example is based on an accumulation of all published ACM DocEng abstracts into one document. The total amount of submissions comprises 387 abstracts between the years 2001 to 2010. These are excellent candidates for keyword extraction because each single abstract corresponds to a window, so that each window contains a subtopic of the document. The difference in window size was handled by a normalization approach presented in Section 2.4.5. Then, I applied the combination proposed in Section 5.1.1 on the document and extracted the keywords.

Table 8.1 shows the computed keywords in comparison to the author-assigned keywords for six representative abstracts. In my opinion, the extracted keywords match the corresponding paper well. However, some results are interesting. Stop word filters would have removed the extracted keyword "other" in row six, though in this context, it refers to document relations, which is an essential idea of the referred paper. There is no doubt that author-assigned keywords should outperform automatically extracted keywords, but the keywords in row one and three show that the results of this particular algorithm come close. In fact, the extracted keywords in row two reveal, that the paper focuses on "Chinese" language and the submission in row five uses "technical" documentation.

The State of the Union Address of US President Barack Obama of January 25th 2011 is a single document that covers several topics, that were current at that time. I have split up the text into 26 consecutive windows of equal size.

Table 8.2 shows the top keywords of some sample windows. They clearly illustrate the variety of topics within this single speech. Additionally, this example shows the importance of keyword extraction for small portions of a text to help the reader to depict the actual semantic context.

Window	**Top Keywords**
5	how, innovation, future, what, change
12	rebuilding, infrastructure, high-speed, internet
16	freeze, spending, decade, chamber
21	afghan, qaeda, troops, taliban, thanks

Table 8.2: Extracted keywords for different parts of President Obama's State of the Union Address show significant content changes within the speech.

These examples show quite well, that a successful combination of two keyword extraction approaches can lead to meaningful results. These results may even outperform author-defined keywords. The evaluation will always be a subjective task as long as language can be interpreted differently.

8.3.4 Performance

The speed of a keyword extraction algorithm is especially important when the extraction is performed in a real-world scenario including user interactions. The response time of an application is a key criterion for its success. In this section, I aim to measure the speed of the here combined algorithm and compare it to the well known TF-IDF (TF-IWF) weighting algorithm [19]. I have implemented both algorithms according to the specifications made in Section 8.1. The hardware setup for this experiment is also described in that section.

Both keyword extraction algorithms were performed on plain text files of different length. In this case, I have measured the complete process of parsing the text, pre-processing the words, segmentation of the document, and performing the weight calculation for each term. It is crucial to measure the whole process, because document parsing and initialization of the algorithms require extra time, depending on the document length. Mostly in literature, pre-processing steps are excluded. As I aim to give a hint on real-world scenarios, the pre-processing was part of this runtime evaluation.

Figure 8.2 shows the total execution time of both algorithms – the sample combination and TF-IWF. The runtime scales almost linearly for TF-IWF and my approach as well. A linear growth of the execution time is a crucial feature that enables applications to ensure responsiveness. My algorithm performed

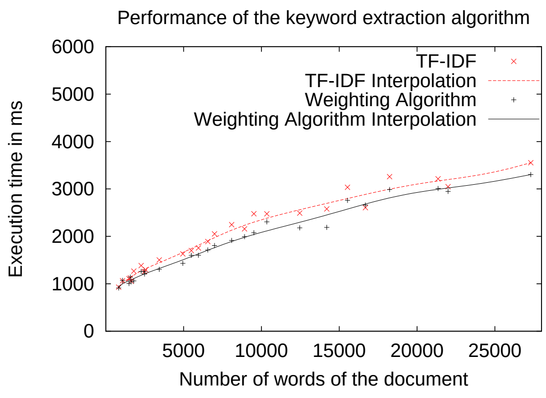


Figure 8.2: Performance of TF-IWF and my proposed weighting algorithm applied to documents of different lengths

slightly faster than TF-IWF. Here, I recall that in contrast to TF-IWF, the combined algorithm does not rely on stop word lists or other training data, whose time for creation is not listed here.

In conclusion, my approach appears to be fast for real-world applications, especially when run in a thread-based environment. The keywords can be extracted in parallel to the actual editing or viewing application. This experiment clearly shows, that a combination of two algorithms can – although performing similar to TF-IWF – clearly provide benefits such as the superfluous stop-word removal. The potential gain in meaningfulness provided by the additional keywords may lead to valuable insights for the reader.

8.3.5 The Phone Book

A use-case scenario for a keyword extraction algorithm would generally be a text of a size that does not allow for a quick information extraction for the human reader. A document that is usually not considered for keyword extraction is a typical phone book as it is known in the paperback form to most people. What happens, one applies the above mentioned keyword extraction algorithm to this kind of data? Which names would be considered as keywords – containing the key information?

For this experiment, I used a phone book from the Universität der Bundeswehr München. The original phone book contained more information than just names, titles, and the phone numbers of the people. I extracted the names and surnames of all the entries and preprocessed them according to the steps described in Section 2.2.2. The phone book contained a total number of 1795 entries with a total of 3617 terms. The phone book contains 1958 different terms. At a first glance, I analyzed the term distribution for this phone book. Figure 8.3 clearly shows that Zipf's law also applies to this scenario, but the curve is much flatter than the curve in Figure 4.1 in Section 4.3.2. This accounts for a low repetition rate of the terms.

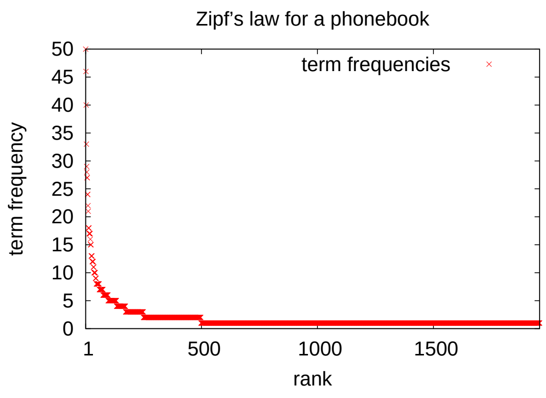


Figure 8.3: The term-frequency and rank distribution for the entries of a phone book.

To apply the keyword extraction approach described in Section 5.1.1, I separated the phone book into windows of almost equal size. The number if windows was set to ten as a window size of 10% turned out to be quite reasonable for this example. What is to be expected from the results of this keyword extraction algorithm? The results should account for the frequency of the terms, the term distribution across the windows (window frequency) and the burstiness.

The top five results of the extracted keywords appeared with a term frequency of $\{x_t^d(t_{a..e}) = (4, 3, 3, 3, 3) \mid t_{a..e} \in d, \phi(t_{a..e}) = (1..5)\}$, whereas the most frequent term in the phonebook "Andreas" appeared 50 times. The top keywords appeared somewhere in between the extreme values of Zipf's distribution, which complies with the assumptions of Luhn an Zipf (see Section 4.3.2).

In addition, these top keywords were similarly distributed within the phone book. They all appeared to be last names. This indicates that their position within the phone book is limited to a defined area. They all showed bursty behavior, which was detected by the keyword extraction algorithm. In summary, a keyword extraction from a phone book is not a common approach but this experiment confirmed the expected behavior of this particular combination quite well.

8.4 Evaluation of the Combination Approaches

Based on the properties of the individual weighting algorithms that I presented in Section 4.2, a combination of these algorithms seems beneficial [5; 76; 108; 81]. In Section 8.3, I compared the runtime of a combined algorithm to TF-IWF and showed that a combination may outperform a well known and widely used algorithm. In Section 8.4.1, I compare the different combination approaches presented in Chapter 5. Some of the results presented in this section have been pre-published in [16].

8.4.1 Comparison of the Data Fusion Methods

A common approach to measure the precision of a keyword extraction algorithm is to choose a telling real-world example and evaluate the extracted keywords by hand. The abstracts of a scientific conference are ideal candidates for such kind of an evaluation because each abstract clearly comprises a single topic and keywords can be identified by a human reader. They all are of similar length, which reduces the necessity and the effect of normalization approaches as normalization is always an assumption. For the comparison of the combination approaches, the first 14 ECBS 2012 conference abstracts were chosen as samples.

Each of the 14 ECBS conference paper abstracts was preprocessed and constituted a single window, with all 14 windows being treated like a single document – a summary version of the conference proceedings. Besides that, the abstracts provided additional information that facilitated the rating of the extracted keywords such as: author assigned keywords, headline, classification category.

The goal of this experiment was the comparison of five different data-fusion methods:

1. The Divergence from Randomness Framework (Amati)
2. The Minimum Ranking Method
3. Borda Count
4. The Schulze Method
5. Principal Component Analysis

I combined paired heuristics with different characteristics with all these combination methods.

1. $\mathcal{W}_\Gamma$ and $\mathcal{W}_{TF-IWF}$,
2. $\mathcal{W}_{Laplace}$ and $\mathcal{W}_{Binom}$.

The Divergence from Randomness model (see Section 5.1) allows the combination of only two algorithms, thus I combined exactly two with each of the combination approaches. The two paired algorithms were selected with respect to their properties as summarized in Section 4.5. Their properties indicate that a combination of these heuristics should lead to a successful keyword extraction algorithm. However, the focus of this experiment was not on the combined algorithms but rather on the differences the combination methods.

Figure 8.4 shows the precision of the top ten extracted keywords from all data fusion methods with the two different combinations. The graph in Figure 8.4 shows a high precision, which implies that the extracted keywords of the two combinations were meaningful and fit the topics of the samples. However, the results of the combination methods show certain differences.

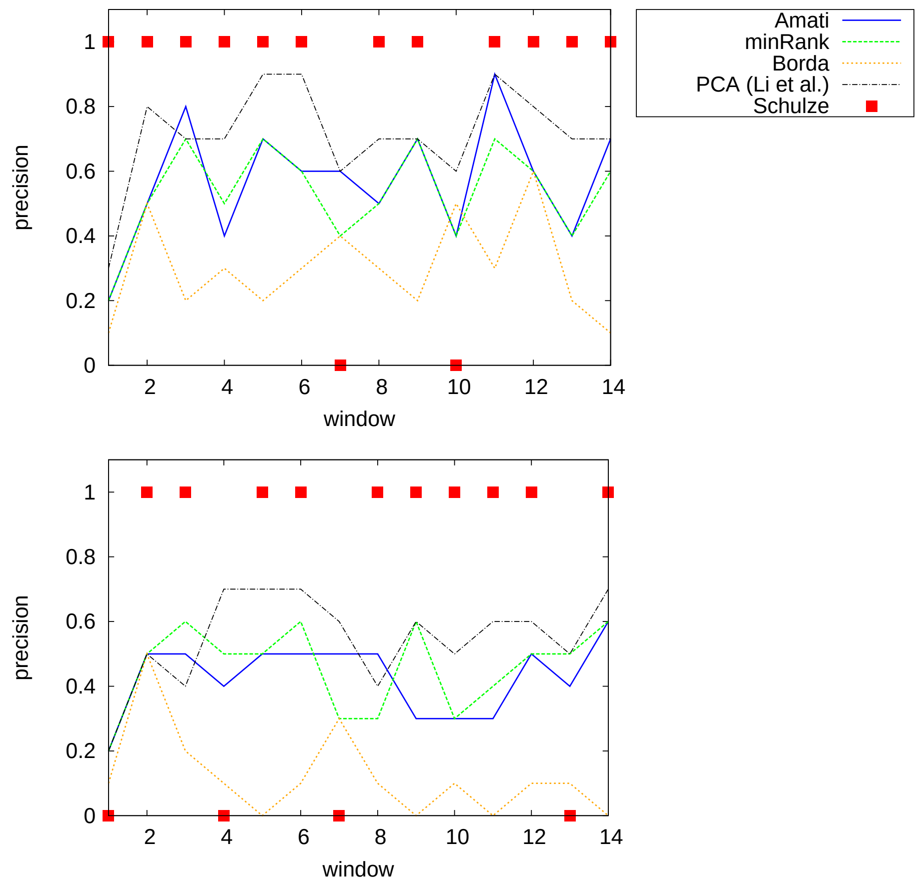


Figure 8.4: The graph shows the precision of the extracted keywords from 14 ECBS 2012 paper abstracts. The data fusion methods have been applied to the results of $\mathcal{W}_{\Gamma}$ / $\mathcal{W}_{TF-IWF}$ and $\mathcal{W}_{Laplace}$ / $\mathcal{W}_{Binom}$.

The PCA-based combination shows the highest precision overall. The results of the Divergence from Randomness approach and minRank rank lower in precision. But the extracted keywords of these methods appear to be subjectively more compelling, although the quality of keywords is not presented with this metric. The implemented Schulze method only determines one winner candidate for both heuristics and the potential keyword is either categorized as a keyword (100% precision) or not (0% precision). Since no extensive preprocessing nor stop-word filtration had been performed, I expected a relatively low precision for the extracted keywords.

One of the questions that arise is: How can the appropriate algorithms be determined and why do some of the algorithms perform better than others? The next section provides the answer through the results of an experiment.

8.5 Selection for a Successful Combination

To determine an optimal combination, let us consider the selection process of algorithms. As the weighting algorithms behave differently depending on the length, the structure and the writing style of the author, I decided to analyze their retrieval results. In this section, I analyze the result space of the presented heuristics (see Chapter 4) and present a simple method to determine optimal candidates for a successful combination procedure. The results of the first experiment presented, have been pre-published in a research paper prior to this work [16].

As a first step, I applied PCA to the result space of six different combinations of retrieval heuristics to determine potential candidates for a successful combination. Here, the objective of using PCA was not to reduce a highly dimensional dataset. Quite the contrary: a balanced set of components accounts for a low correlation rate of the data and therefore for a diverse set of results. I assume, a highly diverse set of results embraces more individual characteristics of the underlying document and is therefore desirable. Since PCA exclusively considers the result space, it accounts for all effects that have influenced the result generation and does not require any adjustment of additional parameters.

Based on the basic properties of the heuristics presented in Chapter 4, I have selected the following combinations for evaluation:

1. $\mathcal{W}_{\Gamma}$ and $\mathcal{W}_{TF-IWF}$
2. $\mathcal{W}_{\Gamma}$ and $\mathcal{W}_{Binom}$
3. $\mathcal{W}_{TF-IWF}$ and $\mathcal{W}_{Binom}$
4. $\mathcal{W}_{Laplace}$ and $\mathcal{W}_{\Gamma}$
5. $\mathcal{W}_{Laplace}$ and $\mathcal{W}_{TF-IWF}$
6. $\mathcal{W}_{Laplace}$ and t$\mathcal{W}_{Binom}$

I only combined two algorithms at a time in order to emphasize the effect of their individual properties on the result space. A successful selection of retrieval algorithms should comprise the defined constraints with respect to the source text (Chapter 5) and may require a set of three or more retrieval algorithms.

To determine the variance of the result set for each of the six algorithm pairs, I applied them to three different texts of different genres, lengths, and structures: the top 14 ECBS paper abstracts of 2012, US-President Obama's State of the Union Address in January 2012, and the English book Treasure Island by Robert Louis Stevenson.

Scientific research paper abstracts are very suitable for keyword extraction because of their similar length. The scientific texts contain a lot of potential keywords, and the number of windows is relatively low. I treated each abstract of the top 14 ECBS papers from 2012 as a window and the collection of windows as a single document. In contrast, the presidential speech represents a larger document of a different writing style. It is divided into 104 consecutive paragraphs of different length, representing the individual windows with subtopics. My largest

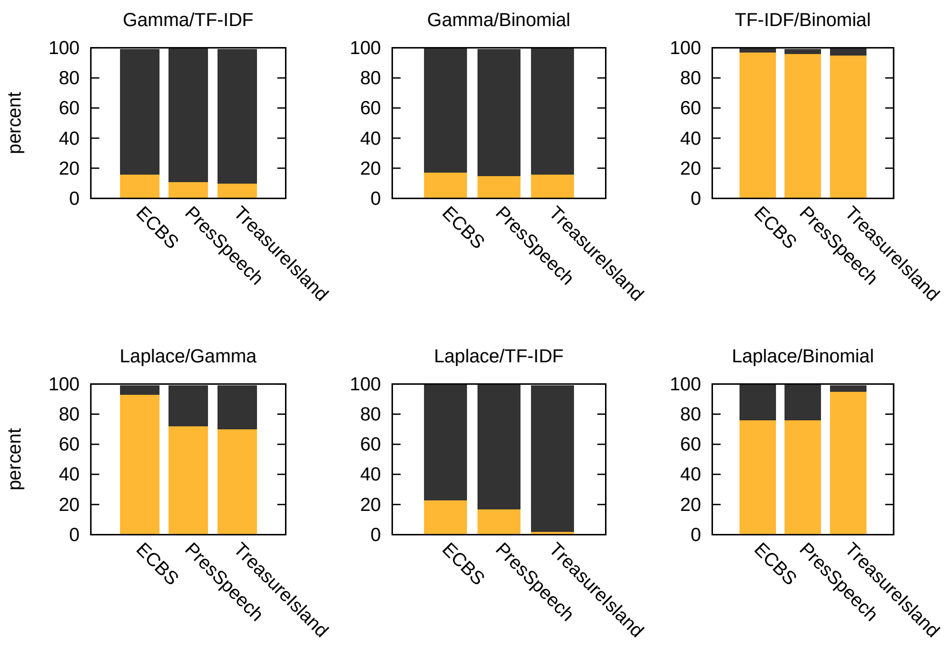


Figure 8.5: The PCA eigenvalue proportions for six different heuristic retrieval combinations show the individual contribution of variance to the result set.

sample – the book Treasure Island – is the biggest document of the three test cases and consists of 5758 paragraphs of different size. Each paragraph was treated as an individual window.

The algorithms were performed sequentially for each of the combinations 1) to 6). Subsequently, I analyzed the results with PCA. The fraction of the individual eigenvalues of the sum of the eigenvalues of all components as defined in Equation 5.3.1 is depicted in Figure 8.5. The single bar charts depict the contribution of the results of each individual component to the variance of the results of the combination for the three sample texts. A strong contribution of both components indicates two individual, uncorrelated variables; this seems beneficial for this analysis.

The results for combination 1) and 2) show that the combination of a frequency-based heuristic with a term-distance measure appears to be beneficial compared to the combination of $\mathcal{W}_{TF-IWF}$ and $\mathcal{W}_{Binom}$. It is clearly visible, that the two frequency-based measures $\mathcal{W}_{TF-IWF}$ and the $\mathcal{W}_{Binom}$ do not contribute to richer results when combined with each other. Another characteristic that can be observed is the influence of the length of the documents. The combination of $\mathcal{W}_{TF-IWF}$ and $\mathcal{W}_{Laplace}$ performs well for the short texts but the contribution of the second component is almost non-existent for Treasure island. This is due to the fact that the contribution of $\mathcal{W}_{Laplace}$ to the variability of the result set decreases with larger document size. Apparently, the performance of this algorithm depends on the size of the texts.

A similar behavior can be observed in the chart of the combination for $\mathcal{W}_{Binom}$ and $\mathcal{W}_{Laplace}$. This behavior is not surprising since the Laplace Law of Succession is solely based on term frequency. In fact, this indicates that $\mathcal{W}_{\Gamma}$ is a better candidate for modeling the burstiness for larger documents.

As the results of the PCA-analysis seem to account for the properties of the retrieval heuristics as well as for the characteristics of the test documents, I propose to apply PCA to the set of results of the individual heuristics before the final combination procedure to select the most beneficial retrieval heuristics for a combination.

As an example, I ran all algorithms on a single document: the English book Treasure Island by Robert Louis Stevenson and performed PCA as it is described on Section 5.3 subsequently. The results of this analysis are shown in Figure 8.6.

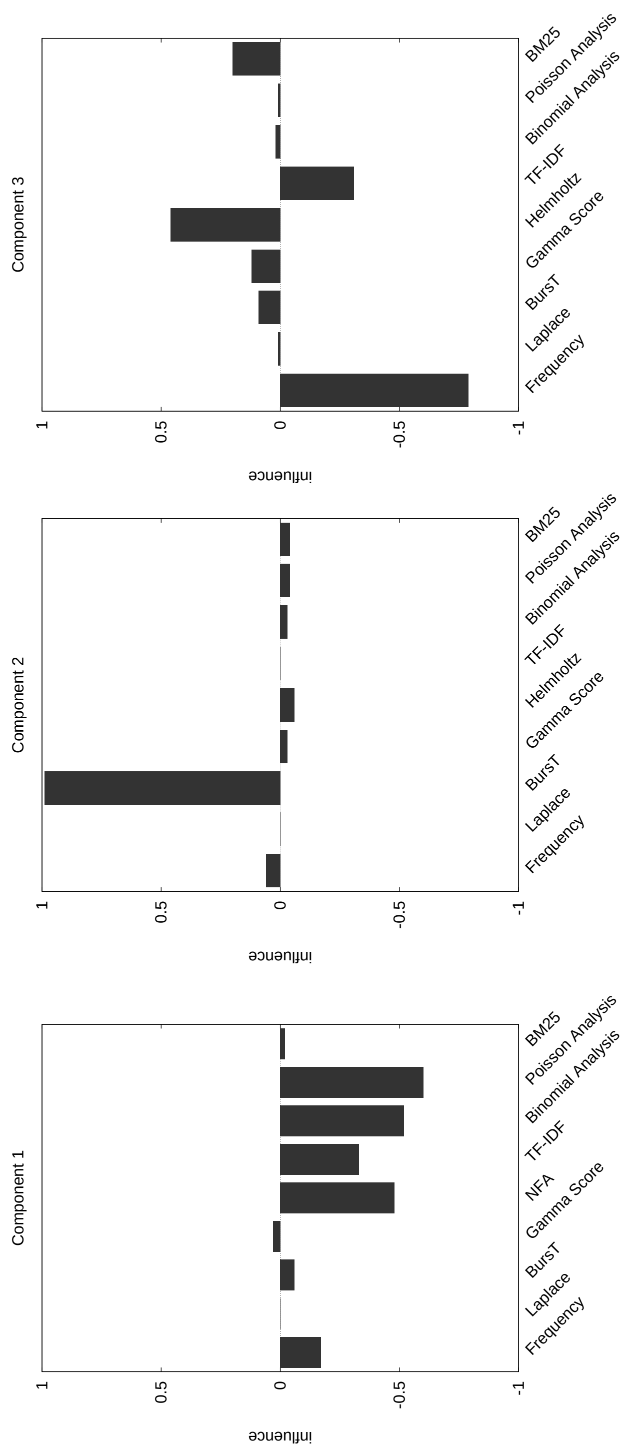


Figure 8.6: The PCA eigenvalue proportions for six different heuristic retrieval combinations show the individual contribution of variance to the result set.

The results of this analysis were surprising as the individual components clearly indicated the properties of the algorithms as stated in Section 4.2. The first two components comprise more than 98% of the result set. One may interpret that these two components contain more than 98% of the information of the result set. The direction of the bars in the graph indicates the results of the algorithms are positively correlated. If one bar is positive and another negative, the results are negatively correlated accordingly. The height of the bars indicates the level of correlation.

The first component indicates a very strong correlation of $\mathcal{W}_{TF-IWF}$, $\mathcal{W}_{NFA}$, $\mathcal{W}_{Binom}$, and $\mathcal{W}_{Poisson}$. All of these algorithms are based on term frequency, which I stated in Section 4.5. This can also be observed by the first bar. It represents the term frequency as a reference value. The results of $\mathcal{W}_{Binom}$ do not correlate well with the first property. In Table 4.1 is stated that $\mathcal{W}_{Binom}$ does not consider the window frequency, whereas $\mathcal{W}_{TF-IWF}$, $\mathcal{W}_{NFA}$, $\mathcal{W}_{Binom}$, and $\mathcal{W}_{Poisson}$ do consider window frequency to a certain degree. In summary, the principal component indicates that the results of $\mathcal{W}_{TF-IWF}$, $\mathcal{W}_{NFA}$, $\mathcal{W}_{Binom}$, and $\mathcal{W}_{Poisson}$ show a very similar behavior. This behavior represents a large portion of the information of the result set, and is based on their common properties *term frequency* and *window frequency*.

The second component indicates, that the results of $\mathcal{W}_{BS}$ represent a portion of the variance of the result set, which does not correlate with any other algorithm. I assume that this is based on its very specific modeling of term burstiness, that differs quite significantly from the other algorithms, that model burstiness. The *BursT* algorithm clearly represents features of this particular text that none of the other algorithms may capture.

The third component only represents less than 2% of the variance of the result set, but its importance is not decreased in any way for this evaluation. The bars clearly indicate a very strong correlation with the reference value term frequency. Interestingly, one may observe that the bars of the results of the TF-IWF-algorithm and the results of the Helmholtz algorithm indicate a quite strong decorellation. This is due to the aforementioned fact: TF-IWF contains a lot of stop words and Helmholtz does not. Stop words usually appear frequently and therefore correlate strongly with term frequency as shown in Figure 8.6.

The results shown in Figure 8.6 only account for the chosen sample. With PCA, I clearly identified certain properties that were represented by the algorithms used. These properties account for information contained in the text, but they may differ for other texts as the algorithms may behave differently for longer texts or texts of a different style.

The consequence of such an analysis leads to a proper selection of algorithms, that are able to consider most of the information within the text and combine them. For this example, the algorithms of choice would be Helmholtz and BursT or a combination of TF-IWF and BursT if a stop-word removal is performed before the analysis. The optimal selection of algorithms may vary for each sample as the properties of written texts can be very diverse.

In this section, I presented a flexible method for clear identification of properties of the different algorithms for a specific sample. These properties also correspond to the theoretical analysis, presented in Chapter 4.

8.6 Trie-Based Change-Point Detection Algorithm

The comparison of different measures of entropy in Section 8.6.1 undermines the claim of the existence of long range correlations in texts. The presented CPD approach analyzes these correlations and detects hidden passages in texts. Section 8.6.3 contains a performance analysis of this algorithm as well as a comparison with another state-of-the-art CPD algorithm. The results of the comparison are remarkable.

8.6.1 Entropy Measures Compared

A range of different entropy measures for written text exist, but only a few of them account for long range correlations. Here, I compare a basic estimation of the Shannon Entropy with the entropy estimate of the parameter free LZ77 algorithm and my trie-based version of the LZ77 algorithm. I applied the following entropy estimation techniques:

The Shannon Entropy for a sample was estimated with the maximum likelihood estimate, that was described in Section 6.2. The estimation based on characters differs from the one based on terms. The estimates of the LZ77 algorithm were solely character-based and were calculated with the formula provided in [71]. Kontoyiannis et al. defined an entropy estimator for a sliding window LZ algorithm as follows:

$$\hat{H}_{k,n} = \left[\frac{1}{n} \sum_{i=1}^{n} \frac{\Lambda_i^n}{\log n} \right]^{-1}$$

The denotation Λ_i^n is the match length of the next phrase to be encoded by the sliding window LZ. The size of sliding window is denoted with n. In other words, Λ_i^n denotes the length of the shortest substring starting at position i that has not been seen as a substring in the previous n symbols.

I also applied the calculation of $\hat{H}_{k,n}$ to the trie-based version of the adapted LZ77 algorithm for characters and terms individually. As the matches of LZ77 and the trie-based algorithm differ, the results of the estimator $\hat{H}_{k,n}$ were likely to be different. In this experiment, the purpose of $\hat{H}_{k,n}$ was to serve as a measure of the compression rate and convergence of LZ77 and the trie-based algorithm.

In order to show the difference between these three estimates, I applied them to a number of different texts of different size and estimated the entropy of these texts. The texts were retrieved from *Project Gutenberg* [110] and [135]. I analyzed State of the Union addresses of former Presidents of the US as well as literary books from Jane Austen (as in [71]), Lew Tolstoi, Johann Wolfgang Goethe, as well as the Holy Bible. All texts were available in the English language. Throughout my experiments I discovered that scientific texts seem to have a lower entropy than literary texts such as novels or poems. This may be resulted in the fact, that redundancies are generally avoided and the vocabulary of scientific texts is usually more limited than in literary texts.

Figure 8.7 shows the entropy estimates of the three different algorithms. Shannon estimated the entropy of written English to be between 0.6 and 1.3 bits per character (bpc). The LZ77 algorithm was closest to this estimate as it estimated the entropy of all sample texts close to a value of 2 bpc and approached a

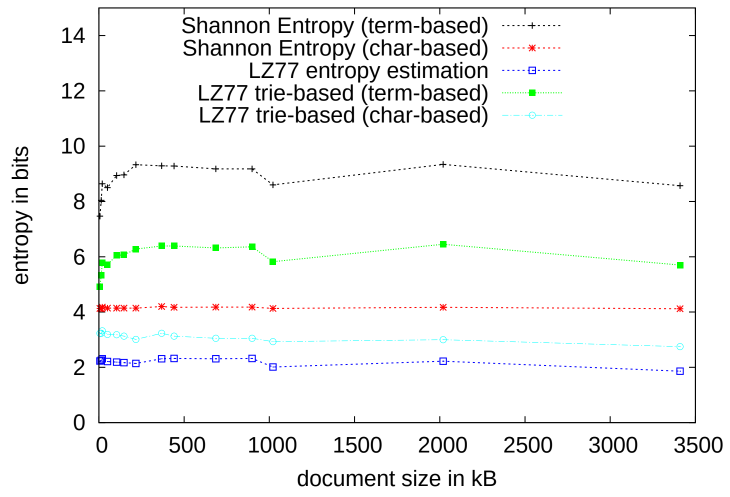


Figure 8.7: Comparison of Shannon Entropy, LZ77 estimator, and trie-based LZ77 estimator with texts of different size.

value even lower than two for large texts. As expected, the Shannon estimation method did not capture all correlations in the sample texts and estimated much higher values than the LZ methods. Figure 8.7 shows clearly, that my trie-based approach was able to capture far more information than any of the plug-in Shannon estimates. The difference between the LZ77 entropy estimation and trie-based estimation represents the amount of information lost by the adaptions made in this work. The character-based approaches were able to catch more correlations than the term-based methods. This is because some words share similar character combinations and no stemming was performed beforehand. Minor positive (at 2 MB) and negative deviations at 1 MB from the expected continuous results were observed. The reason for these deviations is likely to be justified in the fact that not all the texts were from the same author and of the exact same genre.

8.6.2 Trie-Based Change-Point Detection

This section provides the results of the trie-based CPD method, which is described in Section 7.4. To demonstrate its functionality, I applied this novel algorithm to several real-world examples while the following properties of the trie are measured as the algorithm processes the text:

- maximum depth
- total number of nodes
- total number of leaf nodes
- input position

First, I applied the trie-based algorithm to the English version of Goethe's Faust. During this evaluation, I observed a predominantly flat trie-structure. This observation was made for a large amount of texts. Consequently, the maximum depth provided only limited amount of information. However, the total number of nodes accounted for the whole trie and responded immediately to structural changes. The measured total number of nodes and the filtered input positions (described below) are show in Figure 8.8. The algorithm has been launched in the term-based version and the window size $|w|$ was set to 200.

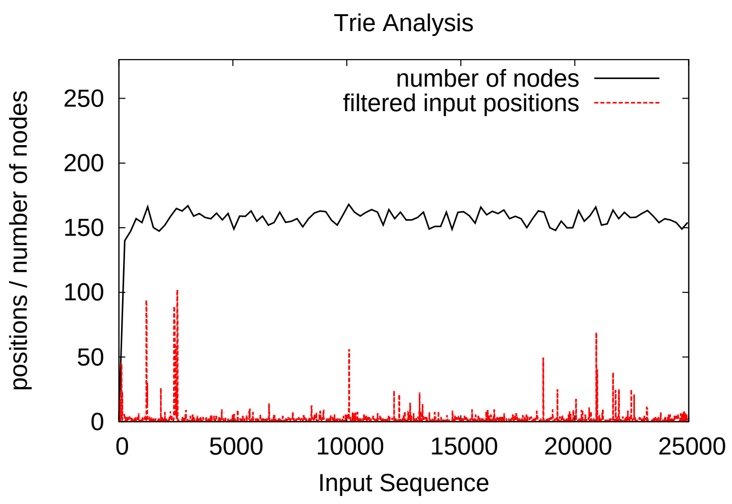


Figure 8.8: Trie-based analysis of Goethe's Faust.

The *number of nodes* measure indicated a sudden growth of the trie after the start of the process up to approximately 150 nodes. A rise of the *number of nodes* was always accompanied by input sequences not observed previously. As a result, the input positions increased likewise because new entries were inserted into the trie. The values shown in Figure 8.8 were smoothened with a median filter to perform a noise reduction and to preserve the edges of the curve. The noticeable peaks of the filtered input positions were surprising, since Faust was written by a single author and it was not immediately obvious that the book contains noticeable changes of the writing style. As I later observed, the English version actually contained German sections. These German sections triggered the peaks of the input position measure and a simultaneous increase of the total number of nodes. Apparently, this trie-based algorithm was able to detect these "hidden" passages well.

Although the difference between terms and characters has a significant impact on the entropy estimates as shown in Section 8.6.1, the results of the CPD algorithms did not show considerable difference between the results. This may be due to the fact, that the statistical effect of a change-point in my examples was strong enough to be captured by terms. In addition, terms are at a higher abstraction level than characters and account for less ambiguity between different languages. Consequently, I based all the examples presented in this section on an analysis of terms rather than characters.

To show that this algorithm can detect a change-point within a text, I applied it to a concatenated excerpt of the English and German versions of Goethe's Faust. This example has been chosen in [62] in order to detect a change-point between these two texts. In a first step, I generated a Java-version of the match-length-based approach described in [62] and applied it to the sample text. The documents were preprocessed according to the specifications made in Section 2.2.2. The result of this experiment serves as a reference and is shown in Figure 8.9.

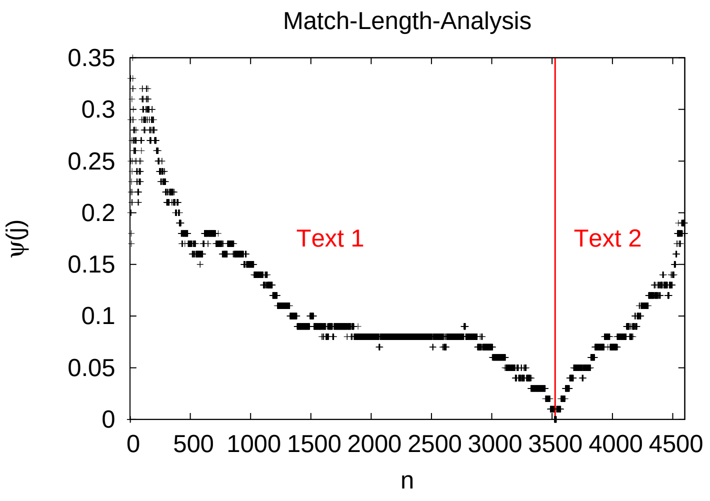


Figure 8.9: Analysis of a concatenation of two Faust excerpts with the match-length-based algorithm.

The minimum of the crossing function $\psi(j)$ (see Section 7.1) marks the estimated change-point between the two texts. I expected a similar result from my trie-based CPD algorithm and applied it to the same text ($|w| = 200$, term-based). The result of the trie-based approach is shown in Figure 8.10. The number of nodes increased at the beginning of the analysis as the trie was initialized. The significant increase of filtered input positions clearly indicates a change-point at about 2500 steps, which matches the position in the text that 8.9 estimated as the most probable change-point. The difference in steps between the identified change-points of two algorithms is due to the following:

In Figure 8.10, each step accounts for a new match sequence added to the trie, contrary to the term position in Figure 8.9. I used a minimum filter to detect a lower bound of input positions because a change-point was most probable at a point where a series of almost exclusively new term sequences was inserted into the trie. Since this is not a sufficient criterion for a change-point, I experimented with different *thresholds* of the input positions to set a criterion for change-points. The minimum filter of input positions in combination with a threshold between 25% to 50 % of $|w|$ appeared to be reasonable to automatically detect a change-points for the examples presented. The thresholds depend on the text itself and are not domain-dependent.

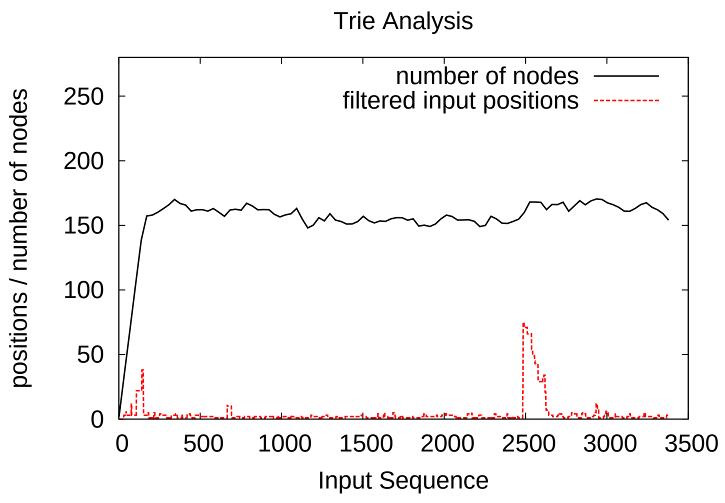
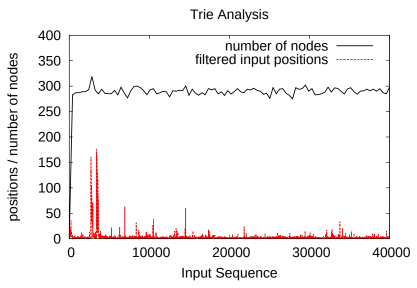
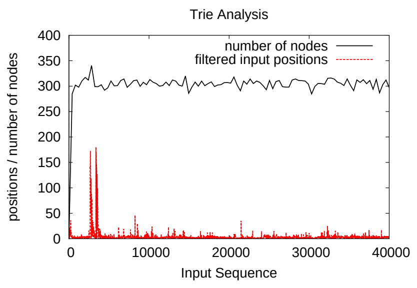


Figure 8.10: Analysis of a concatenation of two Faust excerpts with the trie-based algorithm.

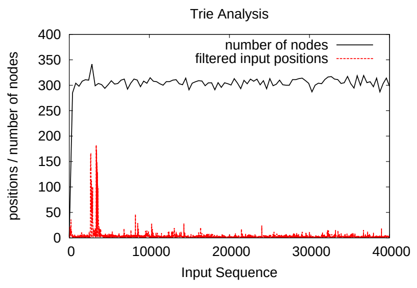
In the next example, I applied the trie-based algorithm to a text with one injected passage. Due to the length of the book Treasure Island by Robert Louis Stevenson, I considered it a text stream. This text stream contained the German Wikipedia article about Treasure Island, inserted at a random position. The same preprocessing steps as above were applied and the initial value for $|w|$ was set to 400. The trie-based algorithm was applied with the four different versions of the *decay* function, presented in Section 7.5. Figure 8.11 shows the results for this example.



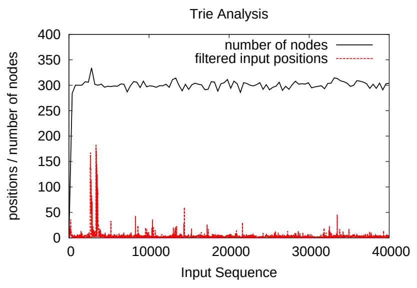
(a) $decay_1$



(b) $decay_2$



(c) $decay_3$



(d) $decay_4$

Figure 8.11: Trie analysis to locate the boundaries of the German Wikipedia article that was randomly inserted into the text of the book Treasure Island.

The boundaries of the Wikipedia article can be clearly identified in all four pictures by the two peaks of input positions within the first quarter of the book. The first of the two peaks is accompanied by a rise of number of nodes in the trie due to the new terms in the Wikipedia article, that have not been observed before by the trie. The hidden passage was identified best with function $decay_1$. The change-points were located exactly at sequence numbers 2684 and 3409 with the threshold-based procedure described above. However, the fluctuations in the remainder of the text differed. Through analyzing the results of $decay_2, decay_3, decay_4$, I realized that the position of the peaks (input positions and number of nodes) was slightly off the expected location. Investigations revealed, that these decay functions erased nodes with terms that appeared more frequently earlier in time, instead of increasing their life time. Only an increased life time for these nodes leads to more matches with similar sequences, thus preserving the long term correlations. Eliminating these nodes faster, leads to unexpected results. The improved decay functions are displayed in the following equation:

$$decay_1(\vec{\Delta\tau}) = |w|, \qquad |w| \in \mathbb{N} \tag{8.2}$$

$$decay_2(\vec{\Delta\tau}) = |w| + \Delta\tau_n, \qquad \Delta\tau = \Delta\tau_1, ..., \Delta\tau_n \tag{8.3}$$

$$decay_3(\vec{\Delta\tau}) = |w| + median(\vec{\Delta\tau}) \tag{8.4}$$

$$decay_4(\vec{\Delta\tau}) = |w| + min(\vec{\Delta\tau}) \tag{8.5}$$

With the improved decay functions, the trie-based algorithm detected the change-points correctly and adapted the size of the trie automatically to a certain degree. Nevertheless, the "resolution" of the detection procedure has to be set beforehand with $|w|$.

During my experiments, I used a dictionary as control data for my change-point analysis method. The expected result was a flat tree structure with only child nodes of the root node, containing the terms of the dictionary. The English dictionary had 349.900 terms in total. I performed an analysis with my trie-based algorithm, the window size set to 200 terms. The result is shown in Figure 8.12.

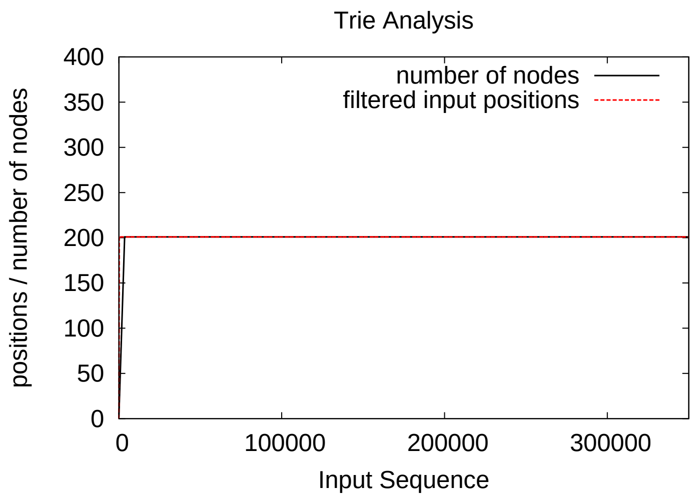


Figure 8.12: This figure shows the results of the trie-based CPD algorithm applied to a dictionary.

While the algorithm read the first 200 terms, the trie built up until it had reached the maximum window size. Afterwards, the input position and the number of position remained constant until the algorithm had read the 349.900 terms.

A few approaches aim to detect plagiarism [133; 23] with IR methods by using stop words or n-grams. These methods aim to detect authorships plagiarism in texts. The purpose of the following experiment was to test, if my trie-based algorithm would detect a change-point between two texts written by two different authors. Therefore, I placed the first State of the Union address of the first President of the United States of America (USA) George Washington from January 8, 1790 before the most recent State of the Union address of the present President of the USA Barack Obama from January 8, 2014. The two speeches were written by different authors in different times, and they are of different length: Washington's speech contains 1089 terms and Obamas speech contains 6843 terms. I treated the two speeches as one text and analyzed them with the trie-based change-point algorithm. The window size was 200 terms and $decay_1$ had been used. The results of this sample can be seen in Figure 8.13.

Figure 8.13 contains three measures during the analysis of a total of 7932 terms and 5214 input sequences. The first speech ends at input sequence 675, which is marked with a vertical red line in the graph. The *minimum filtered input*

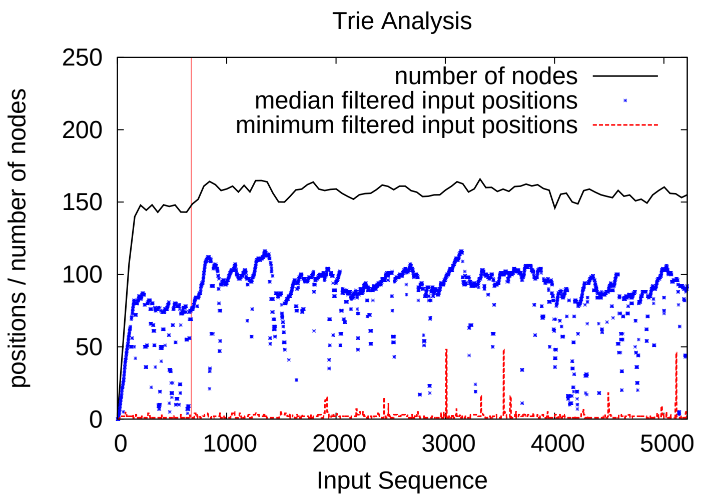


Figure 8.13: This figure shows the results of the trie-based CPD algorithm applied to two consecutive presidential speeches.

positions measure indicates three change-points at positions 3008, 3533, and 5112. These positions are clearly located within Obamas speech and don't indicate a change-point between the two speeches. Instead, they appear to be at paragraphs with new subtopics in Obamas speech:

- trie input sequence 3008: *equal pay for women*
- trie input sequence 3533: *health care system*
- trie input sequence 5112: *veterans, freedom, democracy*

The difference in word use between the two State of the Union Addresses appeared to be smaller than the differences between the subtopics in Obamas speech. In this case, the *minimum filtered input positions* measure did indicate different authorship. Instead, the *number of trie nodes* grew instantly as the algorithms processed the first terms of Obamas speech.

In Figure 8.13 I also depicted the *median filtered input positions*, which show this increase more clearly. Apparently, the trie receives a significant number of new first level nodes as the algorithm reads the first terms of Obamas State of the Union Address. Interestingly, this has not been identified as a change-point simply because the minimum filter was not suitable for this sample. Since both texts are the same language, there are enough common words (stop words) that appear in both texts with a high frequency. These terms have low input positions: the rise of the *median number of input positions* near sequence 675 was accompanied by low values, whereas a rise of *median number of input positions* near sequence 3000 was not. Consequently, the *median number of input positions* may be an additional indicator for authorship plagiarism detection.

To conclude, the presented algorithm detected fluctuations of information content, but it does not consider the meaning of the information. However, this structural information can be used for further analysis, such as the segmentation of the text into individual windows containing a single subtopic. A topic-based segmentation is desirable for a number keyword extraction approaches (see Section 2.3).

8.6.3 Performance Analysis

In this section, I describe the execution time analysis of the trie-based algorithm. The execution time of an algorithm is a very important usability aspect, especially if the algorithm is used for online streaming text data, or for an information extraction system with user interactions. My reference application has been implemented according to the specifications described in Section 7.6. The analysis has been performed with the setup specified in Section 8.1.

In order to compare the execution time, I implemented the state-of-the-art algorithm presented in [62] as a benchmark. I denoted this benchmark algorithm "match-lengths", since the core of this algorithm is based on the match-lengths described in Section 6.3. The new trie-based algorithm has been applied in its most advanced (adaptable) version with a window size of $|w| = 200$. Both algorithms were executed on texts of different size ranging from 6 Kilobyte (kB) up to 247 kB and the results of the measured execution time are shown in Figure 8.14. The execution time has been plotted in seconds on the y-axis.

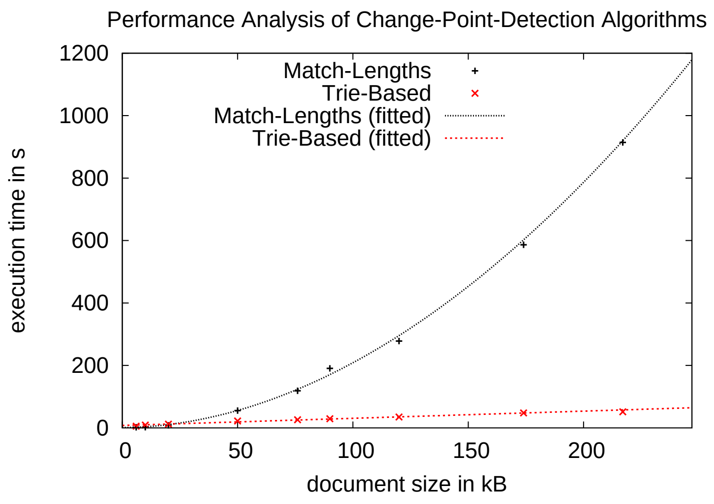


Figure 8.14: Performance analysis of the trie-based algorithm.

Whereas the curve of the trie-based algorithm scales linearly with respect to the document size and runs very close to the x-axis, the benchmark instead performed with quadratic complexity. For the smallest tested documents the match-length-based algorithm performed even better than the trie based algorithm, which I attribute to the trie build up overhead at the beginning of the scan.

As soon as the trie has reached it's defined size, the trie transformation operations continue to be the major operations. The fitted line of the trie-based execution time grew with a gradient of 0.218 and with an offset of approximately 8.10. The maximum asymptotic standard error for the fitted line is 5.913%. The fitted curve for the CPD algorithm exhibits almost quadratic growth with an exponent of 1.93 and an asymptotic standard error of 3.012%.

These curves revealed one of the key features of my algorithm: its complexity and therefore its performance allow the application to indefinite streams and the detection of changes in almost real-time. The implementations used have room for improvement since they are only reference implementations, but they revealed the performance characteristics of the algorithms adequately.

Chapter 9

How Keywords link Social Media

In this chapter, I describe how automatic keyword extraction based on heuristic text analysis has been applied to the CommunityMashup – a use case scenario for the heuristic analysis. The CommunityMashup is a data integration solution for aggregation of data from different social services developed at the Universität der Bundeswehr München by Peter Lachenmaier and his colleagues [74]. This software aims to provide an environment for more than just connecting persons and their online profiles to their real individuals. It aims to identify connections between contents of different social media services with automated linking techniques. Automatic keyword extraction and text analysis were successfully employed to calculate connections between objects in aggregated datasets. The quick, language-independent retrieval approach fits exactly the needs of the constantly changing data of the CommunityMashup. First, I describe the CommunityMashup as a data integration solution. Subsequently, I describe the link building process in detail.

9.1 CommunityMashup

One of the key success factors of Web 2.0 platforms is the transparency of an individual's activities and interests to friends, followers and specific groups. Subsequently, a multitude of technically enabling platforms exist and each of them provides individual benefits to its users. Real world users create a number of independent profiles on different social media platforms to share information. Support for interlinking platforms and especially for interlinking content is missing, which results in redundant, unlinked cross-posts.

The microblogging service Twitter has a built in feature that does exactly this [151]. When connected to other social media services, it automatically tweets all posts of the connected social media platforms, although the have been posted already. Various data integration solutions attempt to fill this gap by aggregating data from different services using techniques such as visual or dynamic cross-media linking. A major problem with aggregated datasets from different services is, that they generally have poor interlinking. In this approach users are allowed to modify data: either by interacting with a specific application or by directly using the source system, which results in frequent changes of the dataset. Thus, connections must be recalculated very quickly to keep them up-to-date. This approach has been implemented within an existing person-centric data integration solution to ensure that each (natural) individual is represented only once in the overall dataset, independent of how many virtual identities for separate services each has. Furthermore, user-generated existing connections can be filtered.

Entity recognition is a subtask of information extraction and it is essential for the identification of real persons, locations, or organizations [41]. Here, entities are defined as persons – representing real individuals – users of social media platforms, or authors of platform content. The entities are identified by their email or full name, and they can own multiple accounts, or the can be authors of multiple contents.

This technical solution has been built inside the environment of the CommunityMashup [74]. In the following, I present an overview of the CommunityMashup to clarify the integration of the keyword-based link generation into the whole system. The CommunityMashup is a flexible integration solution for data from social services and provides features such as application frameworks with offline data access for different platforms. In contrast to existing mashup solutions, it aims to provide unified and aggregated information based on a person-centric data model. This enables the integration of social media data that naturally belongs to a person or an organization, but is artificially distributed over different services on the web. Figure 9.1 depicts the layered overall structure of the

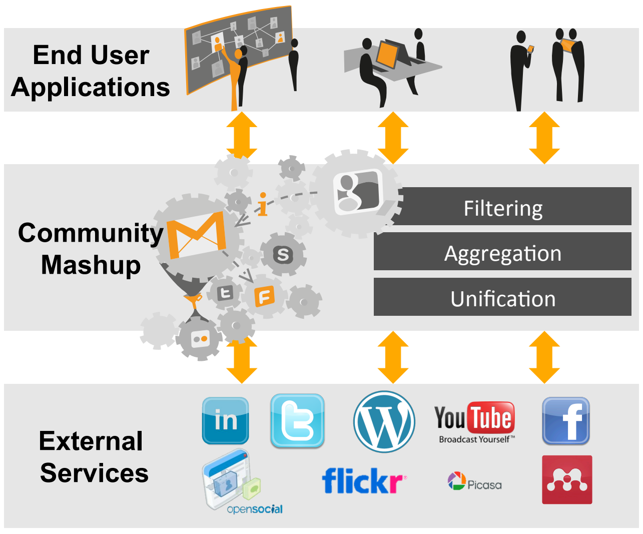


Figure 9.1: CommunityMashup Overview

CommunityMashup containing a few exemplary external services at the bottom and abstract mashup components in the middle, which are responsible for data unification, aggregation, and filtering. The top layer shows mobile-, web-, and desktop-clients as three different exemplary consumer applications representing stereotypical usage scenarios.

The aggregation of data from different sources allows the creation of connections across system boundaries. Furthermore, it allows the combination of automatic- and user-generated connections. The person-centric aggregation of the CommunityMashup also enables the connection of persons with their respective contents and interlinking of persons that, for instance, work on similar contents.

This setup allows links to be created in background processes on the server side, and to easily provide the results for consuming (desktop or mobile) applications.

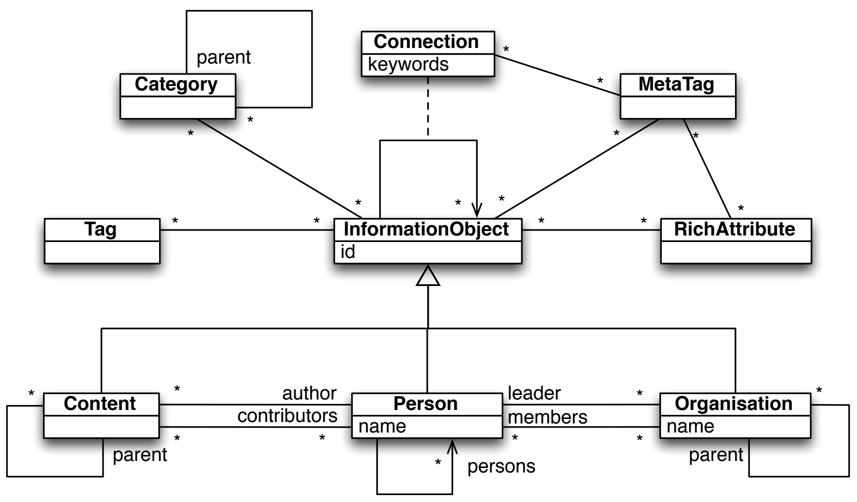


Figure 9.2: CommunityMashup data model containing core objects and connection class

Figure 9.2 shows the core elements: *person*, *content*, and *organization* of the mentioned internal person-centric data model. The data model also shows the connection class, which can connect two of these information objects. This core model is limited only to the most important entities of Social Software. The central element is the person, which can be grouped in organizations, and can author or contribute to content. Organizations and content can be structured hierarchically (parent relation). Additional information can then be assigned to the information objects by rich attributes, tags, meta-tags and categories. Categories, in comparison to tags, can be modeled hierarchically. Tags, categories, and rich attributes carry information that is directly gathered from external services whereas meta-tags are specific to concrete scenarios.

9.2 Link Building

The link building process requires a content analysis beforehand, where similarities between contents can be identified. In this case, the following two keyword extraction approaches were performed:

- $\mathcal{W}_{TF-IWF}$
- $\mathcal{W}_{BS}$

These two algorithms were combined with the PCA-based combination method (see Section 5.3). These two algorithms were chosen because of their properties and their robustness. Additionally, I performed stop-word extraction, because some of the sample texts contained a lot of terms that do not serve well as keywords, such as: “tweet”, “retweet“, and URLs.

They had a more functional relevance such as: URLs, and application specific commands. Due to the different size of the sample texts I also normalized the texts according to the specifications presented in Section 2.4.5. I experimented with a number of approaches and this configuration served well for this use-case scenario.

This keyword extraction approach was applied to three different preprocessed datasets individually. From the content analysis, different types of links were computed:

9.2.1 Types of links

Depending on the concrete application scenario, a balance between accuracy and interlinking needs to be specified. Two parameters were defined in order to configure the accuracy and the number of links being created. For all three types previously existing connections were filtered. Existing connections could be two people or two authors of the same content. The newly created connections can be categorized as follows:

Content-Content (CC)

I extracted a list of the best keywords for all contents and compared them to the keywords of all other contents. A connection between two contents is then created if there is more than the configured number of required common keywords.

Person-Content (PC)

As mentioned above, the CommunityMashup creates links between contents, their author, and additional contributors automatically. If the keywords of a contribution of a person match the keywords of another content, a new connection between this content and the person is being created. Consequently, persons are linked with potentially relevant contents.

Person-Person (PP)

Similar to Person-Content (PC) connections, connections between two persons are created if they author similar contents.

Abstracts of scientific papers

Scientific paper abstracts are suitable for heuristic keyword analysis because they precisely comprise a single research project and thus provide inherent topic segmentation. Furthermore, they are all of similar length, which leads to minimal distortions by normalization. Two different datasets of 50 scientific paper abstracts were used for our evaluation:

- the abstracts of the Communities and Technologies (C&T) conference of the years 2009 and 2011
- the abstracts of the 50 most recent German and English papers of the Cooperation Systems Center Munich (CSCM)

The precision of the extracted keywords for the scientific paper abstracts was generally very high, but the precision was not explicitly measured in this paper. Instead, a telling example is provided to further illustrate our approach:

From the C&T abstracts 242 unidirectional links between contents (Content-Content (CC)) with a minimum of three matching keywords were generated. This resulted in 121 newly created bidirectional CC links. One of those links is based on the three matching keywords:

```
[wiki, knowledge, communities]
```

The keywords have been extracted from the two following papers:

- Wiki-based community collaboration in organizations [87]
- Mail2Wiki: low-cost sharing and early curation from email to wikis [52]

The extracted keywords indicate a conceptual association between the two papers and therefore new Person-Person (PP) and PC connections have to be created. Three persons have been contributing to the first paper and seven people to the second. Consequently, 21 PP connections $(3 * 7)$ and 10 PC connections $(3 + 7)$ are created. The analysis of the two source documents results in a total of 32 connections, based on the three keywords.

Tweets

The third test set consisted of 400 tweets. The challenges of analyzing tweets are: their short length, the high repetition rate (retweets), and their general lack of content within the single tweets. It should be mentioned, that in almost all tweets, each term appears at most once due to the short length of the individual texts. For this evaluation, the most recent 400 tweets, hash-tagged with the terms “Europe” or “Greece”, were analyzed. Furthermore, the few preprocessing steps (see Section 2.2.2) were applied and the terms “europe” and “greece” were deleted from the source texts.

Connections based on multiple matching keywords often indicated retweets. They haven’t been excluded from the dataset because they served as a basis for PP connections. Connections based on a single keyword can reveal interesting links between independent events: just as the keyword “crisis” links a report about the ongoing crisis of the European Union and the recent hostage crisis in Algeria.

In addition, single keywords can link independent sources that refer to the same event: The keyword “eyesight” links a New York Times newspaper article with an Indian article. Both refer to the same acid attack in Russia. A very interesting effect can be observed by analyzing tweets that span multiple languages. Due to the fact that important terms reappear in tweets of different languages and are identified as keywords by the keyword extraction algorithm, it is possible to create links between tweets of different languages that link sources of separate communities. This was done without a resource intensive semantic analysis in almost real-time. The only limitations were observed within languages that do not separate their words with spaces such as Chinese or Japanese.

9.2.2 Number of Links

The number of new connections created for the three different datasets are displayed in Figure 9.3.

Fifteen keywords were extracted from each of the 50 C&T conference abstracts and from the 50 most recent Cooperation Systems Center Munich (CSCM) paper abstracts. Due to the short length of the Twitter messages, it seemed reasonable to only extracted a maximum of four keywords per tweet. For all three test sets, links were created between persons and contents (CC, PC, PP) by comparing the extracted keywords for each of these. The minimum number of matching keywords for a connection to be established varied between one and four. Figure 9.3 shows the number of connections for each of the test sets and the required number of keywords to connect. The graphs clearly indicate that the highest number of connections is created when only one matching keyword is required. With a threshold of four matching keywords the number of connections decreases significantly.

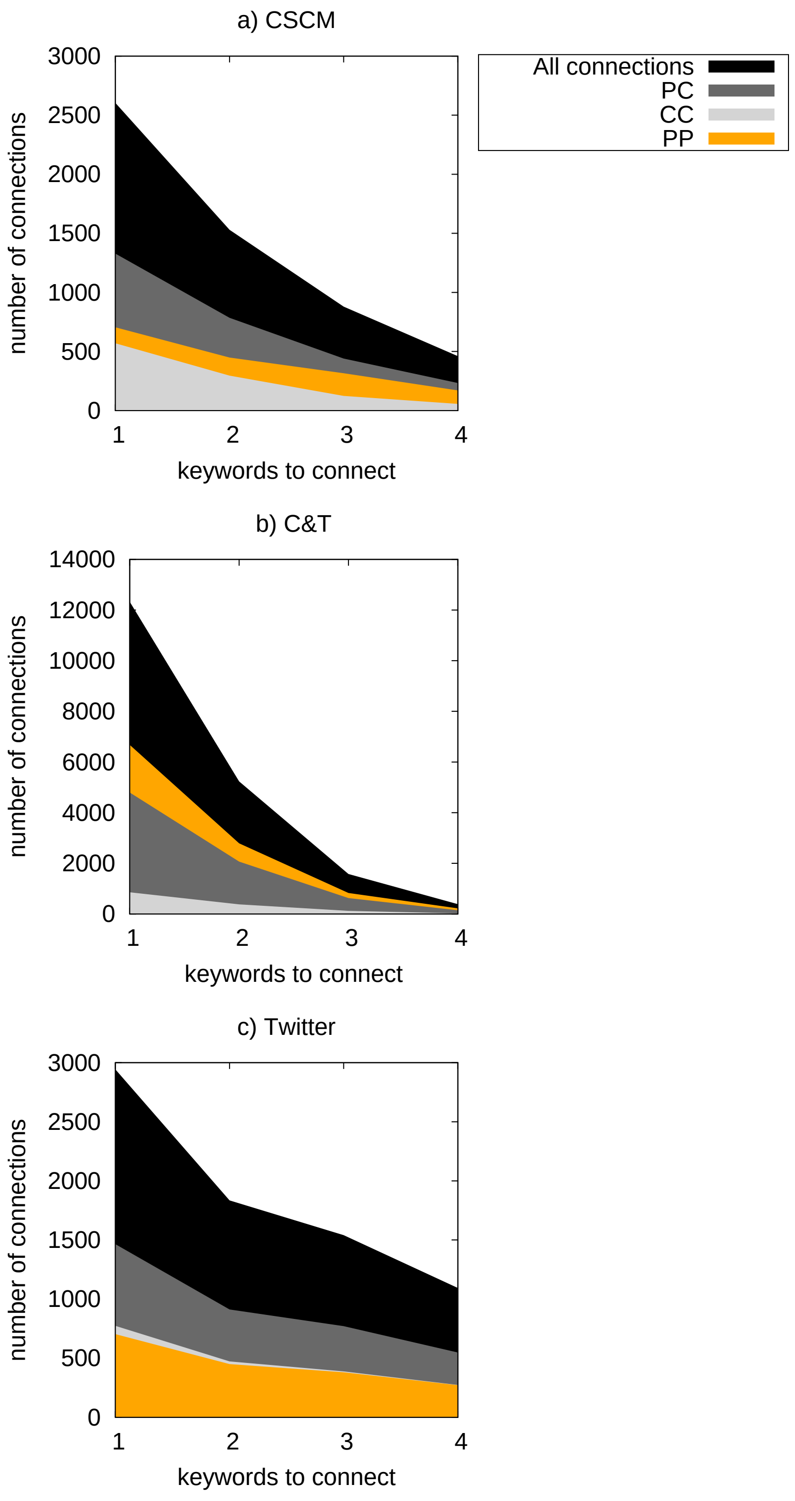


Figure 9.3: Number of connections based on number of keywords to connect

The high number of PP-connections for the 50 C&T abstracts is due to the fact that a total of 142 authors and coauthors were identified throughout the analysis, whereas only 48 persons were identified as contributors of the papers of the CSCM research group. Many of these persons wrote the papers together and therefore have pre-existing explicit connections. This results in less PP-connections.

A total of 356 Twitter users emitted the 400 tweets, that were analyzed. The high retweet rate in this test set leads to a large number of PC and CC connections. This indicates that the retweets were identified correctly and that a lot of PP connections were subsequently created. Tweets with identical content, that have not been tagged as retweets led to additional PP connections.

The average runtime of the extraction algorithm was only a few seconds. This can be considered almost real-time since the update procedure is a constantly running background process.

9.3 Visualization

In a collaboration with Peter Lachenmaier and Martin Burkhard we have implemented a visualization for the links within the CommunityMashup. Figure 9.4 shows this visualization for a sample dataset. The software is web-based and has been implemented with the JavaFX platform. It has been displayed on portable mobile devices as well as desktop computers, and large, touch-sensitive, wall-mounted screens. The objects are not static, they rather position themselved around the objects they are linked to and appear to be pulled by their gravity. This effect creates a very organic visualization.

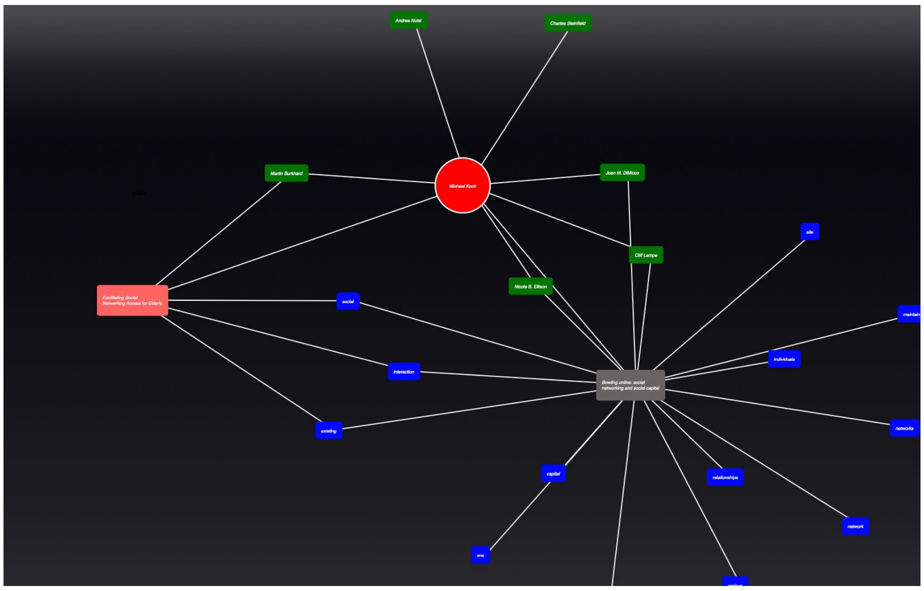


Figure 9.4: This visualization of the CommunityMashup displays users, their related content, and the associated keywords.

The objects displayed in Figure 9.4 are categorized as follows:

- red circle: person in focus
- green rectangle: person
- red rectangle: first author content
- grey rectangle: co-author content
- blue rectangle : keyword

Before the visualization is initialized, the user has to select a person from a list. Subsequently, the visualization displays a person-centered view with the focused person-object (red circle) and all the related contents for this person. The person Michael Koch is the person object in focus in Figure 9.4. A white line connects this object to two linked objects. The white line to the content "Facilitating Social Networking Access for Elderly" (red rectangle) shows, that he is the first author of that content. Furthermore, he is the second author for the content "Bowling online: social networking and social capital" (grey rectangle). The link to the grey rectangle is established via the person Nicole B. Ellison (green rectangle).

The displayed objects are constantly moving in order to encourage users to interact with them. All of the objects can be dragged across the screen and placed somewhere else. Keywords appear for a content in case a user hovers with the mouse over the content or if a user presses on the content on a touch-screen. As described above, objects can be linked due to an author/co-author relationship but also via a specified number of keywords.

This visualization may link people with contents they were not aware of before. These newly established connections can be created in almost real time and are visualized instantly. Not only content-person but also person-person relationships are created and displayed. By adding a social networking service that is constantly emitting content, this whole setup is continually evolving.

Chapter 10

Conclusions

This chapter contains a summary of the results and provides a discussion of the main contributions of this thesis.

10.1 Summary

The motivation for this work as well as the main scientific contributions are presented in Chapter 1. Additionally, I introduced the topic of IR with heuristics to the specific research field of single documents.

In Chapter 2 I provided general information about this specific research subject that is necessary for the understanding of the remainder of this thesis. The analysis of single documents requires segmentation approaches for the different retrieval algorithms. One of these segmentation approaches has been developed throughout this work. As one of my main objectives is context-independence, I perform minimal preprocessing. Furthermore, a number of language models and basic mathematical concepts were introduced in this chapter.

IR and IE are two major research areas in computer science that have been studied for many years. Chapter 3 provides an overview of related research for major fields: summarization, keyword extraction, combination approaches, and CPD.

A successful combination of different retrieval algorithms comprises all characteristics of a successful IR algorithm. Therefore, I provide formal definition of retrieval constraints for single documents in Chapter 4. Additionally, a selection of sample algorithms is presented in this chapter. These sample algorithms fit the demands of a fast, independent, and parameter-free keyword-extraction algorithm and are suitable for combination.

The most common approaches for combination of retrieval algorithms have been presented in Chapter 5. I have presented one approach based on the model of Divergence of Randomness and four other methods, that combine the results of the individually executed retrieval algorithms. These combinations allow for the composition of an algorithm that meets all the desirable retrieval constraints.

The algorithms in Chapter 4 are based on frequency information of single terms in a document. They don't consider the structure of the text, and they rely on a segmentation based on windows. In Chapter 6, I have described how the concept of entropy can provide further insights into the structure of texts. The basis for my CPD approach is the well known LZ77 compression algorithm, which has been described in that chapter. The fundamental principle of match lengths play a crucial role for the understanding of this algorithm and for the reference algorithm presented in Chapter 7. With the concept of entropy and the approximation technique based on match lengths, I was able to develop my own CPD algorithm, that has been presented in Chapter 7. This algorithm is based on tries and analyzes their structural data. The trie structure and the trie-transformation procedures were also described in that chapter. Additionally I have provided a general overview of my reference implementation.

Chapter 8 has revealed that a combination of algorithms may be superior to individual algorithms. Apparently, the combination method is less crucial than the combined algorithms. An example of a successful approach has been presented. Furthermore, I have shown, that the results of individual algorithms reveal details about their characteristics, which may lead to successful combinations of algorithms. The CPD algorithms has performed well on the test set of documents and has clearly identified the change-points. In addition, I have shown that it is faster than the reference algorithm and it can be applied to indefinite streams of text data.

10.2 Discussion

Single retrieval heuristics fail to encompass all information of a text that is potentially relevant for a keyword extraction process. In this thesis, I have presented and analyzed heuristics for IR and IE from single documents. I introduced state-of-the-art retrieval combination and ranking aggregation methods to combine well-known retrieval heuristics that work best with single documents. Moreover, I have evaluated these combination methods with real-world-examples. The individual compositions are based on retrieval constraints for single documents, that I have formally defined in this work. All algorithms are based on a common notation. In this work, I utilized PCA as a parameter-free and effective method for determining an optimal selection of retrieval heuristics for combination. This approach accounts for the properties of the heuristics as well as for the characteristics of the analyzed text. I have also introduced self-regulating windows to achieve more meaningful results.

To demonstrate the success of combinations, I have presented an efficient and flexible keyword extraction approach for arbitrary text documents. Its key features are independence from language, structure, and content. Unlike most other approaches, it does not rely on a training phase or extensive pre-processing steps. This algorithm satisfies the constraints of a successful retrieval algorithm as it combines the Helmholtz approach with information related to the burstiness of terms. I have exemplified the meaningfulness of the results and compared them to human-assigned keywords. Additionally, I have evaluated the compression ratio as well as the efficiency of this approach. The runtime scales linearly to document size and performs better than the well-known TF-IDF approach. Essentially it enables a fast perception of the content of a larger portion of text.

To extract more information from a document or text stream, I analyzed its structural composition, and I have presented a novel measure for information value within a text over a time series. This approach identifies entropy fluctuations within a text based on a history of predefined size. It is flexible, language-independent, and fits the needs for a fast performing application. I demonstrate its correctness with several telling examples, presented in a publication of a related approach, and showed its compelling performance with a benchmark algorithm.

Finally, I introduced an application of this approach by integrating it into the implementation of the CommunityMashup with Peter Lachenmaier. With the contribution of an efficient keyword extraction approach, it was possible to interlink information objects from different source services based on automatically extracted keywords. We have evaluated our implementation with three real-world datasets from different services, varying in length of the contents, community and topic. The new links revealed new connections between people and content that were previously unknown. Furthermore, we developed an interactive, platform-independent visualization approach that allows people to discover new knowledge, and new social interactions may be forged.

The Wider Scope

Recent applications such as *Summly* show that the retrieval and extraction of information is a current topic [123]. Supervised learning algorithms are extremely popular and most of the successful algorithms incorporate them. However, I clearly aim to focus the analysis on the text itself. Most of the research contributions either focus on the single algorithm itself, or on additional knowledge such as user feedback, large databases, or semantics. Combinations of heuristics have not been intensively studied.

These heuristics have been selected because they allow for keyword extraction. Keyword extraction is only one way to extract information from texts, but it is a very efficient way to obtain meaningful results in short time. Instead, Summly extracts sentences from texts.

My objective with this work was to extract as much information as possible out of a plain text, because sometimes a plain text is the only source available. First, I analyzed some of the most successful retrieval algorithms and found that they may supplement each other. The presented algorithm in Section 5.1.1 proves this assumption well. The retrieval constraints were a first step to a formal definition of the requirements of an optimal algorithm. To my knowledge, no such formal analysis and composition of retrieval heuristics for single documents has been done before.

Apparently, the field of single document analysis has not been extensively studied. Most of the retrieval algorithms have been applied to document collections instead of single documents. This is partially based on the Defense Advanced Research Projects Agency (DARPA) Message Understanding Conference (MUC) research projects. Due to the requirements of these algorithms, single documents have to be treated like a collection. The presented self-regulating window approach resulted out of this research question.

The accuracy of the IR heuristics is not always convincing. The quality of the results often depends on a good segmentation of the document and on the document itself. I experienced massive problems with the analysis of Tweets, because a large number of Retweets change the statistics of the text significantly and therefore strongly influence the retrieval results of a particular algorithm. Certain syntax had to be filtered manually. A learning algorithm could easily overcome that issue.

Some recent studies show that the structure of a text document may reveal valuable insights into the content [94]. The CPD approach can definitely identify some of these structures. A remarkable feature is its applicability to indefinite streams. Unfortunately, this prototype is not sensitive enough to detect less apparent changes in a texts. A more sensitive, automatic approach might be used for segmentation of documents as a preprocessing step before the analysis with heuristics.

It also has to be emphasized, that there is still room for improvement for the reference implementations that were implemented throughout this work. They serve the purpose of proof of concept and have a great potential for speed and efficiency optimization. For example, the trie data structure may be implemented more efficiently with the programming language C/C++ as it facilitates low-level access to memory [102].

10.3 Future Work

The presented combination approaches show that a combination of heuristics can be beneficial for the results of a retrieval algorithm. I also provide a PCA-based method to analyze the properties of the heuristics. If the properties of the single heuristics could be measured and compared to each other, a framework for an automatic composition of suitable algorithms would be possible. This framework can theoretically self-adjust, depending on the text source that has to be analyzed.

A fast and independent information extraction application can be extremely useful on small electronic devices such as smartphones and tablets. The development of a tag-cloud application on a mobile platform may provide the user with instant summaries to the live content on the screen.

A higher resolution for the CPD approach would be desirable. Experiments with only certain types of terms (stop words or function words) might provide finer results. The identification of the change-point can be optimized and may lead to an automatic segmentation of a document. This automatic partitioning could be the basis for a keyword extraction process. Additionally, an extensive study within a microblogging environment could be conducted to analyze the burst detection capabilities of this algorithm.

The first experiments with the CommunityMashup showed promising results. Nevertheless, this approach can be utilized for bigger scenarios. The presented visualization for large screens allows, for example, participants of a scientific conference to browse all conference submissions as well as the contents and profiles of all other participants. This keyword-based link creation could be extended by analyzing the social structure inside the aggregated data of the

CommunityMashup. Similarities between preferences or keywords of linked documents could be used to generate new links.

Bibliography

[1] *SigniTrend: Scalable Detection of Emerging Topics in Textual Streams by Hashed Significance Thresholds*, KDD '14, New York, NY, USA, August 2014. ACM.

[2] Andrew Albanese. Google book search grows. http://lj.libraryjournal.com/2007/06/industry-news/google-book-search-grows/, June 2007.

[3] James Allan, Jaime Carbonell, George Doddington, Jonathan Yamron, Yiming Yang, Brian Archibald, and Mike Scudder. Topic detection and tracking pilot study final report. In *Proceedings of the DARPA Broadcast News Transcription and Understanding Workshop*, pages 194–218, Lansdowne, VA, USA, February 1998.

[4] Giambattista Amati. *Probability Models for Information Retrieval based on Divergence from Randomness.* Thesis of the degree of doctor of philosophy, Department of Computing Science University of Glasgow, June 2003.

[5] Gianni Amati and Cornelis Joost Van Rijsbergen. Probabilistic models of information retrieval based on measuring the divergence from randomness. *ACM Trans. Inf. Syst.*, 20:357–389, October 2002.

[6] Ernesto D Avanzo, Bernardo Magnini, and Alessandro Vallin. Keyphrase Extraction for Summarization Purposes: The LAKE System at DUC-2004. In *Document Understanding Conferences*, Boston, USA, 2004.

[7] Ricardo A. Baeza-Yates and Berthier A. Ribeiro-Neto. *Modern Information Retrieval – the concepts and technology behind search, Second edition.* Pearson Education Ltd., Harlow, England, 2011.

[8] Alexander A. Balinsky, Helen Y. Balinsky, and Steven J. Simske. On helmholtz's principle for documents processing. In *Proc. of the* 10^{th} *ACM symp. on Document engineering*, DocEng '10, page 283, New York, NY, USA, 2010. ACM.

[9] Andrea Baronchelli, Emanuele Caglioti, and Vittorio Loreto. Measuring complexity with zippers. *European Journal of Physics*, 26(5):S69, 2005.

[10] Michèle Basseville and Igor V. Nikiforov. *Detection of abrupt changes: theory and application.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1993.

[11] Dario Benedetto, Emanuele Caglioti, and Vittorio Loreto. Language trees and zipping. *Phys. Rev. Lett.*, 88:048702, Jan 2002.

[12] A. Beutelspacher. *Kryptologie.* Vieweg, 2007.

[13] G. K. Bhattacharyya and Richard A. Johnson. Nonparametric tests for shift at an unknown time point. *The Annals of Mathematical Statistics*, 39(5):pp. 1731–1743, 1968.

[14] Albert Bifet. Adaptive stream mining: Pattern learning and mining from evolving data streams. In *Proceedings of the 2010 conference on Adaptive Stream Mining: Pattern Learning and Mining from Evolving Data Streams*, pages 1–212, Amsterdam, The Netherlands, The Netherlands, 2010. IOS Press.

[15] Roi Blanco and Alvaro Barreiro. Probabilistic document length priors for language models. In Craig Macdonald, Iadh Ounis, Vassilis Plachouras, Ian Ruthven, and RyenW. White, editors, *Advances in Information Retrieval*, volume 4956 of *Lecture Notes in Computer Science*, pages 394–405. Springer Berlin Heidelberg, 2008.

[16] T. Bohne and U.M. Borghoff. Data fusion: Boosting performance in keyword extraction. In *Engineering of Computer Based Systems (ECBS), 2013 20th IEEE International Conference and Workshops on the*, pages 166–173, April 2013.

[17] Thomas Bohne and Uwe M. Borghoff. Beyond frequency: Structural analysis of texts. In *Proc. 14th Int. Conf. on Computer-Aided System Theory*, Eurocast 2013, 2013.

[18] Thomas Bohne and UweM. Borghoff. Detecting information structures in texts. In Roberto Moreno-Díaz, Franz Pichler, and Alexis Quesada-Arencibia, editors, *Computer Aided Systems Theory - EUROCAST 2013*, volume 8112 of *Lecture Notes in Computer Science*, pages 467–474. Springer Berlin Heidelberg, 2013.

[19] Thomas Bohne, Sebastian Rönnau, and Uwe M. Borghoff. Efficient keyword extraction for meaningful document perception. In *Proceedings of the 11th ACM symposium on Document engineering*, DocEng '11, pages 185–194, New York, NY, USA, 2011. ACM.

[20] U.M. Borghoff, P. Rödig, J. Scheffczyk, and L. Schmitz. *Long-Term Preservation of Digital Documents: Principles and Practices.* Physica-Verlag, 2007. ISBN 9783540336402.

[21] B.E. Brodsky and B.S. Darkhovsky. *Nonparametric Methods in Change Point Problems.* Mathematics and Its Applications. Springer, 1993.

[22] Zdeněk Češka and Chris Fox. The influence of text pre-processing on plagiarism detection. In *Proceedings of the International Conference on Recent Advances in Natural Language Processing (RANLP 2009)*, Borovets, Bulgaria, 2009.

[23] Zdenek Ceska and Chris Fox. The influence of text pre-processing on plagiarism detection. In *Proceedings of the International Conference on Recent Advances in Natural Language Processing*, pages 55–59. RANLP 2009 Organizing Committee / ACL, 2009.

[24] H. Chernoff and S. Zacks. Estimating the current mean of a normal distribution which is subjected to changes in time. *The Annals of Mathematical Statistics*, 35(3):pp. 999–1018, 1964.

[25] Kenneth W. Church and William A. Gale. Poisson mixtures. *Natural Language Engineering*, 1:163–190, 1995.

[26] R. Clausius. *On the Motive Power of Heat, and on the Laws which Can be Deduced from it for the Theory of Heat.* Annalen der Physik. Dover, 1960.

[27] Stéphane Clinchant and Eric Gaussier. The bnb distribution for text modeling. In *Proceedings of the IR Research, 30th European Conference on Advances in Information Retrieval*, ECIR'08, pages 150–161, Berlin, Heidelberg, 2008. Springer-Verlag.

[28] Stéphane Clinchant and Eric Gaussier. Bridging language modeling and divergence from randomness models: A log-logistic model for ir. In Leif Azzopardi, Gabriella Kazai, Stephen Robertson, Stefan Rüger, Milad Shokouhi, Dawei Song, and Emine Yilmaz, editors, *Advances in Information Retrieval Theory*, volume 5766 of *Lecture Notes in Computer Science*, pages 54–65. Springer Berlin Heidelberg, 2009.

[29] Stéphane Clinchant and Eric Gaussier. Retrieval constraints and word frequency distributions a log-logistic model for ir. *Inf. Retr.*, 14:5–25, February 2011. ISSN 1386-4564.

[30] John M. Conroy and Dianne P. O'leary. Text summarization via hidden Markov models. In *Proc. of the* 24th *ann. int. ACM SIGIR conf. on Research and development in information retrieval - SIGIR '01*, pages 406–407, New York, NY, USA, September 2001. ACM.

[31] Copernic Summarizer. `http://www.copernic.com/en/products/summarizer/`, November 2013.

[32] Oxford English Corpus. The oec: Facts about the language. `http://www.oxforddictionaries.com/words/the-oec-facts-about-the-language`, November 2013.

[33] T.M. Cover and J.A. Thomas. *Elements of Information Theory.* Wiley, 2012.

[34] Cruxbot. `http://www.cruxbot.com/`, November 2013.

[35] J. Lafferty D. Blei. Topic models. In A. Srivastava and M. Sahami, editors, *Text Mining: Classification, Clustering, and Applications*, CRC Data Mining and Knowledge Discovery Series. Chapman & Hall, 2019.

[36] Pierre Simon de Laplace. *Essai philosophique sur les probabilités. English; A philosophical essay on probabilities.* J. Wiley, 1902.

[37] Agns Desolneux, Lionel Moisan, and Jean-Michel Morel. *From Gestalt Theory to Image Analysis: A Probabilistic Approach*. Springer Publishing Company, Incorporated, 1st edition, 2007. ISBN 0387726357, 9780387726359.

[38] Allen B. Downey. A novel changepoint detection algorithm, December 2008.

[39] Werner Ebeling and Thorsten Pöschel. Entropy and Long range correlations in literary English. *Europhysics Letters (EPL)*, 26(4):241–246, September 1993.

[40] H. P. Edmundson. New Methods in Automatic Extracting. *Journal of the ACM*, 16(2):264–285, April 1969.

[41] Oren Etzioni, Michael Cafarella, Doug Downey, Ana-Maria Popescu, Tal Shaked, Stephen Soderland, Daniel S. Weld, and Alexander Yates. Unsupervised named-entity extraction from the web: An experimental study. *ARTIFICIAL INTELLIGENCE*, 165:91–134, 2005.

[42] Hui Fang, Tao Tao, and ChengXiang Zhai. A formal study of information retrieval heuristics. In *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '04, pages 49–56, New York, NY, USA, 2004. ACM.

[43] William Feller. *An Introduction to Probability Theory and Its Applications, Vol. 2*. Wiley, 1967.

[44] Eibe Frank, Gordon W. Paynter, Ian H. Witten, Carl Gutwin, and Craig G. Nevill-Manning. Domain-specific keyphrase extraction. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'99, pages 668–673, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[45] Edward Fredkin. Trie memory. *Communications of the ACM*, 3(9): 490–499, September 1960.

[46] Gabriel Pui Cheong Fung, Jeffrey Xu Yu, Philip S. Yu, and Hongjun Lu. Parameter free bursty events detection in text streams. In *Proceedings of the 31st International Conference on Very Large Data Bases*, VLDB '05, pages 181–192. VLDB Endowment, 2005.

[47] Yanbin Gao and Gang Zhao. Knowledge-based information extraction: A case study of recognizing emails of nigerian frauds. In Andrés Montoyo, Rafael Muńoz, and Elisabeth Métais, editors, *Natural Language Processing and Information Systems*, volume 3513 of *Lecture Notes in Computer Science*, pages 161–172. Springer Berlin Heidelberg, 2005.

[48] Yun Gao, Ioannis Kontoyiannis, and Elie Bienenstock. Estimating the entropy of binary time series: Methodology, some theory and a simulation study. *Entropy*, 10(2):71–99, 2008.

[49] Andy Georges, Dries Buytaert, and Lieven Eeckhout. Statistically rigorous java performance evaluation. *SIGPLAN Not.*, 42(10):57–76, October 2007.

[50] Javier Girón, Josep Ginebra, and Alex Riba. Bayesian analysis of a multinomial sequence and homogeneity of literary style. *The American Statistician*, 59(1):pp. 19–30, 2005.

[51] Peter Grassberger. Estimating the information content of symbol sequences and efficient codes. *IEEE Transactions on Information Theory*, pages 669–669, 1989.

[52] Ben Hanrahan, Guillaume Bouchard, Gregorio Convertino, Thiebaud Weksteen, Nicholas Kong, Cedric Archambeau, and Ed H. Chi. Mail2wiki: Low-cost sharing and early curation from email to wikis. In *Proceedings of the 5th International Conference on Communities and Technologies*, C&T '11, pages 98–107, New York, NY, USA, 2011. ACM.

[53] Stephen P. Harter. A probabilistic approach to automatic keyword indexing. Part I. On the Distribution of Specialty Words in a Technical Literature. *Journal of the American Society for Information Science*, 26 (4):197–206, July 1975.

[54] Qi He, Kuiyu Chang, and Ee-Peng Lim. Using Burstiness to Improve Clustering of Topics in News Streams. In *Data Mining, 2007. ICDM 2007. Seventh IEEE International Conference on*, pages 493–498, 2007.

[55] Qi He, Kuiyu Chang, Ee-Peng Lim, and A. Banerjee. Keep it simple with time: A reexamination of probabilistic topic detection models. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 32(10):1795 –1808, oct. 2010.

[56] Marti A. Hearst. Multi-paragraph segmentation of expository text. In *Proceedings of the 32nd annual meeting on Association for Computational Linguistics*, ACL '94, pages 9–16, Stroudsburg, PA, USA, 1994. Association for Computational Linguistics.

[57] Marti A Hearst. Untangling text data mining. In *Proceedings of ACL'99: the 37th Annual Meeting of the Association for Computational Linguistics*, pages 3–10. Association for Computational Linguistics, 1999.

[58] Marti A. Hearst and Christian Plaunt. Subtopic structuring for full-length document access. In *Proceedings of the 16th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '93, pages 59–68, New York, NY, USA, 1993. ACM.

[59] David V. Hinkley. Inference about the change-point in a sequence of random variables. *Biometrika*, 57(1):pp. 1–17, 1970.

[60] R.N. Horspool and G.V. Cormack. Constructing word-based text compression algorithms. In *Data Compression Conference, 1992. DCC '92.*, pages 62 –71, March 1992.

[61] H. Hotelling. Analysis of complex statistical variables into principal components. *Journal of Educational Psychology*, 24(6):417–441, September 1933.

[62] Oliver Johnson, Dino Sejdinovic, James Cruise, Ayalvadi Ganesh, and Robert J. Piechocki. Non-parametric CPD using string matching algorithms. *CoRR*, 2011.

[63] R.A. Johnson and D.W. Wichern. *Applied Multivariate Statistical Analysis*. Applied Multivariate Statistical Analysis. Pearson Prentice Hall, 2007.

[64] Spärck Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, 28(1):11–21, 1972.

[65] Marcin Kaszkiel and Justin Zobel. Effective ranking with arbitrary passages. *Journal of the American Society for Information Science & Technology*, 52(4):344 – 364, 2001.

[66] Jon Kleinberg. Bursty and hierarchical structure in streams. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '02, pages 91–101, New York, NY, USA, 2002. ACM.

[67] Jon M. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5):604–632, September 1999.

[68] Donald E. Knuth. *The Art of Computer Programming, Volume 3: Sorting and Searching (2nd Ed.)*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1998.

[69] Andrey N. Kolmogorov. Three approaches to the quantitative definition of information. *Problems in Information Transmission*, 1:1–7, 1965.

[70] Manu Konchady. *Text Mining Application Programming*. Charles River Media, Inc., Rockland, MA, USA, 2006.

[71] I. Kontoyiannis, P.H. Algoet, Yu.M. Suhov, and A.J. Wyner. Nonparametric entropy estimation for stationary processes and random fields, with applications to english text. *Information Theory, IEEE Transactions on*, 44(3):1319 –1327, May 1998.

[72] Jan H. Kroeze, Machdel C. Matthee, and Theo J. D. Bothma. Differentiating data- and text-mining terminology. In *Proceedings of the 2003 annual research conference of the South African institute of computer scientists and information technologists on Enablement through technology*, SAICSIT '03, pages 93–101, Republic of South Africa, 2003. South African Institute for Computer Scientists and Information Technologists.

[73] Niraj Kumar and Kannan Srinathan. Automatic keyphrase extraction from scientific documents using N-gram filtration technique. In *Proc. of the 8th ACM symp. on Document engineering - DocEng '08*, page 199, New York, NY, USA, 2008. ACM.

[74] Peter Lachenmaier, Florian Ott, and Michael Koch. Model-driven development of a person-centric mashup for social software. *Social Network Analysis and Mining*, 3(2):193–207, 2013.

[75] Thomas K. Landauer, Peter W. Foltz, and Darrell Laham. An Introduction to Latent Semantic Analysis. *Discourse Processes*, 25: 259–284, 1998.

[76] Chung-Hong Lee, Chih-Hong Wu, and Tzan-Feng Chien. Burst: A dynamic term weighting scheme for mining microblogging messages. In Derong Liu, Huaguang Zhang, Marios Polycarpou, Cesare Alippi, and Haibo He, editors, *Advances in Neural Networks – ISNN 2011*, volume 6677 of *Lecture Notes in Computer Science*, pages 548–557. Springer Berlin / Heidelberg, 2011.

[77] Chang-Jin Li and Hui-Jian Han. Keyword extraction algorithm based on principal component analysis. In Ran Chen, editor, *Intelligent Computing and Information Science*, volume 135 of *Communications in Computer and Information Science*, pages 503–508. Springer Berlin Heidelberg, 2011.

[78] Wentian Li and Pedro Miramontes. Fitting ranked english and spanish letter frequency distribution in us and mexican presidential speeches. *Journal of Quantitative Linguistics*, 18(4):359–380, 2011.

[79] Marina Litvak and Mark Last. Graph-based keyword extraction for single-document summarization. In *MMIES '08 Proc. of the Workshop on Multi-source Multilingual Information Extraction and Summarization*, pages 17–24, August 2008.

[80] Robert M. Losee. A discipline independent definition of information. *J. Am. Soc. Inf. Sci.*, 48(3):254–269, February 1997.

[81] G. Louloudis, A.L. Kesidis, and B. Gatos. Efficient word retrieval using a multiple ranking combination scheme. In *Document Analysis Systems (DAS), 2012 10th IAPR International Workshop on*, pages 379 –383, march 2012.

[82] D.G. Lowe. *Perceptual organization and visual recognition.* Kluwer international series in engineering and computer science: Robotics. Kluwer Academic Publishers, 1985.

[83] H. P. Luhn. A statistical approach to mechanized encoding and searching of literary information. *IBM J. Res. Dev.*, 1(4):309–317, October 1957.

[84] Dmitrii Y. Manin. On the nature of long-range letter correlations in texts. *CoRR*, abs/0809.0103, June 2008.

[85] Christopher D. Manning and Hinrich Schütze. *Foundations of Statistical Natural Language Processing.* MIT Press, Cambridge, MA, USA, 1999.

[86] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval.* Cambridge University Press, Cambridge, Juli 2008.

[87] Osama Mansour, Mustafa Abusalah, and Linda Askenäs. Wiki-based community collaboration in organizations. In *Proceedings of the 5th International Conference on Communities and Technologies*, C&T '11, pages 79–87, New York, NY, USA, 2011. ACM.

[88] Ali Mehri and Amir H. Darooneh. The role of entropy in word ranking. *Physica A: Statistical Mechanics and its Applications*, 390(18–19):3157 – 3163, 2011.

[89] Saket Mengle and Nazli Goharian. Detecting hidden passages from documents. In *SIAM Conference on Data Mining (SDM 2008) Workshop*, April 2008.

[90] Saket Mengle and Nazli Goharian. Passage detection using text classification. *Journal of the American Society for Information Science and Technology*, 60(4):814–825, April 2009.

[91] David R. H. Miller, Tim Leek, and Richard M. Schwartz. A hidden markov model information retrieval system. In *Proceedings of the 22Nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '99, pages 214–221, New York, NY, USA, 1999. ACM.

[92] MIT Senseable City Lab. http://senseable.mit.edu/tweetbursts/, August 2015.

[93] Marcelo A. Montemurro and Damián H. Zanette. Entropic analysis of the role of words in literary texts. *CoRR*, 2001.

[94] Marcelo A. Montemurro and Damián H. Zanette. Keywords and co-occurrence patterns in the voynich manuscript: An information-theoretic analysis. *PLoS ONE*, 8(6):e66344, 06 2013.

[95] Marcelo A. Montemurro and Damián H. Zanette. Towards the quantification of the semantic information encoded in written language. *Advances in Complex Systems*, 13(2):135–153, 2010.

[96] M. E. J. Newman. Power laws, pareto distributions and zipf's law. *Contemporary Physics*, 2005.

[97] Martin A. Nowak, Natalia L. Komarova, and Partha Niyogi. Computational and evolutionary aspects of language. *Nature*, 417(6889): 611–617, June 2002.

[98] D.S. Ornstein and B. Weiss. Entropy and data compression schemes. *Information Theory, IEEE Transactions on*, 39(1):78–83, 1993.

[99] Joseph O'Sullivan and Adam Smith. All booked up. http://googleblog.blogspot.de/2004/12/all-booked-up.html, January 2014. Google Official Blog.

[100] The Pagerank, Citation Ranking, and Bringing Order. The PageRank Citation Ranking: Bringing Order to the Web. *World Wide Web Internet And Web Information Systems*, pages 1–17, 1998.

[101] David D. Palmer. Text pre-processing. In Nitin Indurkhya and Fred J. Damerau, editors, *Handbook of Natural Language Processing, Second Edition*. CRC Press, Taylor and Francis Group, Boca Raton, FL, 2010.

[102] Adam Pauls and Dan Klein. Faster and smaller n-gram language models. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1*, HLT '11, pages 258–267, Stroudsburg, PA, USA, 2011. Association for Computational Linguistics.

[103] K. Pearson. On lines and planes of closest fit to systems of points in space. *Philosophical Magazine*, 2(6):559–572, 1901.

[104] Jay M. Ponte and W. Bruce Croft. A language modeling approach to information retrieval. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '98, pages 275–281, New York, NY, USA, 1998. ACM.

[105] H.V. Poor and O. Hadjiliadis. *Quickest Detection*. Cambridge books online. Cambridge University Press, 2009.

[106] K.R. Popper. *The Logic of Scientific Discovery*. Classics Series. Routledge, 2002.

[107] M. F. Porter. An algorithm for suffix stripping. In Karen Sparck Jones and Peter Willett, editors, *Readings in information retrieval*, pages 313–316. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1997.

[108] R.C. Prati. Combining feature ranking algorithms through rank aggregation. In *Neural Networks (IJCNN), The 2012 International Joint Conference on*, pages 1–8, june 2012.

[109] Oxford University Press. Oxford dictionary. http://oxforddictionaries.com/words/what-is-the-frequency-of-the-letters-of-the-alphabet-in-english, August 2015.

[110] Project Gutenberg Literary Archive Foundation. http://www.gutenberg.org/, August 2015.

[111] L R Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2): 257–286, 1989.

[112] Anand Rajaraman and Jeffrey David Ullman. *Mining of massive datasets*. Cambridge University Press, Cambridge, 2012.

[113] Benjamin Reilly. Social choice in the south seas: Electoral innovation and the borda count in the pacific island countries. *International Political Science Review*, 23(4):355–372, 2002.

[114] Alex Riba and Josep Ginebra. Diversity of vocabulary and homogeneity of literary style. *Journal of Applied Statistics*, 33(7):729–741, 2006.

[115] S. E. Robertson and Sparck K. Jones. Relevance weighting of search terms. *Journal of the American Society for Information Science*, 27(3): 129–146, 1976.

[116] S. E. Robertson, S. Walker, S. Jones, M. M. Hancock-Beaulieu, and M. Gatford. Okapi at TREC-3. In *Overview of the Third Text REtrieval Conference (TREC–3)*, pages 109–126. Gaithersburg, MD: NIST, 1996.

[117] S.E. Robertson, S. Walker, M. Beaulieu, and Peter Willett. Okapi at trec-7: Automatic ad hoc, filtering, vlc and interactive track. *In*, 21: 253–264, 1999.

[118] Stephen Robertson. Understanding inverse document frequency: on theoretical arguments for IDF. *Journal of Documentation*, 60(5):503–520, 2004.

[119] Sebastian Rönnau and Uwe Borghoff. Xcc: change control of xml documents. *Computer Science - Research and Development*, pages 1–17, 2010.

[120] G. Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. *Communications of the ACM*, 18:613–620, November 1975. ISSN 0001-0782.

[121] Gerard Salton, J. Allan, and Chris Buckley. Approaches to passage retrieval in full text information systems. In *Proceedings of the 16th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '93, pages 49–58, New York, NY, USA, 1993. ACM.

[122] Markus Schulze. A new monotonic, clone-independent, reversal symmetric, and condorcet-consistent single-winner election method. *Social Choice and Welfare*, 36:267–303, 2011.

[123] Ami Sedghi. Summly founder: teen entrepreneurs have the net advantage. http://www.theguardian.com/technology/2013/oct/21/summly-founder-teen-entrepreneurs-nick-daloisio, October 2013.

[124] C. E. Shannon. A mathematical theory of communication. *Bell system technical journal*, 27, 1948.

[125] P.C. Shields. *The Ergodic Theory of Discrete Sample Paths*. Graduate studies in mathematics. American Mathematical Society, 1996.

[126] Jonathon Shlens. A tutorial on principal component analysis. In *Systems Neurobiology Laboratory, Salk Institute for Biological Studies*, 2005.

[127] Amit Singhal. Modern Information Retrieval : A Brief Overview. *IEEE Data Engineering Bulletin 24*, pages 35–43, 2001.

[128] Amit Singhal, Chris Buckley, and Mandar Mitra. Pivoted document length normalization. In *Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '96, pages 21–29, New York, NY, USA, 1996. ACM.

[129] J. Sivic and A Zisserman. Efficient visual search of videos cast as text retrieval. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 31(4):591–606, April 2009.

[130] Josef Sivic and Andrew Zisserman. Video google: A text retrieval approach to object matching in videos. In *Proceedings of the Ninth IEEE International Conference on Computer Vision - Volume 2*, ICCV '03, Washington, DC, USA, 2003. IEEE Computer Society.

[131] C. Soanes and A. Stevenson, editors. *The concise Oxford dictionary.* Oxford University Press, 11. ed., rev. edition, 2006.

[132] Efstathios Stamatatos. Plagiarism detection based on structural information. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, CIKM '11, pages 1221–1230, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0717-8.

[133] Efstathios Stamatatos. Plagiarism detection based on structural information. In *Proceedings of the 20th ACM International Conference on Information and Knowledge Management*, CIKM '11, pages 1221–1230, New York, NY, USA, 2011. ACM.

[134] Heather A. Stark. What do paragraph markings do?. *Discourse Processes*, 11(3):275 – 303, 1988.

[135] State of The Union Address Library. http://stateoftheunionaddress.org/, August 2015.

[136] Brian Stelter. He has millions and a new job at yahoo. soon, he'll be 18. *New York Times*, 11, mar 2013.

[137] Danie Stolte. Experts determine age of book 'nobody can read'. http://phys.org/news/2011-02-experts-age.html, 2011. PhysOrg.

[138] Gilbert Strang. Introduction to linear algebra. *Cambridge Publication*, 2003.

[139] Russell Swan and James Allan. Automatic generation of overview timelines. In *Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '00, pages 49–56, New York, NY, USA, 2000. ACM.

[140] Alexander G. Tartakovsky, Boris L. Rozovskii, Rudolf B. Blažk, and Kim Hongjoong. A novel approach to detection of intrusions in computer networks via adaptive sequential and batch-sequential CPD methods. *IEEE Transactions on Signal Processing*, 54(9):3372 – 3382, 2006.

[141] Zhi Teng, Ye Liu, Fuji Ren, and Seiji Tsuchiya. Single Document Summarization Based on Local Topic Identification and Word Frequency. In 7^{th} *Mexican Int. Conf. on Artificial Intelligence*, pages 37–41. IEEE, October 2008.

[142] TextTeaser. http://www.textteaser.com/, November 2013.

[143] Nitin Thaper. Using compression for source baed classification of text. Master's thesis, Massachusetts Institute of Technology, 2001.

[144] The Graph Visualization Software. Graphviz. http://www.graphviz.org/, August 2015.

[145] Takashi Tomokiyo and Matthew Hurst. A language model approach to keyphrase extraction. In *Proceedings of the ACL 2003 Workshop on Multiword Expressions: Analysis, Acquisition and Treatment - Volume 18*, MWE '03, pages 33–40, Stroudsburg, PA, USA, 2003. Association for Computational Linguistics.

[146] Topicmarks. http://techcrunch.com/2011/12/09/tagged-acquires-topicmarks/, August 2013.

[147] Jurij Toplak. Preferential voting: definition and classification. In *Annual Meeting of the Midwest Political Science Assication 67th Annual National Conference*, 2010.

[148] Jordi Turmo, Alicia Ageno, and Neus Català. Adaptive information extraction. *ACM Computing Surveys*, 38(2), July 2006.

[149] Peter D. Turney. Learning algorithms for keyphrase extraction. *Information Retrieval*, 2:303–336, May 2000. ISSN 1386-4564.

[150] Peter D. Turney and Patrick Pantel. From frequency to meaning: vector space models of semantics. *Journal of Artificial Intelligence Research*, 37: 141–188, January 2010. ISSN 1076-9757.

[151] Twitter. http://www.twitter.com/, August 2015.

[152] UK Twitter Blog. The olympics on twitter. https://blog.twitter.com/en-gb/2012/the-olympics-on-twitter, November 2012.

[153] Vinícius Rodrigues Uzêda, Thiago Alexandre Salgueiro Pardo, and Maria Das Graças Volpe Nunes. Evaluation of Automatic Text Summarization Methods Based on Rhetorical Structure Theory. In 8^{th} *Int. Conf. on Intelligent Systems Design and Applications*, pages 389–394. Ieee, November 2008.

[154] J. Ventura and J.F. da Silva. *Brain, Vision and AI*, chapter Ranking and Extraction of Relevant Single Words in Text. InTech, 2008.

[155] J. W. von Goethe and David Luke. *Faust: Part One*. Oxford World's Classics. OUP Oxford, 2008. ISBN 9780191501258.

[156] Xiaojun Wan and Jianguo Xiao. Exploiting neighborhood knowledge for single document summarization and keyphrase extraction. *ACM Transactions on Information Systems*, 28(2):1–34, May 2010.

[157] Xing Wei. *Topic Models in Information Retrieval*. Ir, University of Massachusetts, August 2007.

[158] Colin Wheildon and M. Warwick. *Type & Layout: How Typography and Design Can Get Your Message Across–or Get in the Way*. Strathmoor Press, 1995.

[159] Ian H. Witten, Gordon W. Paynter, Eibe Frank, Carl Gutwin, and Craig G. Nevill-Manning. KEA: Practical Automatic Keyphrase Extraction. In *Proc. of the 4th ACM conf. on Digital libraries - DL '99*, pages 254–255, New York, NY, USA, August 1999. ACM.

[160] A.D. Wyner and J. Ziv. Some asymptotic properties of the entropy of a stationary ergodic data source with applications to data compression. *Information Theory, IEEE Transactions on*, 35(6):1250–1258, 1989.

[161] A.D. Wyner and J. Ziv. The sliding-window lempel-ziv algorithm is asymptotically optimal. *Proceedings of the IEEE*, 82(6):872–877, 1994.

[162] C. T. Yu and G. Salton. Precision weighting – an effective automatic indexing method. *J. ACM*, 23(1):76–88, January 1976.

[163] Chengxiang Zhai and John Lafferty. A study of smoothing methods for language models applied to ad hoc information retrieval. In *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '01, pages 334–342, New York, NY, USA, 2001. ACM.

[164] Chengxiang Zhai and John Lafferty. A study of smoothing methods for language models applied to information retrieval. *ACM Trans. Inf. Syst.*, 22:179–214, April 2004. ISSN 1046-8188.

[165] Wayne Xin Zhao, Baihan Shu, Jing Jiang, Yang Song, Hongfei Yan, and Xiaoming Li. Identifying event-related bursts via social media activities. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, EMNLP-CoNLL '12, pages 1466–1477, Stroudsburg, PA, USA, 2012. Association for Computational Linguistics.

[166] Hongding Zhou and Gary W. Slater. A metric to search for relevant words. *Physica A: Statistical Mechanics and its Applications*, 329(1–2):309 – 327, 2003.

[167] G. K. Zipf. *Selective Studies and the Principle of Relative Frequency in Language.* Harvard University Press, 1932.

[168] G K Zipf. *Human Behavior and the Principle of Least Effort*, volume 6. Addison-Wesley, 1949.

[169] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *Information Theory, IEEE Transactions on*, 23(3):337–343, 1977.

[170] J Zobel, R Wilkinson, and R Sackes-Davis. Efficient retrieval of partial documents. *Information Processing & Management*, 31(3):361 – 377, 1995.

Appendix A

The Reference Implementation

This appendix describes the detailed trie construction procedure for a sample and provides information about the reference implementation for this work.

My reference implementation contains two major parts:

- the keyword extraction and combination approaches and
- the CPD approach.

Both implementations were created with Java, version Java SE 1.7. Section A.2 depicts the trie construction and Section A.1 provides an overview of the code structure of the keyword extraction software part. The structure of my reference implementation for the change-point analysis is described in Section A.3.

A.1 Keyword Extraction Reference Implementation

The package *KeywordExtraction* contains the main Java code for the keyword extraction approaches and their combination methods. Figure A.1 provides an overview of the main classes that implement the analysis. The input text data is represented by the class *document*, that contains zero or more *Windows* – the classes are associated. Each *Window* has a *content* that consists of zero or more *Terms*.

The class *KeywordExtraction* uses the class *Preprocessor* to preprocess input text for the analysis. The class *Preprocessor* performs all preprocessing tasks, extracts raw text from websites and file formats such as Portable Document Format (PDF). Additionally, the class *Preprocessor* performs different segmentation techniques on the input text into windows.

The class *Analyzer* analyzes an instance of the class *Document*. It contains a number of different methods to count term frequencies and term distances. The

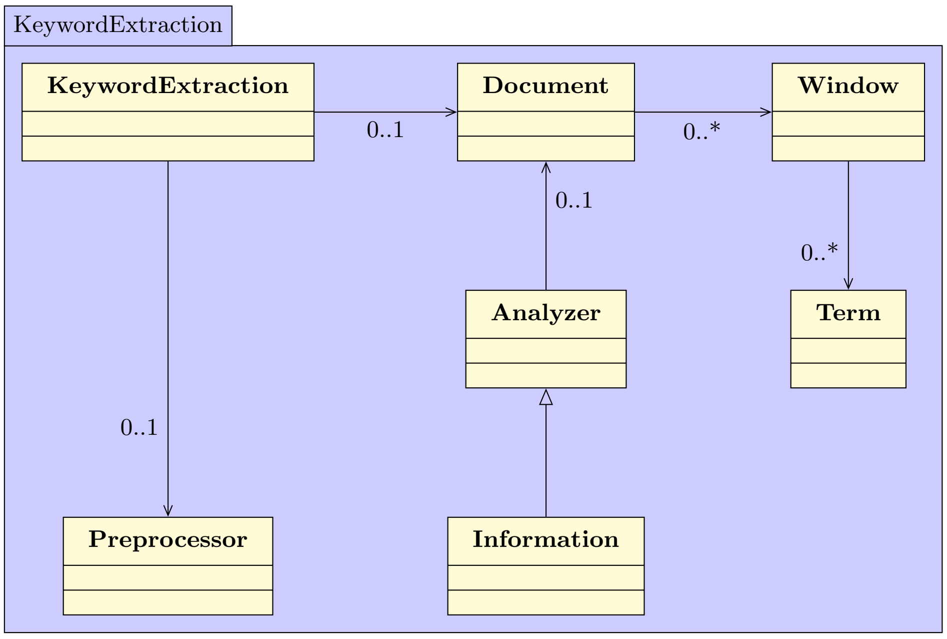


Figure A.1: This Unified Modeling Language (UML) class diagram provides an overview of the structure of the reference implementation for the combination of the heuristics.

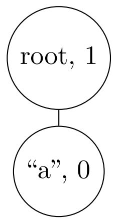
class *Information* is a specialization of the class *Analyzer*. It contains all described methods to calculate and to combine different term weights, based on the data generated by an instance of the class *Analyzer*.

The class *KeywordExtraction* instantiates the required classes for a keyword extraction task. It coordinates the processes necessary for a certain keyword extraction algorithm for a specific type of input data.

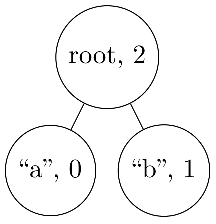
A.2 Detailed description of the Trie-Structure

This section provides a detailed description of the input procedure of the trie-based dictionary described in Section 7.2. At the beginning, the trie is empty and contains the root node only. At that time, the root node is the only leaf node in the trie and the trie has to be built up. The input sequence "abrakadabra" is read character by character.

Figure A.2a shows the trie after the first character "a" has been read. After adding "a" to the trie, the time τ of the trie increases and $\tau = 1$ for the root node. The same happens for the letters "b" and "c" in Figure A.2b and A.3a.

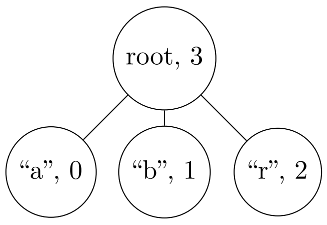


(a) The first character has been read.

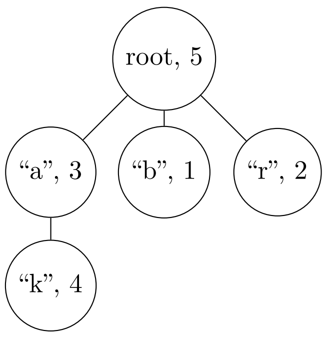


(b) Character "b" has been read.

As the trie already contains character "a", the next input sequence is "ak". Consequently, the node with the character "a" is updated and a child node with character "k" is attached to the parent with character "a". The time of the node with the character "a" is updated and the time of the root node increases to $\tau = 5$ because two nodes were added to the trie.

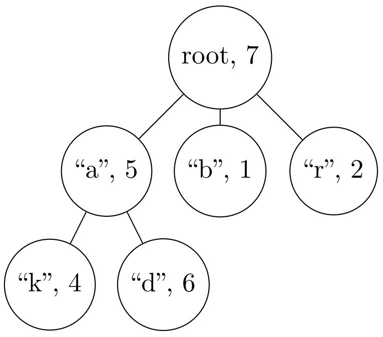


(a) Character "r" has been read.

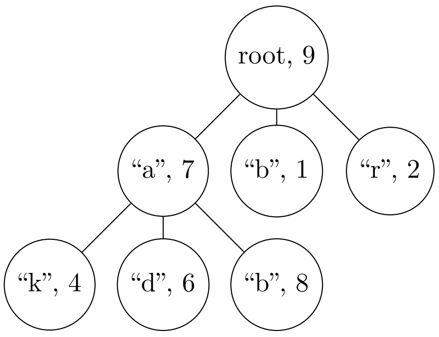


(b) "a" was recognized and "ak" has been read.

Since the trie contains the information of sequence "ak" already – the first letter is an "a" – the algorithm recognizes the "a" and reads the next character of the input sequence. The next character is a "d". Consequently, a child node "a" is added to the node with character "a" at level one. The following step is very similar to the last one: the sequence "ab" is added to the trie. Whenever the node with character "a" gets another child node, its lifetime is updated.

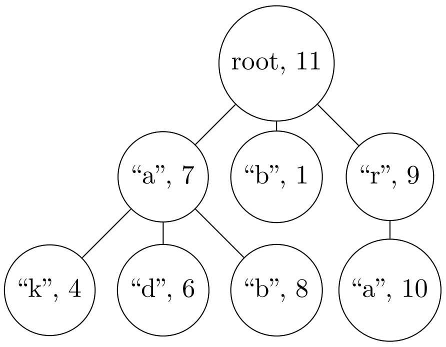


(a) “a” was recognized and “ad” has been read.



(b) Character “a” is recognized in the trie and “rb” is read and inserted into the trie.

The next character of the input sequence is “r”. Since the trie contains a node at level one with the character “r”, the subsequent character of the input sequence can be read. As the node with the character “r” is a leaf node, a node with the character “a” is added as a child node. The complete trie is displayed in Figure A.5g. After reading the complete input sequence with 11 characters, the trie contains 8 nodes.



(g) Character “r” is recognized in the trie and “ra” is read and inserted into the trie. This figure shows the dictionary trie after processing the input sequence “abrakadabra”.

A.3 Structure of the Reference Implementation for Change-Point Analysis

This section provides a general overview of the reference implementation for this thesis. The focus here is CPD. The UML class diagram in Figure A.6 shows the main classes in the package *CPD*. The classes *Document*, *Window*, and *Term* contain the input data, similar to the ones in Section A.1.

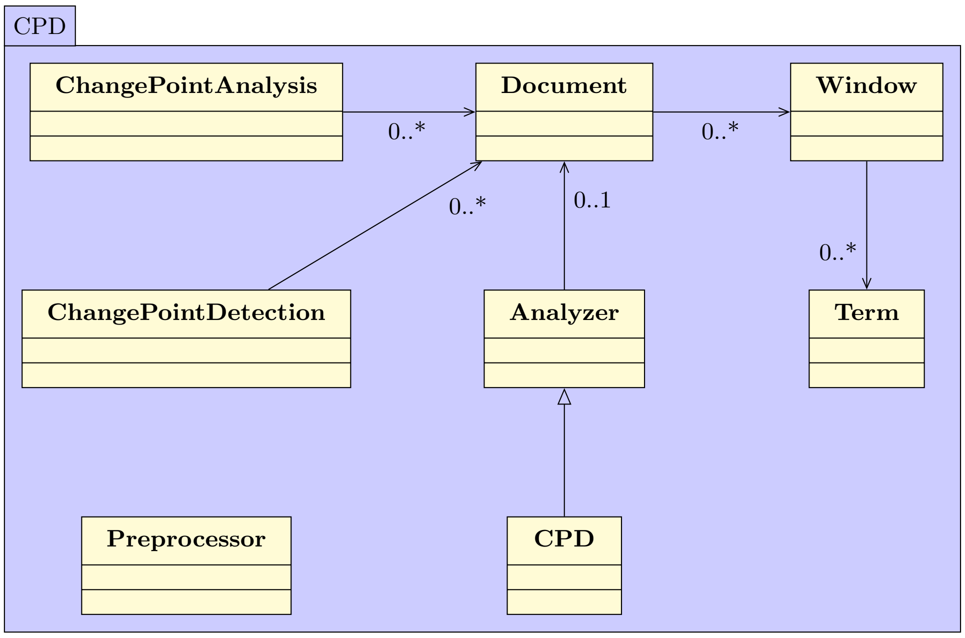


Figure A.6: This UML Class diagram provides an overview of the structure of the reference implementation of the CPD algorithm.

The class *ChangePointAnalysis* coordinates the change-point analysis and instantiates all required classes for a specific approach. The class *Preprocessor* performs all necessary preprocessing steps for a document if required. My main approaches are performed by the methods of class *CPD*, which is a specialization of the class *Analyzer*. The class *Analyzer* performs document analysis and provides different frequency data for an instance of the class *Document*.

The reference approach described in Section 7.1 is implemented with the class *ChangePointDetection*. This class does not require and of the analyses performed by the class *Analyzer*.

The trie data structure is implemented in a separate package *Trie*, which is presented in Section 7.6. Different types of tries have been implemented in this package:

- a trie that grows while reading input data,
- a tries that is based on a window of fixed size, and
- an adaptive trie with a decay value for its nodes.

The different algorithms for CPD and the different types of tries can utilized for character or term-based CPD.

Acronyms

ACM Association for Computing Machinery.

API application programming interface.

BC Borda Count.

BIM Binary Independence Model.

bpc bits per character.

C&T Communities and Technologies.

CC Content-Content.

CPD change-point detection.

CSCM Cooperation Systems Center Munich.

CUSUM cumulative sum.

DARPA Defense Advanced Research Projects Agency.

DocEng Symposium on Document Engineering.

ECBS Engineering of Computer Based Systems.

HTML HyperText Markup Language.

IDE integrated development environment.

IDF Inverse Document Frequency.

IE Information Extraction.

IR Information Retrieval.

IWF Inverse Window Frequency.

JIT Just-In-Time.

JVM Java Virtual Machine.

kB	Kilobyte.
KEA	Keyphrase Extraction Algorithm.
KL	Kullback-Leibler.
LM	language model.
LSA	Latent Semantic Analysis.
LSI	Latent Semantic Indexing.
LZ	Lempel-Ziv.
MB	Megabyte.
MUC	Message Understanding Conference.
NLP	natural language processing.
ODF	Open Document Format for Office Applications.
PC	Person-Content.
PCA	Principal Component Analysis.
PDF	Portable Document Format.
POS	part of speech.
PP	Person-Person.
SVD	Singular Value Decomposition.
TD	Topic Detection.
TDT	Topic Detection and Tracking.
TF	term frequency.
TF-IDF	Term Frequency - Inverse Document Frequency.
TF-IWF	Term Frequency - Inverse Window Frequency.
TOP	Term Occurrence Probability.
TPM	Tweets per minute.
TREC	Text REtrieval Conference.
UML	Unified Modeling Language.
URL	Uniform Resource Locator.
US	United States.

USA	United States of America.
VM	Virtual Machine.
VSM	Vector Space Model.
XML	Extensible Markup Language.