Using Text Segmentation to Improve Back-Of-The-Book Indexing

Bachelor-Thesis von Mateusz Parzonka
15. Dezember 2010

TECHNISCHE UNIVERSITÄT DARMSTADT

UBIQUITOUS KNOWLEDGE PROCESSING

Using Text Segmentation to Improve Back-Of-The-Book Indexing

vorgelegte Bachelor-Thesis von Mateusz Parzonka

Supervisor: Prof. Dr. Iryna Gurevych
Coordinators: Dr. Torsten Zesch, Nicolai Erbs

Tag der Einreichung:

Erklärung zur Bachelor-Thesis

Hiermit versichere ich, die vorliegende Bachelor-Thesis ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den December 15, 2010

(Mateusz Parzonka)

Abstract

Nicht-fiktionale Bücher enthalten üblicherweise ein Stichwortverzeichnis, da es den Nutzen und Wert des Buches steigert. Die Herstellung eines Stichwortverzeichnisses ist aufwändig und erfordert viel menschliche Arbeitskraft, so dass versucht wird, diese Aufgabe mittels Methoden der natürlichen Sprachverarbeitung zu automatisieren. Die automatische Erstellung von Stichwortverzeichnissen wird dabei als stark verwandt mit der Schlüsselwortextraktion angesehen. Viele Methoden der Schlüsselwortextraktion eignen sich jedoch nur bedingt für lange Texte, so dass in dieser Arbeit geprüft wird, ob durch eine Unterteilung des Textes in kleinere Segmente die Anwendung dieser Methoden verbessert werden kann. Dazu implementieren wir ein unüberwachtes Verfahren zur automatischen Erstellung von Stichwortverzeichnissen, welches Textsegmentierung verwendet. Die Effektivität des Systems wird mittels eines existierenden Gold-Standards, bestehend aus 29 Büchern und den zugehörigen, von Menschen erstellten, Stichwortverzeichnissen evaluiert. Dabei gelingt es mittels einer Kombination aus Named-Entity-Recognition, Tf.idf und einer einfachen Segmentierung, den bestehenden Referenzwert für diesen Gold-Standard zu übertreffen. Wir zeigen, dass die Segmentierung zu einer Verbesserung beiträgt, aber auch das Ergebnis verschlechtern kann. Zu den überraschenen Ergebnissen gehört unter anderem, dass wir mit den verwendeten Verfahren aus der Schlüsselwortextraktion nur vergleichsweise schlechte Ergebnisse erzielen konnten. Unsere Untersuchung wird begleitet von einem umfassenden Überblick über die existierenden Methoden der automatischen Erzeugung von Stichwortverzeichnissen. Der verwendete Gold-Standard wird in diesem Kontext ebenfalls einer kritischen Analyse unterzogen.

Abstract

Non-fictional books usually contain a back-of-the-book index, because it significantly increases the usability and contributes to its overall value. Since the creation of such an index requires many days of intensive human labor, it is tried to automatize the task using methods of Natural Language Processing. Automatic back-of-the-book indexing is thereby considered to be strongly related to methods rooted in keyphrase extraction. Many methods of keyphrase extraction are only conditionally suitable for long texts, so we analyze, if a partitioning of the text in smaller segments can improve these methods. For that purpose we implement an unsupervised system for automatic back-of-the-book indexing, which incorporates text segmentation. The effectivity of the system is evaluated using an existing gold standard consisting of 29 books with associated indexes created by humans. In the process we are able to exceed the existing benchmark for this gold standard, using a combination of named entity recognition, Tf.idf and simple segmentation. We demonstrate, that segmentation can improve but also worsen the results. It was one of the surprising findings, that by using a prominent method from keyphrase extraction, only considerably poor results could be achieved. Our investigation is accompanied by an exhaustive overview over existing methods of back-of-the-book indexing. The applied gold standard is thoroughly analyzed in that context.

Contents

1 Introduction

This chapter gives a short motivation and outlines the goals of this work.

1.1 Motivation

To give an insight how text segmentation can improve back-of-the-book indexing we would like to sketch the two domains:

- **Automatic Back-of-the-book indexing:** A back-of-the-book index is a list of words or phrases associated with pointers that refer to occurrences in the book. When well crafted, it offers a way to easily locate a given piece of information without having to skim manually through a great amount of text. Non-fictional books usually contain a back-of-the-book index, because it significantly increases the usability and contributes to the overall value of the book. The creation of such an index requires many days of intensive human labor, in which the book has to be read multiple times. Since indexing of books is expensive in both time and money, researchers tried to develop methods of automatic creation of back-of-the-book indexes over the years. Some progress had been made, but the task of automatic indexing remains difficult.
- **Text Segmentation:** Text Segmentation is a method to partition text into smaller units. Advanced segmentation algorithms aim to partition texts respecting the semantics of document, thus creating boundaries which divide the text into meaningful units or subtopics. Apart from its usefulness in various NLP-areas we will explore if segmentation is helpful in the domain of automatic back-of-the-book indexing. As we will see, the special task of index entry extraction can be seen as a special application of keyphrase extraction, which means the extraction of the most important phrases from the book. Books are usually very long, which can make the application of many extraction methods difficult. Text segmentation can divide books into shorter and meaningful units and thus enable certain extraction again and may improve working ones.

1.2 Goals

The main objective of this thesis is the analysis if and how text segmentation can improve automatic back-of-the-book indexing.

To answer this question, we will implement and evaluate a back-of-the-book indexing system incorporating text segmentation. This implies:

1. The design of methods for back-of-the-book indexing that allow comparison between approaches with and without segmentation.
2. The implementation of these methods using the UIMA-framework and existing components provided by the UKP-group.
3. The conduction and evaluation of experiments guided by hypotheses derived from the main objective.

This approach is explorative, meaning the conduction of experiments with different configurations. The performance of the indexing system is evaluated measuring the agreement with human created indexes as a gold standard. We will not create a own dataset but instead rely on the publicly available dataset provided by other Csomai and Michalcea [CM07].

The scope of this thesis further includes:

- An exhaustive outline of the current state-of-the-art in automatic back-of-the-book indexing including semi-automatic human-guided systems.
- An analysis of the dataset used as a gold standard for evaluation.

2 Back-of-the-Book Indexing

This chapter is divided into three sections. The first section outlines the problem domain, the second section sketches the current state of research in back-of-the-book indexing and the closing third section gives an outlook on some related fields of research.

2.1 Introduction

In this section we will introduce the problem domain. We will introduce basic concepts and terminology in the first two subsections. In the last subsection we will give an overview over the methods of human indexers and try to sensitize for the specific problems in automatic indexing.

2.1.1 The Back-of-the-Book Index

A definition of *back-of-the-book index* taken from a book for students of that topic [Mul05] states as follows:

> “A [back-of-the-book] index is a structured sequence – resulting from thorough and complete analysis of text – of synthesized access points to all information contained in the text.”

From this statement we can distill the following properties that must hold for a back-of-the-book index:

The index

1. is structured *sequentially*.
2. provides *access points* to locations in the text.
3. allows to access *all* information in the text.
4. has access points that are *synthesized*.

Any indexer, if human or machine, has to respect these properties when crafting a good index. This enables the reader to efficiently access all information in the book, which comprises concepts, items, names and places.

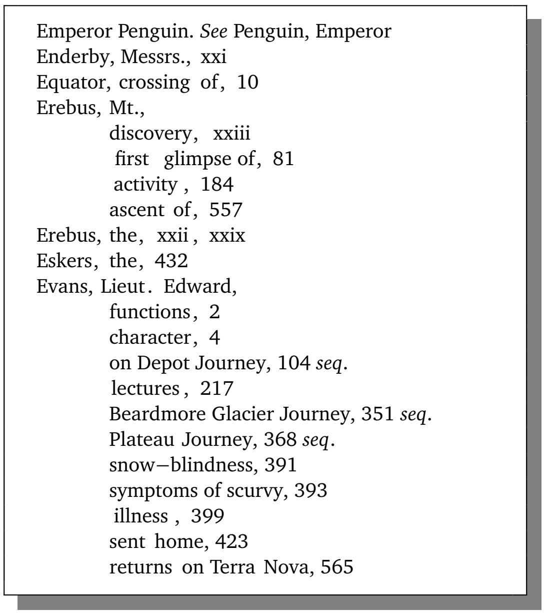
The above definition further bears hints about the workflow that is necessary to achieve the index: A systematic analysis of the text is needed, during which the index entries are *synthesized* from the text, meaning a process of subsumptive extraction of only relevant entries.

Without analysis and synthesization, we would not have a back-of-book index but a concordance, a long list of words that appear in the book. A concordance easily satisfies the first three properties, but, lacking synthesized entries, is impracticable to be printed in the back of the book, due to its huge number of often irrelevant entries.

Analysis and synthesization, as we will see, are complex operations, that are difficult to perform for humans but much more difficult for machines.

2.1.2 The Structure of the Index

A book index is roughly described as an alphabetical list of words with associated references contained in a hierarchical layout, often realized by indentation (Figure 2.1[1]).



Emperor Penguin. *See* Penguin, Emperor
Enderby, Messrs., xxi
Equator, crossing of, 10
Erebus, Mt.,
- discovery, xxiii
- first glimpse of, 81
- activity , 184
- ascent of, 557

Erebus, the, xxii , xxix
Eskers, the, 432
Evans, Lieut. Edward,
- functions, 2
- character, 4
- on Depot Journey, 104 *seq*.
- lectures, 217
- Beardmore Glacier Journey, 351 *seq*.
- Plateau Journey, 368 *seq*.
- snow—blindness, 391
- symptoms of scurvy, 393
- illness , 399
- sent home, 423
- returns on Terra Nova, 565

Figure 2.1: Excerpt from index in *indented style*

The exact mode of structuring the index and its entries, as well as the realization of the access points are defined by convention and can vary over time. Current indexes follow rules that were refined by experts during decades. A general structure and nomenclature that should be quite invariant in history is specified in the *Chicago Manual of Style* [G$^+$95] and in *Indexing Books* [Mul05]:

An index entry consists of a word or phrase called **heading** and a reference to a information location. This reference is either called **locator**, when pointing to a location in the text, or called **cross-reference**, when pointing to a location in the index itself. Locators in printed work are usually page numbers, but can also reference paragraphs, sections and the like.

The heading consists of a **main heading** which is usually a noun or a noun phrase and should never be an adjective. Noun phrases are often **inverted** to show the **keyword** that would be used for lookup appear first, so when indexing the phrase *Otto Graf von Bismarck* it is inverted to *Bismarck, Otto Graf von*.

An entry that would require more than five or six locators is suggested to be broken into **subentries**. A subentry, like the entry, consists of a heading, which is referred to as **subheading**, and a locator. Subheadings *can* form a grammatical relationship with the main heading, so that heading and subheading can be combined into a single phrase, as shown in figure 2.2.

According to professional indexer Mulvany [Mul05] the grammatical relationship can be abandoned in most cases for a more space saving, minimal *logical relationship* using subheadings without connecting prepositions. Users "know" the kind of relationship of the subheading to the main heading, so that it does not have to be stated more formally, when the relation is clear.

[1] CHERRY-GARRARD, A. *The worst journey in the world: Antarctic 1910-13.* Published 1922. This book is part of the dataset discussed in section 4.2. How styles change over time is apparent consulting Rule 16.13 in the Chicago Manual of Style [G$^+$95]: "The abbreviations *ff.* or *et seq.* should never be used in an index."

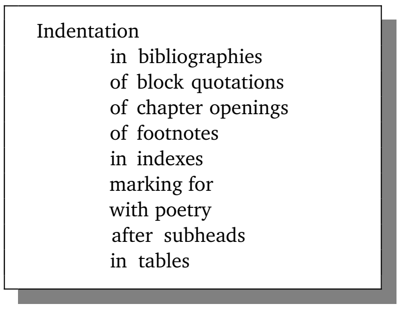

Indentation
- in bibliographies
- of block quotations
- of chapter openings
- of footnotes
- in indexes
- marking for
- with poetry
- after subheads
- in tables

Figure 2.2: Grammatical relationship formed by subheadings and main heading

Headings can be singular or plural, whereas countable nouns like *books*, *mice* and *politicians* should be in plural and uncountable nouns like *air*, *pepper*, *tea* in singular. There is no limit on the number of words in a heading or subheading. Entries should be as succinct as possible but sometimes the introduction of additional terms is necessary to increase clarity.

The index shown in figure 2.3 is in *indented style*. It is one of the two general formats for indexes, next to *run-in style*. The styles are distinguished by the way the subheadings are formatted. In indented style, each subheading begins with a new line where the amount of indentation reflects the level of heading: Main heading, subheading, sub-subheading, sub-sub-subheading and so on. This type of layout is also known as *hierarchical* or *line-by-line-style*.

In run-in format subheadings follow one another without line breaks in between, instead separated by semicolon. According to Mulvany, the only purpose of run-in style is to save space (Figure 2.3)

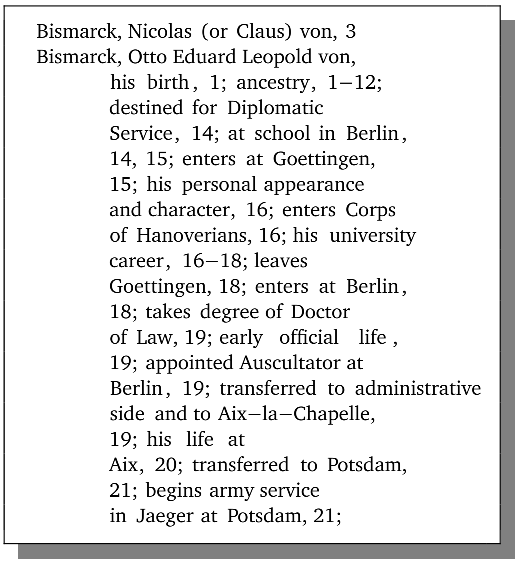

Bismarck, Nicolas (or Claus) von, 3
Bismarck, Otto Eduard Leopold von,
his birth, 1; ancestry, 1–12; destined for Diplomatic Service, 14; at school in Berlin, 14, 15; enters at Goettingen, 15; his personal appearance and character, 16; enters Corps of Hanoverians, 16; his university career, 16–18; leaves Goettingen, 18; enters at Berlin, 18; takes degree of Doctor of Law, 19; early official life, 19; appointed Auscultator at Berlin, 19; transferred to administrative side and to Aix–la–Chapelle, 19; his life at Aix, 20; transferred to Potsdam, 21; begins army service in Jaeger at Potsdam, 21;

Figure 2.3: Excerpt from index in *running style*

Specifications and guidelines exist for alphabetization and many other things like handling of names, abbreviations, and cross-references – none of them are necessary for the understanding and application of automatic indexing methods. We can safely declare further details as out of scope of this work.

2.1.3 The Challenge of Automatic Back-Of-The-Book Indexing

The former teacher of indexing Nancy Mulvany describes herself as being “disturbed” that only 10% of the students really seem learn the ability in her course. She demands a set of cognitive abilities for the job of indexing: Reading comprehension and classification ability are required, as well as conceptualization skills regarding particularly thematic relations[2].

How should an indexer decide what terms should be the basis for headings? How does he know what readers will look for and what structure they will expect? “There are no rules”, according to Mulvany, which is “perhaps one of the most frustrating issues for students.” The general advice is, to use “succinct and clear entries” in the index, especially when wording the main headings, and to choose the proper *depth* of indexing, depending on the material and the available space. This can mean the indexing of three to five entries per page for light, mass-market trade books, and up to ten entries and more for technical documentations like service manuals and the like.

How could a computer select terms? It is easy for him to generate a list which contains all words and phrases that appear in a text. At first glance, this list could be used as list of starting point for a list of headings. The problem is, that such a list would be very big, and would contain words and phrases that are not relevant for the text, and would normally not appear in an back-of-the-book index. So we would end up with a huge list of phrases, with the most being ungrammatical, meaningless or irrelevant for the book. Without being able to part relevant from the irrelevant, the unbounded extractive approach is fruitless.

For a high quality index, simple extraction is not enough anyway – a human indexer needs sometimes to *generate* headings that do not occur directly in the text. For several reasons: He may want to rephrase entries to a more common wording, because an author sometimes uses an outdated or rare vocabulary which can be out of place in an index for a modern audience. The second reason is, that some *implicit* concepts are discussed in longer passages of the text, without being mentioned directly, and it may be necessary to create a heading to allow the lookup of this concept. This task, which involves a great amount of text understanding, is often depending on the expectations of the target audience and on the topic of the book. An illustrative example for implicit concepts is given by the American Society of Indexing (ASI)[3]

> “[A] book on protective gloves for occupational use might have a chapter discussing surgical gloves, how they get punctured and how they are tested for integrity, but might never use the word ‘holes’. Yet a user of the book would expect to find this word in the index and be directed to the appropriate chapter. ”

According to the ASI, an indexer has to handle dozens or hundreds of such issues in every book. Common other examples include entries containing biographical information about people. The main heading is the persons name, with possible subheadings like *birth*, *early school years*, *move to england* etc., which have a higher abstraction from the written text applying semantic analysis.

It is worth noting that the task of heading generation is much more difficult that the task of heading extraction. As a consequence nearly all systems in the field apply purely extractive methods, including the system studied here. But heading generation is still an interesting task which may be addressed by systems in future that rely on technologies like text summarization and such.

At this point, a note about the second part of the index: the locators. The setting of locators to occurrences in the text may look trivial for a machine at first sight. But, assuming a relevant heading is

[2] The thematic relation expresses a categorized meaning of the noun phrase in the context of the other elements of the sentence, particularly the verb. These relations can be `agent`, `experiencer`, `location`, `theme`, `instrument`, etc.

[3] ASI is a nonprofit organization founded in 1968 to promote excellence in indexing and increase awareness of the value of well-written indexes. http://www.asindexing.org

found, the mapping to the right occurrences is not without ambiguity. The word *key* can be mapped to a sentence where it means a tool to unlock doors, as well as to a text fragment about the interface of a piano. This is usually not intended, and the indexer has to develop a index structure that handles words with multiple meanings. A second problem is the use of synonyms by the author, for he can use words like *application*, *software*, and *program* interchangingly, which should be referenced by one index entry. Simpler and more obvious problems involve inflectional variants, spelling variations and accidental misspelling.

So, facing these difficulties, what is a automatic back-of-the-book indexer capable to achieve? Since generative methods are to difficult to implement, which capture implicit concepts in the text, automatic indexing currently means finding “good” headings that occur – more or less – directly in the text. When the automatic indexer is capable of prioritizing headings by quality, he is able to retrieve a list of headings of reasonable length, that hopefully resembles an index constructed by humans.

2.2 Existing Methods

The task of Back-of-the-book indexing can be roughly characterized as the retrieval of the most important phrases from a book. *Automatic* back-of-the-book indexing tries to solve this task without involving any (or very little) human labor. Since fully automatic solutions are difficult, some researchers develop systems that are involve human interaction in some points of the index creation process – these systems are called *machine-aided* back-of-the-book indexing systems.

2.2.1 Automatic Back-Of-The-Book Indexing

Most of the existing methods for automatic back-of-the-book indexing can be generalized to a common setup: At first, the system has to extract a arbitrary large number of phrases that are suitable for being included in an back-of-the-book index. This set of index entry candidates is usually so large that it has to be reduced somehow, so that only those entries remain, which are the best index entry candidates according to a metric of relevance. The remaining index entry candidates after the filtering stage are the index entries that form the generated back-of-the-book index. In practice, these two steps are subdivided in multiple smaller units, especially the reduction of the candidate set is usually a multi-step processing technique.

[Bor70]

One of the earliest attempts to automatically generate book-indexes is made by Harold Borko [Bor70]. His method was developed and evaluated in the early seventies, running on a computer with approx. 3 Megabytes of RAM, storing intermediate results to tape. Although the resources where extremely limited compared to todays possibilities and the approach a quite naive, the general structure exemplifies the form described before of a candidate generation step and filtering afterwards.

All words that are not appearing in a list of stop-words are counted as index entry candidates. In an additional processing step, Borko further tries to combine words, that have the same stem. The list of candidates is further filtered with a list manually composed generalized non-index-entries (f.e. *alternatively*, *altogether*, *amount*, *ample*, etc.). The remaining words are counted as index entries.

The attempt fails because the remaining list, after filtering stop-words and other overly general words, still contains too many index entries of low quality. Borko later cuts back his system to a human assisted system, where a human indexer can choose, which terms to keep from the priorly extracted list.

FASIT

The *Fully Automatic Syntactically-based Indexing of Text* (FASIT) system developed by Dillon [DL83] features a candidate extraction stage that tags text tokens with part-of-speech tags. In the following step, candidate phrases are selected using predefined part-of-speech patterns as shown in the table below:

Phrase	POS-pattern
library filing rules	NN VGN NNS
conversion rules	NN NNS-VBZ
sort rules	NN-VB NNS

Dillon then applies a reduction filter that merges phrases into equivalence classes: Using a measure of association based on properties of stems contained in the phrases, like frequency and position in the phrase, phrases are merged to a class where all members are indexed by one representing term.

With few exceptions, for any two phrases, the one with the fewer stems is declared as index entry, while comparing terms with the same length, the term with the most significant stem on the left is considered as superior. The back-of-the-book index subsumes all index entries found by this method and additional proper names, which are detected using a simple named-entity-recognition heuristic.

Dillon evaluates his system using a single book "technical in nature" but with "unspecialized vocabulary" and reports on recall of 86.75 and a precision of 66.26 comparing the FASIT index with the human index[4].

[Sal88]

Saltons [Sal88] does not report on evaluation results, his publication describes a system yet to be implemented.

The system consists of an automatic phrase construction system that generates a large number of two-term noun phrases from the input text. The generation is based on syntactic parse trees, in contrast to Dillon [DL83], who uses lookups in exception word dictionaries and (after stemming the phrase words) in suffix dictionaries. The noun phrases are filtered or privileged using post-processing rules: Phrases that are morpho-syntactic variations of another phrase are removed, and phrases consisting of three or more words and noun-noun constructions are privileged, while adjective-noun constructions are attenuated. The candidate phrases are finally ranked by Tf.idf [SB88], so that the n phrases with the highest score can be declared as index entries.

[CM07]

After introducing a testbed for evaluation of automatically generated back-of-the-book indexes [CM06] Csomai and Michalcea propose their own unsupervised indexing system [CM07]. The system is evaluated using a subset of dataset introduced in [CM06] (see also section 4.2).

- **Candidate Extraction:** The researchers explore four different candidate sets: N-grams, Noun-phrase chunks, named entities created by a state-of-the-art classifier, and named entities generated by a simple heuristic.
- **Filtering:** Csomai and Michalcea apply stopword and common word filtering to minimize the candidate sets and increase the quality of the extracted phrases.
- **Ranking:** The ranking tries to maximize the *phraseness* and the *informativeness* of a candidate phrase: Phraseness represents the degree, to which the candidate can be considered as a grammatically valid phrase, while informativeness refers the degree to which the phrase is representative for the document.

  To measure phraseness, Csomai and Michalcea apply methods used in collocation extraction [PS06] like the *pointwise Kullback-Leibler divergence* measure and the χ^2*-independence* test. These methods rank phrases higher, which are composed of components that have a lower probability of appearing combined with other terms, than with the given ones. Using the KL divergence, the probability is

[4] Interestingly, Dillon sees human indexes as "far from perfect in many ways" and disagrees with the opinion that human indexes are the best means of evaluation, because "objective evaluations of human indexing repeatedly demonstrate the numerous weaknesses and we assume that an evaluation of any book index would have similar results".

based on a background corpus, whereas the χ^2-independence is calculated using just the given document.

The traditional Tf.idf metric [SB88] is also applied to measure informativeness, using the British National Corpus[5] as document collection.

Some of these methods are applied as well to measure informativeness, to rank phrases higher that appear with a higher probability as anticipated for an average phrase.

- **Paraphrase Recognition:** This step implements a simple paraphrase recognition. To increase the precision of the candidate sets, morpho-syntactic variations and lexical synonyms of candidate phrases are addressed using "extended token sets". Each word in the phrase is replaced by a set of "extensions", which contains the stem of this word and synonyms defined in WordNet [Mil95]. Two candidates are recognized as paraphrases, when there exists a bijective mapping between their associated token sets.

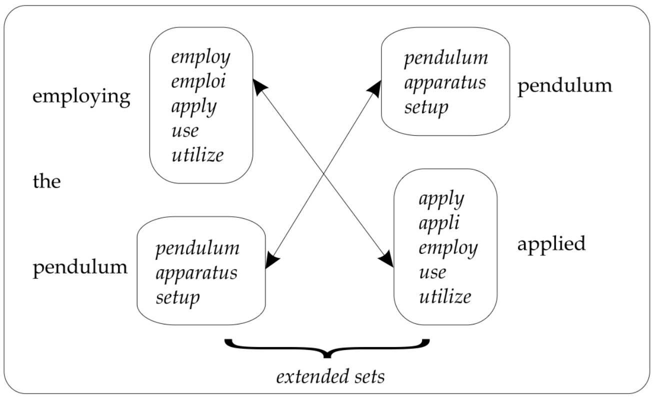


Figure 2.4: Example for paraphrase recognition using extended token sets [CM07]

As a conclusion, the best result with 26.7 precision and 26.4 recall was obtained using a combined n-gram and named entity candidate extraction method applying stopword and common word filtering, and coupled with a ranking scheme using χ^2-informativeness and χ^2-phraseness followed by paraphrase recognition.

[CM08]

Csomai and Michalcea extend their approach with a supervised method, implying the usage of three machine-learning algorithms, which were selected for their diversity: multilayer perceptron, support vector machine and decision tree[6].

- **Candidate Extraction:** The candidate set used here is the set of n-grams (with n in $\{1, ..., 4\}$) not crossing sentence boundaries.
- **Filtering:** The first step reduces the size of the data set by eliminating all n-grams that begin or end with a common word, as well as those n-grams that cross a comma.
- **Training-Set:** The set of positive examples is the subset of index entry candidates that match entries from human compiled index, while the set of negative examples is the subset that does not match this gold standard. Since the negative set is much bigger, Csomai and Michalcea apply

[5] http://www.natcorp.ox.ac.uk The BNC is also used as background corpus with the KL divergence.
[6] The implementations provided in WEKA where used: http://www.cs.waikato.ac.nz/ml/weka/

undersampling, meaning the usage of only 10% of the negative examples to retrieve a balanced set of examples.

- **Features:** Csomai and Michalcea reuse the metrics for phraseness and informativeness as features, and introduce a set of the features which are shown in table 2.1 including their information gain weights.

Feature	Weight
Part-of-speech pattern	0.1935
Discourse Integration CI shortterm	0.1935
Wikipedia keyphraseness	0.1731
Discourse Integration CI maxscore	0.1689
Discourse Integration CI shortterm normalized	0.1379
χ^2-informativeness	0.1379
Document frequency (df)	0.1031
Tf.idf	0.0870
χ^2-phraseness	0.0660
Length of phrase	0.0416
Named entity heuristic	0.0279
Within document frequency	0.0227
Term frequency (tf)	0.0209

Table 2.1: Features and information gain weights in the supervised back-of-the-book indexing system ([CM08])

The strongest feature is based on a metric, which maps the POS-pattern of candidate to a value, representing the probability that a candidate with this POS-pattern is an index entry. Other strong features are derived from an implementation of construction integration used to model discourse comprehension [Kin98]. Through construction integration (CI), Csomai and Michalcea can keep a phrase in a short term memory, even if a phrase appears only once, the CI process ensures the presence of the phrase in this memorym as long as it is relevant to the current topic.

Using the Wikipedia allows the integration of another phraseness metric based on community generated semantic knowledge – a candidate gets a higher score when it is contained in headlines or in link anchors.

After training using 259 books, the system is tested with 30 books. The multilayer perceptron performs best with a F-measure of 27.87, when extracting a averaged amount of index terms from every book, and 27.38 when deciding by itself how many entries to extract. When applying the latter, decision-based extraction threshold, the decision tree algorithm reaches a recall value of 34.12 with a precision of 22.75, which results in an averaged F-measure of 27.30. We think that in back-of-the-book indexing, low precision can be accepted for a higher recall, since it is easier to post-filter extracted indexes by humans, than to find the candidates themselves in the first place. With this in mind, this supervised system has the highest known efficiency to date.

2.2.2 Machine-aided Back-of-the-Book Indexing

Machine-aided indexing systems involve human interaction at some point of the process. This can be either the preparation of special information specific to the indexed book, or a GUI-based, interactive workflow which aids the human indexer by his work.

[Dil82]

Following an idea of Borko [Bor70], who depended on a big list of candidates for the selection of good index headings, Dillon [Dil82] introduces a related method: When books from a certain domain have to be indexed, a thesaurus is build prior by a human. This thesaurus contains the most important terms from the domain, with the intuition that this terms should always be indexed in every book in this domain. When the book contains phrases from the thesaurus, the phrases are labeled as index entries. This technique is likely to be most successful in domains with highly specific vocabulary, where books have to be indexed exhaustively.

IndDoc

IndDoc is a GUI-based system introduced by Zargayouna and El Mekki [ZEMAN06] configured for the indexing of books in the french language. It uses the term extractor YaTeA [AH06], provided with a list of *seed* terms that has to composed by the user.

- **Candidate Extraction:** The list of candidates retrieved from YaTeA includes all terms and proper names which are are considered significant by some metric.
- **Clustering:** IndDoc merges morpho-syntactic variations of phrases (e.g. *activité de coopération*, *activité coopérative*, *activités coopératives*) to one phrase.
- **Ranking:** The candidates are ranked using different methods like term frequency (in the document or in parts of it), and typographical or linguistic emphasis.
- **Subheading-Recognition:** This processing step allows the extraction of subheadings using hyponym recognition and exploiting structural patterns. The multi-word phrase *activité collective* is regarded as a more specific version of activité. From the sentence "*The mouse, keyboard and other computer accessories*" IndDoc derives the following hierarchical index entry:

```
accessory
        computer accessory
                keyboard
                mouse
```

- **Choosing locators:** Considerable attention is paid to the relation between the index entry and the text segments it refers to [AN06]. Adopting the perception of professional indexers, an index entry refers not to a page in the book but to a *span* of text. El Mekki *et al.* applies a text segmentation algorithm exploiting the structuring of the text, some keywords that suggest continuation of topics in following paragraphs, and lexical cohesion based on the recurrence of candidate phrases and their variants and synonyms.

[RLJ10]

This system was developed by Lukon [Luk06] supervised by Juola [Juo05] and later implemented as a stand-alone system together with Reinholt [RLJ10]. It incorporates these steps:

- **Tagging:** A part-of-speech tagger is used to locate all nouns in the document. The user has the possibility to supervise the proper tagging.

- **Filtering by frequency thresholds:** The user has to determine the minimum and maximum threshold for the number of occurrences of a noun phrase in the text. Nouns lying in between these boundaries are classified as candidate entries. At this step it is possible for the user to merge similar words into one.
- **Finding similar candidates:** Reinholt, Lukon and Juola employ a method called *Latent Semantic Analysis* (LSA) introduced by Deerwester *et al*. [DDF+90], which allows to identify terms that are similar in meaning and/or context, without using external semantic knowledge.

  LSA takes a matrix as input, which represents the covariance of every index candidate with every other of the n index candidates in the document. The varying value is the Tf.idf score using the paragraphs of the book as the set of documents, thus the covariance of two terms is the amount by which the Tf.idf score varies together in all paragraphs.

  The resulting $n \times n$ matrix is transformed through *singular value decomposition* to a smaller matrix, which represents a k dimensional *semantic space*, where dimensions are sorted by importance. A point in the semantic space represents a term, so that similar values in important dimensions can be interpreted as "near", thus implicating semantic relatedness or synonymy between terms. Figure 2.5 shows a plot, representing six terms in a semantic space, spanned by its two most important dimensions.

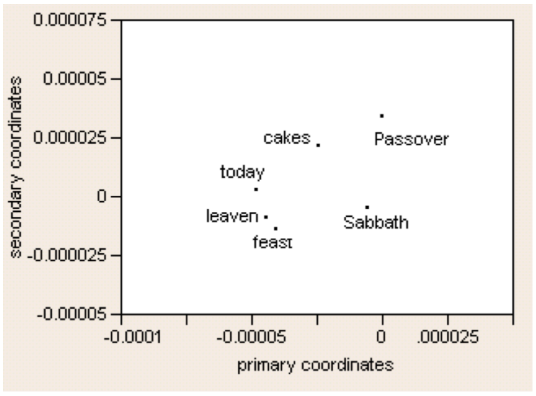


Figure 2.5: Two-dimensional semantic space created by Latent Semantic Analysis [Luk06]

- **Clustering similar candidates:** The result of the previous step was a semantic space of index entry candidates. Euclidian distance between points in the space can now be used to model semantic relatedness and cluster similar candidates. For the clustering, a *centroid distance measure* is used to compare candidates with an existing cluster. This measure calculates the midpoint of each cluster, so that all candidates within a specified threshold to a cluster are incorporated by it. The clustering-by-distance threshold can be visualized as depicted in figure 2.6.

  The threshold can be set by a user, who sees the effects of the new setting of the screen.
- **Word Sense Disambiguation:** In this stage, the system will check for any index entries that may be spelled the same, but have a different meaning depending on the context. The user can specify a certain distance threshold, and split apart those candidates whose surrounding text has another average context. Another iteration of clustering similar candidates will follow, which will possibly generate different clusters.

  The user is be able to see the differences and the contexts in which the entries appear.

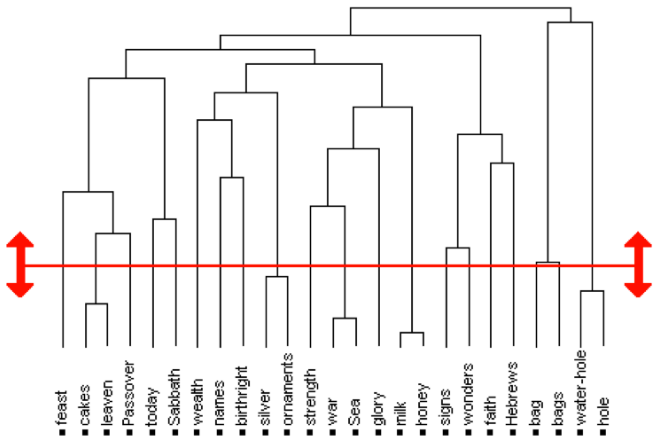


Figure 2.6: Clustering of candidates with variable distance threshold [Luk06]

Commercial Programs

No commercial program is known that is able to automatically create an index of such quality, that it could be used by publishers as back-of-the-book index in a printed book. Some tools appeared the market, but had to leave it fast because of harsh criticism of the indexing community about the indexing results[7]. One semi-automatic program exists, advertised as being used by professional indexers, is called TExtract[8]. It automatically creates an initial index, consisting of compound index terms (like *pre-merger notification requirement*) and prepositional phrases (like *preliminary injunction for predatory pricing*) and offers multiple post-filtering and -processing tools.

Most commercial programs have no automatic generation but simply aid the human indexer by his work: CINDEX[9], MACREX [10] and SKY Index Professional[11] are popular software suits that allow the collection of entries, formatting, alphabetization, creation of subheadings and managing references[12].

2.3 Related Fields of Research

In this section we will outline a few related fields of research. In principle, all fields can be considered as related to back-of-the-book indexing, which aim on the extraction of short pieces of text from a larger text body sufficing specific criteria. In practice this can mean finding

- the most central terms for a document (keyphrase extraction)
- terms or phrases that summarize the documents content best (text summarization)
- terms that are specific for a specific domain (terminology / glossary extraction)

Next to these structurally similar technologies we can also consider *enabling technologies* as related in a broader sense. Looking at the different back-of-the-book indexing systems presented in section 2.2, it becomes apparent how many technologies can be actually incorporated into a back-of-the-book indexing

[7] http://www.bayside-indexing.com/idxcon.htm
[8] http://www.texyz.com/textract
[9] http://www.indexres.com
[10] http://www.macrex.com
[11] http://www.sky-software.com
[12] http://www.asindexing.org/i4a/pages/index.cfm?pageID=3319

system. We have seen collocation discovery, paraphrase recognition, named entity recognition, word sense disambiguation, semantic relatedness measures, discourse comprehension etc.

At this point we would like to narrow the focus a bit and present two methods, which seem to be related to back-of-the-book indexing in a nearer sense.

Since back-of-the-book indexing may be considered as a kind of “keyphrase extraction from long documents” we will discuss a system for unsupervised keyphrase extraction called *TextRank* in the next subsection. This system will be used as a component in our system as well.

in the following subsection we will discuss a system is from the field of glossary extraction. We consider it strongly related, since it works on large document collections targeting on the extraction of domain-specific terms.

TextRank

TextRank [MT04] by Mihalcea and Tarau is a system to extract multi-word terms that represent document semantics. It applies a graph-based approach to text. Selected terms in a document are declared as keyphrase candidates and represented as vertices in a graph. Each vertex is associated with a candidate score set initially to 1. During application of the algorithm, which also connects vertices with edges, a graph based ranking is applied that re-adjusts the scores in the graph. The candidates with the highest score are kept to serve as keyphrases.

A more detailed sketch of the algorithm, using the settings that provided the best result in the evaluation:

- **Vertex Identification:** Selects the terms in the document which shall be used in the graph. Due to the lexical structure of keyphrases, its disadvantageous to use all lexical units in the document as vertices. As pointed out by Zesch [ZG09], this decision depends on the corpus. Nouns and adjectives as tokens can be a optimal with one corpus, but their lemmatizations can perform better used with a different corpus. When applying TextRank to a corpus with long documents noun phrases yield the best results.
- **Connection by Co-Occurrence:** When two terms represented as vertices v_i and v_j occur together in a window of n corresponding lexical units (i.e. other nouns and adjectives in the surroundings of the term) then an edge between v_i and v_j is added to the graph. Mihalcea and Tarau use unweighted graphs to yield the best results, but its also possible to use weighted edges, where the weight of the edge correlates with the number of co-occurrences of the terms associated with the vertices. A rather small window with $n = 2$ has been proven superior over greater windows in this setting.
- **Ranking by PageRank:** PageRank [BP98] is a graph based measure for analysis of the link structure in the web that estimates the relative importance of web sites. Its creators Brin and Page applied the model of a “random surfer”, that travels through the web, following links from one site to another. Each link can be seen as a “recommendation” or a “voting” to the connected site. Stronger connected sites receive more votes which lead to a higher rank, while sites with a higher rank have a stronger effect when voting, compared to sites with a lower rank. An iteration in PageRank updates all ranks in the entire graph. When PageRank is applied to a graph for a sufficient number of iterations, then the ranks of the sites converge to values that represent their relative importance in the voting network.

  When transferred to the keyphrase-candidate graph, these values are associated with vertices that represent keyphrases. After reaching convergence the rank of every keyphrase reflects the importance relative to other keyphrases.

- **Post-Processing:** The keyphrases can now be sorted descending by rank and the highest ranked keyphrases extracted. In the set of the top keyphrase candidates, adjacent candidates are merged into multi-word keyphrases.

Glossary Extraction

GlossEx [PBB02] is a glossary extraction tool created by Y. Park, R.J. Byrd, and B.K. Boguraev. It is based on Textract, a text analysis system developed by the Text Analysis and Language Engineering project at IBM Research [NBB03]. It has been used with success to build glossaries for applications in the automotive-engineering sector, where it was deployed combined with a glossary administration tool to process huge amounts of textual data.

GlossEx is structured as a pipeline which works on a data collection consisting of documents. Basically it consists of three modules:

- **Candidate Extraction:** Only noun phrases and non-auxiliary verbs in their base form are considered as glossary item candidates. The structure of the noun phrase is based on Justeson and Katz's study [JK95]. Candidates are extracted using finite-state transducing trying to match a candidate form. A cascade of finite state transducers is used that incorporates part-of-speech tagging, named-entity recognition (to drop names and places from the candidate set), and a URL filter.
- **Pre-Modifier Filtering:** A pre-modifier is a optional phrase element that provides more definitional meaning to the following modified component, i.e. "*dusty* roads". This step removes pre-modifiers from candidates that do not provide domain specific information. I.e.: the modifier "*other*" is removed from "*other* qualified service technician" but "*dusty* roads" and "*ambient and wide open* throttle" remain unaffected because their pre-modifiers are considered to belong to the automotive domain.
- **Variant Aggregation:** A concept may be represented in the text in different variants. These can be symbolic variants (compositions of words that have different separators, i.e. *audio/visual input – audio-visual input*, *electro-magnetic clutch – electromagnetic clutch*), inflectional variants (*rewinds*, *rewinding*, *rewound*), misspelling variants or abbreviations. All variants that are conceptually identical are identified and aggregated into one glossary item. One of the underlying expressions is selected as the canonical form that is considered as the remaining glossary item candidate.
- **Candidate Ranking:** The candidate glossary items are ranked using the weighted sum of the item's domain-specificity and the degree of term cohesion of all words in the item's canonical form. The degree of domain-specificity is based on the intuition that when an item is used more often in a domain-specific document than in other collections, it more likely a domain-specific term. Since supposedly many methods for evaluating term association are applicable only to a limited degree to terms with more then two words, Park et al. propose a new measure for computing the cohesion of multi-word terms. This measure gives higher values to terms having high co-occurrence frequencies, for details see [PBB02].

3 Improving Back-of-the-Book-Indexing Using Text Segmentation

In this chapter we will present an unsupervised back-of-the-book indexing system involving text segmentation. The first section describes its structure and general architecture while the second section focuses on the components which it will utilize.

3.1 Overview

The first subsection sketches the structure of the system. In the second section we will apply a formalization of the system architecture and flow of results between the components.

3.1.1 The System in a Nutshell

As seen in section 2.2, many different strategies for automatic back-of-the-book indexing exist. We will implement a version featuring candidate extraction, text segmentation, ranking, aggregation and extraction by threshold as visualized in figure 3.1:

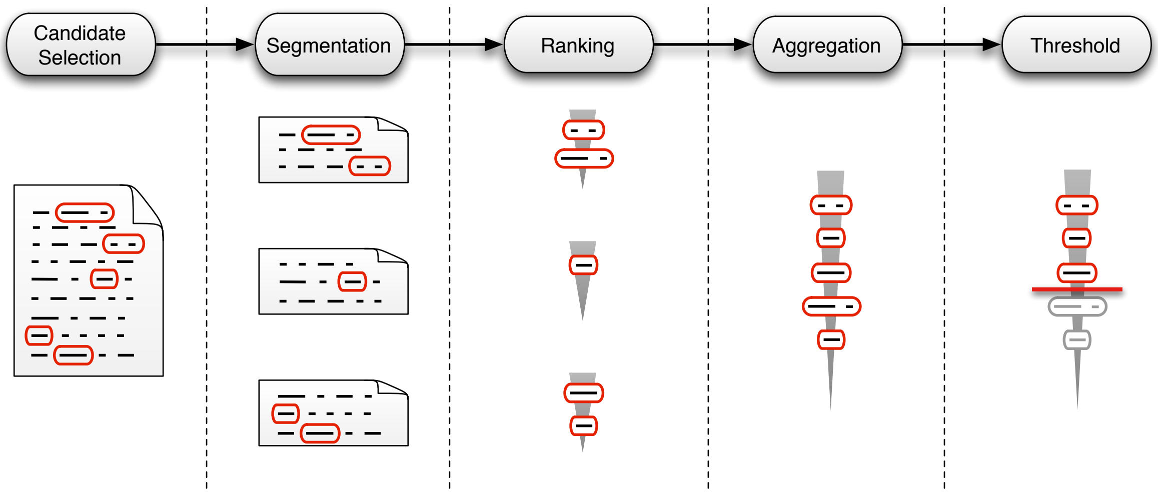


Figure 3.1: System overview

- **Candidate Extraction:** Some phrases in a document are regarded as more capable of being good index entry headings then others. This phrases are referred to as *candidate phrases*, *candidate entries* or simply as *candidates*. In the stage of candidate extraction, the candidate phrases are found and made accessible for the following steps.
- **Text Segmentation:** The main motivation of segmentation is to explore possible effects of candidate ranking in shorter documents. Thus we use different segmentation algorithms and parametrizations to divide the document in multiple parts, which are passed segment-wise to the ranking stage.
- **Ranking:** In this stage, the candidates in the segments are associated with a numerical score provided by a ranking metric. The score indicates the importance of the candidate phrase in the segment.

- **Aggregation:** Because of segmentation we get multiple segments with various ranked candidate phrases. These phrases have to be merged somehow into one hierarchy, so that its possible to extract a number of phrases with the highest ranks as the final set of index entries. Multiple solutions for the aggregation of ranked candidates are available in this stage, which will be evaluated later.
- **Threshold:** In the final stage, a threshold specifies the number of phrases that is to be extracted from the aggregate. These phrases comprise the set of index entries, which is the result of the back-of-the-book indexing system.

3.1.2 Formalization of the System

We can formally define our back-of-the-book indexing system B as a tuple:

$$\mathtt{B} := (\mathtt{C}, \mathtt{S}, \mathtt{R}, \mathtt{A}, \mathtt{T})$$

where C is a *candidate class*, S is a *text segmenter*, R is a *candidate ranker*, A is a *aggregation method* and T is a *threshold*.

The back-of-the-book indexing works on a document collection D consisting of documents d. A document is informally described as a body of text. We will model it as a sequence of *tokens*, where a token is understood as a sequence of characters:

$$d := (t_1, ..., t_{|d|})$$

The stages in the system can be formalized as follows:

- **Candidate Set:** A candidate set is the result of mapping a sequence of tokens to a set a candidate phrases C depending on a candidate class C. Candidates can be created for the whole document:

  $$candidates(\mathtt{C}, d) := C_d = \{c_1, ..., c_{|C_d|}\}$$

  Or for a token sequence:

  $$candidates(\mathtt{C}, (t_\alpha, ..., t_\omega)) := C = \{c_{\dot{\alpha}}, ..., c_{\dot{\omega}}\}$$

  The mapping models the creation of candidates like nouns or named entities from a body of text. Note that obviously $|C|$ is usually much smaller then $|(t_\alpha, ..., t_\omega)|$, which means we have we have fewer candidates then tokens.
- **Segmentation:** The segments of a document are values of the mapping of the document token sequence to a set of disjunct token sequences using segmenter S:

  $$segments(\mathtt{S}, d) := S_d = \{s_1, ..., s_{|S_d|}\} = \bigcup_{i=1}^{|S_d|} \{(t_{\alpha_i}, ..., t_{\omega_i})\}$$

  where $\alpha_1 = 1$, $\omega_{|S_d|} = |d|$ and $\alpha_i = \omega_{i-1} + 1$. These constraints assure that the segment do not overlap and consist of consecutive tokens in the document. It is not required that the segments have the same length, since $\omega_i - \alpha_i$ needs not to be equal to $\omega_j - \alpha_j$.
- **Ranking:** To associate candidates with scores, we will use a ranker R:

  $$rankedcandidates(\mathtt{R}, C_i, s_i, d, D) = R := \bigcup_{c \in C_i} \{(c, score(\mathtt{R}, c, s_i, d, D))\}$$

with $score(\mathtt{R}, c, s_i, d, D) \in \mathbb{R}$. The ranked candidates are modeled as a set of tuples consisting of a candidate and a score. The mapping to the score is depending on the given ranker and *may* depend on the given segment, document and document collection. We will see rankers that will need different amounts of context-information.

At this point, we have defined all elements needed to construct a back-of-the-book indexer up to the *ranking*-stage as sketched in section 3.1.1. It yields the set of all ranked candidate sets:

$$\hat{R} = \bigcup_{s_i \in segments(\mathtt{S},d)} \{rankedcandidates(\mathtt{R}, candidates(\mathtt{C}, s_i), s_i, d, D)\}$$

This is a set of sets, where each set is associated to a segment and contains its candidates coupled to a score.

- **Aggregation:** In the next stage, an aggregator $\mathtt{A}$ consumes the set $\hat{R}$ and flattens it to a set of re-ranked candidate phrases:

  $$aggregatedcandidates(\mathtt{A}, \hat{R}) = A := \{(c_1, \dot{r}_1), ..., (c_n, \dot{r}_n)\}$$

  with $\dot{r} \in \mathbb{R}$ and n the number of unique candidates in this document. Note that $\dot{r}$ *may* be a *new* score calculated on the basis of the old score r. The new score expresses the semantics of the given aggregator $\mathtt{A}$. The function *aggregatedcandidates* is also filtering candidates, that have the same character sequence, so there are no duplicate candidates allowed.

- **Threshold:** To retrieve the final set of index entries, a number of candidates with the highest score is extracted. The given threshold $\mathtt{T}$ determines the number of candidates with the function $entrycount(\mathtt{T}, d, D) \in \mathbb{N}$.

  The final set of index entries is gathered from the set of aggregated candidates:

  $$indexentries(\mathtt{T}, A) = E := \{\ sel_1(a) \mid a \in A \ \wedge \ sel_2(a) > r\ \} = \{e_1, ..., e_{|E|}\}$$

  where $sel_i((x_1, ..., x_i, ..., x_n)) = x_i$ and $r \in \mathbb{R}$ is chosen such, that $|E| = entrycount(\mathtt{T}, d, D) \in \mathbb{N}$.

  This models the sorting of the candidates by score and the extraction of a fixed amount of candidates with the highest score which is specified by the threshold $\mathtt{T}$.

3.2 Used Components and Methods

In this section we will introduce all components and methods which will be used at specific points of the system.

3.2.1 Candidate Sets

Candidate sets are sets of phrases that are created using a candidate class and a sequence of tokens, i.e. the document:

$$candidates(\mathtt{C}, d) = C_d = \{c_1, ..., c_{|C_d|}\}$$

Or a segment:

$$candidates(\mathtt{C}, s_i) = C_i$$

A single candidate is based and created from a sequence of tokens:

$$candidate(\mathtt{C}, (t_{\alpha_i}, ..., t_{\omega_i})) = c_i$$

where the creation can be divided in two different approaches: *Candidate extraction*, which will be the method applied in this work, and *candidate generation*.

Extraction can be roughly described as taking a slice from the examined text and declaring it as index entry candidate, or – speaking in token sequences – as a token sub-sequence. Note, that the tokens in the sequence are not necessarily unaltered, sometimes it may be useful to transform some of those tokens by basic natural processing methods like lemmatizing, change of case and such. These transformations are more moderate then in candidate generation, which describes the process of creating a candidate that is based on tokens from the document, but is a completely *new* phrase, which can't be found in the text at all. Targeting to capture the semantics of a text fragment, generated candidate phrases can introduce new tokens, while moving existing tokens to new positions etc.

An example for a generated candidate phrase: The text fragment "*Otto Eduard Leopold Von Bismarck was born at the manor-house of Schoenhausen* [...]" is indexed by a human indexer using the generated heading: *Bismarck, his birth*.

There is no system known, which is based on candidate generation, as it is yet difficult to realize. We will limit ourselves to candidate extraction as described, and will discuss on four candidate classes: n-grams, POS-filtered tokens, noun-phrase chunks and named entities.

N-Grams

For a given token sequence, the n-gram is a subsequence with length n, formed from consecutive tokens. It is a generalization of the sequence types unigram (a single token), bigram (two tokens), trigram (three tokens) and commonly used in modeling statistical properties of words in a text [JM08]. Often all n-grams for a fixed n of a given sequence are interesting, f.e. all 2-grams from the token sequence $(t_1, ..., t_m)$ that form the set $\{(t_1, t_2), (t_2, t_3), ..., (t_{m-1}, t_m)\}$.

Applying the extractive approach, using all n-grams from a document would provide the most exhaustive set of candidate phrases. Assuming that index entry headings are derived from sequences of text tokens, then these sequences are fully captured by n-grams. Consequently, an candidate set of n-grams with $n \in \{1, ..., k\}$ with a very large k should match with 100% of the index phrases that are present in the text.

The benefit of a n-gram candidate set, its exhaustiveness, is a big drawback at the same time. The set becomes so large, that it is difficult to handle in later stages of the back-of-the-book indexing system. It is helpful to exploit a few properties to reduce the size of this set:

- An entry crossing the sentence border will most likely be a nonsensical entry. So n-grams should not cross sentence borders.
- Based on the studies in [CM06], very long phrases are rather rare index entries. So the upper bound k of the n-grams with $n \in \{1, ..., k\}$ is set to 4.
- Depending on the used tokenizer, punctuation is included in the token sequence leading to the creation of many nonsensical n-grams composited partly of punctuation. As a conclusion, tokens that are not alphanumerics are excluded. We experimented with various filtering settings choosing the setting which yielded the best result in F-measure (see table 4.5, see also section 4.1.1).

It should be noted that tokenization and the algorithm to find sentence borders affect the quality of this candidate set.

POS-filtered tokens

Each token in a text can be mapped to a part-of-speech tag (POS-tag). There are tokens associated with certain POS-tags, which are more likely to be index entries then others. Main headings, for instance, are often nouns.

Therefore we compose a set of POS-filtered tokens, that is a set of all tokens that appear in the text being tagged with one of the filtering tags at least one time. The configuration of the filter is based on the observation, that index entries which are present in the document, are often nouns or nouns modified with an adjective: Therefore, the POS-filter includes nouns, adjectives or both. The set may also contain merged phrases which are composed of consecutive nouns and adjectives.

We will evaluate several settings in chapter 4 (or see table 4.6).

Noun-Phrase Chunks

Chunking allows the classification of subsequences of words in a sentence as *constituents*. Constituents are groups of words that fulfill functions in the sentence like noun phrases, verb phrases, adjective phrases, etc [JM08].

Based on the observation that index entries are often groups of words surrounding a noun, we can use a chunker to extract such groups as noun phrases. To increase the quality of the phrases as index entry headings, the trimming of stop words from the beginning and the end is applied.

Named Entities

A named entity is anything that can be referred to with a proper name. The process of finding named entities is called *named entity recognition* and plays an important rule in extracting information from texts in general [JM08]. In most books, named entities like people, places, animal species and such are often included in the index [Mul05], so the named entities in a text should constitute a candidate set of high quality.

We will use two algorithms for recognizing named entities, a simple and a complex one:

- **Heuristic Named Entity Recognition:** Proper names in the english language have common properties that can be exploited for recognition using some simple heuristics: A token sequence $(t_1, ..., t_n)$ not crossing sentence boundaries is classified as a named entity, if:
  - all t_i are alphanumeric.
  - all t_i are spelled in title case.
  - $n = 1$, then t_1 is not at the beginning of the sentence. This prevents the inclusion of named entities from the beginning of the sentence, since they are are always in title case.
- **Stanford CRF Named Entity Recognizer:** We also employ a state-of-the-art named entity recognition algorithm provided by the University Stanford to extract a candidate set consisting of high quality named entities [FGM05]. The algorithm is based on the modeling of token sequences as *Markov chains* with *hidden* states (Hidden Markov Model, *HMM* [BMSW97]), where the algorithm tries to find the most probable tagging of tokens in the sequence (*person*, *place*, *organization*, *no-named-entity*, and such) without knowing the original tags. This model is further improved by using *conditional random fields* (CRF) instead of the HMM: Whereas HMM are limited to *seeing* only the last token, the CRF model can exploit the whole input token sequence.

3.2.2 Segmentation

Text segmentation in this work is defined as the partitioning of a given token sequence into disjunct subsequences:

$$segments(\mathbb{S}, d) := S_d = \{s_1, ..., s_{|S_d|}\} = \bigcup_{i=1}^{|S_d|} \{(t_{\alpha_i}, ..., t_{\omega_i})\}$$

where $\alpha_1 = 1$, $\omega_{|S_d|} = |d|$ and $\alpha_i = \omega_{i-1} + 1$.

A stronger definition is connected with *discourse segmentation* [Hea94], which demands that the segments reflect the semantics of the document. Therefore, discourse segmentation algorithms try to divide the document into a *coherent* segments, i.e. segments, that are consistent in meaning.

We will apply three very simple text segmentation algorithms to the documents, targeting to estimate the impact on the effectiveness of the back-of-the-book indexing system. The fourth algorithm is a well known discourse segmentation algorithm named TextTiling [Hea97].

Segmentation per Sentence

This work funds on the thesis that ranking and extraction approaches work better with smaller segments. Consequently we can derive the assumption, that there exist a setting for creating small- to medium-sized segments, which leads to optimal results.

As a simplistic segmentation exploring this assumption we will partition the documents into segments consisting of n sentences, neglecting that the optimal segmentation is probably not equidistant in terms of sentence numbers.

We hypothesize, that there is a n in the range of $\{5, 30\}$, that should lead to the best results with other settings fixed, with decreasing effectiveness when n is too small or too big.

Segmentation per Paragraph

Analog to segmentation per sentence, this segmenter divides the document into segments consisting of paragraphs. The size of the paragraphs is derived from the document structure.

We assume that a division into paragraphs following the natural structure of the book, should create some semantical cohesion that could lead to acceptable results. Yet there is some dispute about the view, since paragraphs may not always indicate a change of topic, but just be invoked to change the visual appearance of the text to ease reading [Sta88].

Symmetric Segmentation

This segmentation divides the document by segmentation factor n. When the document consists of m sentences, then S_d comprises n segments, where each incorporates $\lceil m/n \rceil$ sentences.

Hypothesis: If segmentation has positive effects, then the effectiveness of the back-of-the-book indexing system should increase with rising n.

TextTiling

The discourse segmentation algorithm TextTiling introduced by Hearst [Hea93] targets on the partitioning of text documents in coherent segments. *Coherence* is a relation between the *meaning* of lexical units, such that the combined meaning of these units has a natural connection or consistency. An algorithm maximizing coherence allows a segmentation where the segments express subtopics. We hypothesize that such a topical segmentation combined with the extraction of salient phrases will result in good index entries.

TextTiling partitions the text to coherent units by exploiting *lexical cohesion*. *Cohesion* is a relation between lexical units like sentences or paragraphs, that expresses if the units are tied together by either a relationship in grammatical structure or between words appearing in the unit. *Lexical cohesion* between two units indicates a relationship between the words, that are contained in the units. If two words are identical or semantically related (synonyms, hypernyms etc.), then the containing units are lexically cohesive [JM08]. TextTiling is not sensible for semantic relatedness, for TextTiling two words are only cohesive, when their stems are identical.

An overview over the algorithm:

- **Blocks of Stemmed Tokens:** The rationale behind TextTiling is to partition the text into blocks, where the lexical cohesion with the neighboring block is the lowest. The original TextTiling algorithm divides the text into *pseudo-sentences*, which are token sequences of length w. k pseudo-sequences are composited into disjunct blocks, so that the text is divided into blocks comprising an equal number of tokens: $(t_1^1, ..., t_w^1, t_1^2, ..., t_w^2, ..., t_1^k, ..., t_w^k)$.
- **Calculating Similarities between Blocks:**
  The degree of lexical cohesion for every possible boundary between two neighboring blocks is calculated. For every partition a *similarity score* s is calculated:
  Let A be a block with k pseudo-sentences of length w: $(a_1^1, ..., a_w^1, a_1^2, ..., a_w^2, ..., a_1^k, ..., a_w^k)$.
  Let B be its neighboring block: $(b_1^1, ..., b_w^1, b_1^2, ..., b_w^2, ..., b_1^k, ..., b_w^k)$.
  Then there are $k \cdot 2 - 1$ different similarity scores s_i, measuring the lexical coherence between the blocks:

$$s_{AB_1} = similarity\big((a_1^1, ..., a_w^1), (a_1^2, ..., a_w^k, b_1^1, ..., b_w^k)\big)$$

$$s_{AB_2} = similarity\big((a_1^1, ..., a_w^2), (a_1^3, ..., a_w^k, b_1^1, ..., b_w^k)\big)$$

$$\vdots$$

$$s_{AB_{k*2-1}} = similarity\big((a_1^1, ..., a_w^k, b_1^1, ..., b_w^{k-1}), (b_1^k, ..., b_w^k)\big)$$

  Similarity can be calculated by interpreting the blocks as term frequency vectors and estimating the cosine similarity between them.
- **Boundary Identification:** After applying the previous calculation to all blocks in the text, we retrieve a sequence of similarity scores, which can be plotted as a graph (figure 3.2). After some smoothing the segmentation boundaries are assigned to local minima in a given interval.

As a hypothesis we expect this advanced segmenter to outperform the simpler segmentation methods when parametrized right.

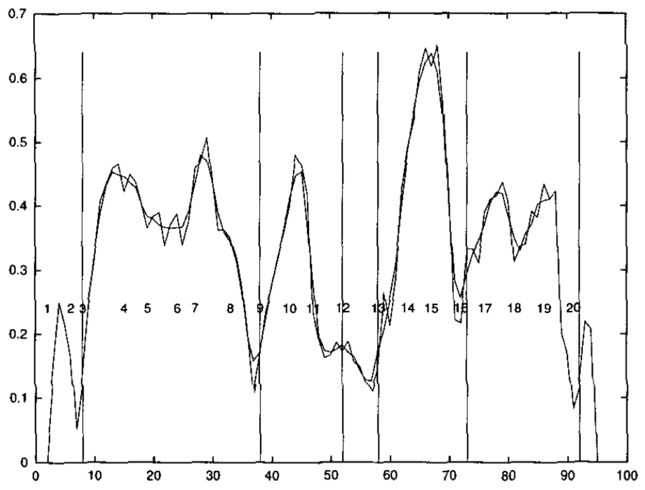


Figure 3.2: Plot of similarity scores (with smoothing). x-axis indicates boundary number. y-axis indicates similarity score. Vertical lines indicate boundaries that are assigned to local minima. [Hea97]

3.2.3 Ranking

Given a set of candidates $C = \{c_1, ..., c_{|C|}\}$ extracted from a document from collection D, we will use a ranker R to associate scores to every candidate:

$$rankedcandidates(\mathrm{R}, C_i, s_i, D) = R := \bigcup_{c \in C_i} \{(c, score(\mathrm{R}, c, s_i, D))\}$$

with $score(\mathrm{R}, c, s_i, D) \in \mathbb{R}$. A ranker can exploit information from the segment s_i and the document collection D as well. The rankers diverge by the way a score is calculated for a candidate, so we will see different scoring mappings for the three rankers Tf.idf, Web1T and TextRank.

Tf.idf

Tf.idf is a classic metric to measure the importance of a word in a document from a given collection. We can use Tf.idf to measure the importance of a candidate phrase in a segment.

The segment-wise Tf.idf for a candidate c and a segment s from the document in collection D is calculated as follows:

$$tfidf(c, s) := \frac{tf(c, s)}{log(df_D(c))}$$

where $tf(c, s) \in \mathbb{N}$ represents the number of occurrences of the candidate phrase in the segment and $df_d(c) \in \mathbb{N}_{>0}$ the number of documents in the collection, where the candidate occurs.

The scoring function utilizes segment and document collection:

$$score(\mathrm{R}_{tfidf}, c, s_i, D) := tfidf(c, s_i, D)$$

The Tf.idf metric is a common baseline, often outperformed by methods incorporating semantics. We will apply it as a baseline to evaluate the other rankers.

Rayson and Garside introduce a method for extracting keywords from documents using an additional *normative* corpus [RG00]. This method is based on the analysis of the word frequency distribution in two or more corpora using log-likelihood. A score can be calculated for each word w in the examined corpus, which signifies its importance based on probability of occurrence compared to the normative corpus:

Taking the examined document as corpus 1 and the normative corpus as corpus 2, a contingency table (3.1) has to be calculated for every word w:

	Corpus 1	Corpus 2	Total
Frequency of all words	A	B	$A+B$
Frequency of word w	a	b	$a+b$
Frequency of other words	$A-a$	$B-b$	$A+B-a-b$

Table 3.1: Contingency table for word frequencies

The next step includes the calculation of the *expected values* E_1 and E_2:

$$E_1 = \frac{A\cdot(a+b)}{A+B} \qquad E_2 = \frac{B\cdot(a+b)}{A+B}$$

The log-likelihood for a word w, occurring in document d for a fixed normative corpus, is calculated as follows:

$$loglikelihood_{R\&G}(w,d) = 2\cdot(a\cdot log\left(\frac{a}{E_1}\right) + b\cdot log\left(\frac{b}{E_2}\right))$$

We decided to use the corpus *Web 1T 5-gram Version 1* as normative corpus [BF06]. The Web 1T consists of n-grams with n in $\{1,...,5\}$ contributed by Google Inc., with frequency counts reflecting the number of web occurrences in 2006. Figure 3.3 shows an example from the 4-gram data, while table 3.2 gives an insight into the dimensions of the corpus by displaying the summed frequency counts for every n:

```
serve as the individual 234
serve as the industrial 52
serve as the industry   607
serve as the info       42
serve as the informal   102
serve as the information         838
serve as the informational       41
serve as the infrastructure      500
serve as the initial    5331
serve as the initiating 125
serve as the initiation 63
serve as the initiator  81
serve as the injector   56
serve as the inlet      41
serve as the inner      87
serve as the input      1323
```

Figure 3.3: Example of the 4-gram data in the Web 1T corpus

n	Count
1	13,588,391
2	314,843,401
3	977,069,902
4	1,313,818,354
5	1,176,470,663

Table 3.2: Summed frequencies of n-grams in Web 1T corpus

In the following, we will call this ranker *Web1T*[1]. Similar to the Tf.idf-ranker, the occurrences of the candidate will be counted in the segments, not in the document. The document collection is not needed. Thus, the score function can be written as follows:

$$score(\mathtt{R}_{web1t}, c, s_i, D) := loglikelihood_{R\&G,web1t}(c, s_i)$$

Since the log-likelihood method, as described by Rayson and Garside, works when extracting keyphrases from texts, we think that it should also work when extracting index entries from books, provided an adequate corpus. The Web 1T corpus is predestinated to be used as a normative corpus because of its size and data. But it is also biased towards a more modern language found in the internet, which can give rise to negative effects using document collections which contain texts from past centuries.

TextRank

Since index entry extraction can be understood as a special application of keyphrase extraction, it is quite obvious to utilize Michalcea and Tarau's TextRank [MT04] as a ranker. The general algorithm was already described in section 2.3, for the application in the indexing domain the we just need to swap index entry candidates for keyphrase candidates.

The processing is applied to the document segment-wise: After choosing a candidate set and a segmentation the ranker builds a co-occurrence graph of candidates for every segment. Then PageRank is applied on the graph to calculate the final scores of the ranked candidates.

TextRank can be parametrized with a window size and if the edges are weighted or unweighted, so the scoring function depends on a parametrization ϕ.

$$score(\mathtt{R}_{textrank,\phi}, c, s_i, D) := textrank(\phi, c, s_i)$$

We hypothesize that there is a configuration of the back-of-the-book indexing system using TextRank, that will yield the highest effectiveness compared to configurations using other rankers.

3.2.4 Aggregation

As a result of the previous stages, we receive a set of ranked candidate for every segment $s_i \in segments(\mathtt{S}, d)$:

$$\hat{R} = \bigcup_{s_i \in segments(\mathtt{S},d)} \{rankedcandidates(\mathtt{R}, candidates(\mathtt{C}, s_i), s_i, D)\}$$

[1] Without whitespace, to distinguish it from the corpus Web 1T.

The task of the aggregator is, to flatten this set and to return a single set of unique candidates with a new score, which reflects the final rank of every candidate phrase:

$$aggregatedcandidates(\mathtt{A}, \hat{R}) = A := \{(c_1, \dot{r}_1), ..., (c_n, \dot{r}_n)\}$$

with $\dot{r} \in \mathbb{R}$ and n the number of unique candidates in this document.

We need to introduce a slightly augmented fundament for the calculations in the aggregators: The set $\hat{R}$ can be written as a matrix $n \times |S_d|$, where n represents the number of unique candidates in the document. Let $r_{i,j}$ denote the score of unique candidate i in segment j, then the candidate-segment-matrix $M_{\hat{R}}$ has the form:

$$M_{\hat{R}} = \begin{bmatrix} r_{1,1} & \cdots & r_{1,|S_d|} \\ \vdots & \ddots & \vdots \\ r_{n,1} & \cdots & r_{n,|S_d|} \end{bmatrix}$$

The columns of this matrix represent segments listing the rank of every candidate in the segment. When the candidate does not appear in the segment s_i, then it has the minimum score, when it appears in the segment, then it has the score provided by the ranker.

Thus, the set of aggregated candidates can be defined in terms of the function *aggregatedscore*:

$$aggregatedcandidates(\mathtt{A}, \hat{R}) = \bigcup_{c_i \in C} \{\ (c_i, aggregatedscore(\mathtt{A}, c_i, M_{\hat{R}}))\ \}$$

Helpful for explanatory purposes are the concepts of *local-* and *global maxima* applied to candidates and documents: A candidate is a global maximum in the document, when it has the highest score of all candidates. It is a local maximum in a segment, if it has the highest score in the segment. Obviously local maximum does not imply global maximum.

We will introduce four aggregation methods: Maximum, sum, top-N-score and top-N-position.

Maximum

The rationale behind maximum aggregation is to merge candidates by keeping the highest score they achieved in one of the segments.

The aggregated score is the maximum score of all scores associated to a c_i in the segments to $\dot{r}_i$:

$$\dot{r}_i = aggregatedscore(\mathtt{A}_{max}, c_i, M_{\hat{R}}) := max(r_{i,1}, ..., r_{i,|S_d|})$$

When a candidate is not a local maximum, it can't become the global maximum.

Sum

Since maximum aggregation takes only the local maximum into account, information from other segments is lost. In the summed aggregation, the score is summed over the scores a candidate achieves in all segments:

$$\dot{r}_i = aggregatedscore(\mathtt{A}_{sum}, c_i, M_{\hat{R}}) := \sum_{j=1}^{|S_d|} r_{i,j}$$

Thus, candidates can theoretically become the global maximum, if they are no local maximum in any of the segments, but appear with a consistently high score in a sufficient number of segments.

Top-N

This aggregation method has to assign new scores that allows it for the later stage to extract just the top N entries from each segment, where N is determined implicitly by the threshold.

- **Top-N-score:** The extraction can be imagined *layer-wise*: It first extracts the highest layer, where a layer consists of all candidates in all segments that have the highest score in a segment. In this layer, the candidates are ranked by this local score, the candidate with the highest score is the global maximum. Then the next layer is extracted, where all candidates have the 2nd highest score. In this layer, the candidates are ranked again by their score, but when a candidate has a higher local score then *any* candidate in the first layer, then it is *not* ranked before it.

  The example in table 3.3 represents two segments with two candidates:

$(c_1, 10)$	$(c_3, 99)$
$(c_2, 9)$	$(c_4, 100)$

  Table 3.3: Example for top-N-aggregation

  The candidates are aggregated, so that their order is: (c_4, c_1, c_3, c_2). Note, that c_1 is sorted before c_3 although its global score is lower.

  This propagates down to the lower layers, until the specified amount of candidates is extracted.

- **Top-N-position:** The inter-layer ranking by score is just one possibility to resolve the collision which would happen when top-n would just order the candidates by the rank in the segments. In that case we end up having $|S_d|$ candidates with the same rank for every layer, so that the inter-layer ordering would be arbitrary. We will evaluate another possibility to resolve the collision using top-n-position aggregation. Instead of inter-layer ranking by score we rank the candidates in each layer by their segment position: Candidates from segment s_1 are ranked higher then candidates from segment s_2 etc.

  Applied to the example segmentation above (table 3.3), we receive the ordering (c_1, c_4, c_2, c_3).

3.2.5 Threshold

The final set of index entries is extracted from the set of aggregated candidates:

$$indexentries(\mathtt{T}, A) = E := \{\ sel_1(a) \mid a \in A\ \wedge\ sel_2(a) > r\ \} = \{e_1, ..., e_{|E|}\}$$

where $sel_i\big((x_1, ..., x_i, ..., x_n)\big) = x_i$ and $r \in \mathbb{R}$ is chosen such, that $|E| = entrycount(\mathtt{T}, d, D) \in \mathbb{N}$.

This models the sorting of the candidates by score and the extraction of a fixed amount of candidates with the highest score which is specified by the threshold $\mathtt{T}$.

There are two thresholds which will be used: $\mathtt{T}_{doc}$ and $\mathtt{T}_{avg}$.

Document-based

The document-based threshold extracts exactly that many index entries as contained in the gold index.

$$entrycount(\mathtt{T}_{doc}, d, D) := |G_d|$$

where G_d is the set of gold entries for this document. This will guarantee a optimal number of extracted entries, so that the demands of the gold standard can be matched theoretically to a perfect degree.

Averaged

The averaged threshold extracts a percentage depending on the word count and the associated gold index of the document. The percentage is averaged over the document collection.

$$entrycount(\mathtt{T}_{avg}, d, D) := |d| \cdot \frac{\sum_{d \in D} |G_d|}{\sum_{d \in D} |d|}$$

This will mimic a standardized ratio of extracted entries, which would be chosen by an experienced indexer. The ratio levels subtle differences between the documents, so we expect inferior results compared to the document-based threshold.

4 Evaluation

In this chapter we will present the evaluation results for the system introduced in chapter 3. Prior to that, we would like to formalize our evaluation methodology. The second section is an analysis of the dataset that is used as a gold standard for evaluation. The chapter closes with the discussion of the results.

4.1 Methodology

In this section we will introduce two necessary measures: The first measure estimates effectiveness of a back-of-the-book indexing system, while the second defines the matching between entries.

4.1.1 Effectiveness of the System

In this section we will provide a definition of the effectiveness[1] of the system, so that we have a clear view what "improvement" will mean for such a system. First we need to clarify its output: A back-of-the-book indexing system $\mathtt{B}$ maps a document d from a collection of documents D to a set of index entries E_d:

$$index(\mathtt{B}, d) = E_d = \{e_1, ..., e_{|E_d|}\}$$

Index entries are specified informally as phrases consisting of one or more words. To evaluate the effectiveness, we need to compare our generated index with an existing index. This index is defined as the gold standard, meaning it is declared as the optimum solution for this task. Let this index for d be G_d, consisting of a sequence of index entries, which we will call *gold entries* g:

$$G_d = \{g_1, ..., g_{|G_d|}\}$$

We can define the effectiveness of a system $\mathtt{B}$ for a document and its associated gold index in terms of a measure of the quality of the created index:

$$effectiveness(\mathtt{B}, d) := m(E_d, G_d) = r \in \mathbb{R}$$

We will have to specify a concrete measure to evaluate the index quality: The creation of the index is considered a classification task [OD08] which allows to view index entries in terms of:

- **True positives:** Index entries appearing in the gold index
- **False positives:** Index entries not appearing in the gold index
- **False negatives:** Entries in the gold index, but not in the created index

Let $match(e, g)$ be a predicate, specifying if an index entry is similar enough to gold entry g, so that it will be contained in the set of true positives. Then we can define the three following sets:

$$TP_d := \{e \mid e \in E_d \land \left(g \in G_d \Rightarrow match(e, g)\right)\}$$

[1] We will use the term "effectiveness" as suggested in [MRS08], to avoid ambiguity introduced by the term "performance", as it used often in the sense of computational performance of the system

$$FP_d := \{e \mid e \in E_d \land \neg(g \in G_d \Rightarrow match(e,g))\}$$

$$FN_d := \{g \mid g \in G_d \land \neg(e \in E_d \Rightarrow match(e,g))\}$$

Following Csomai and Michalcea [CM06] we can now derive these common metrics:

$$precision(d) := precision(E_d, G_d) = \frac{|TP_d|}{|E_d|} \in [0,1]$$

$$recall(d) := recall(E_d, G_d) = \frac{|TP_d|}{|G_d|} \in [0,1]$$

Each of these metrics alone is not sufficient to reflect the quality of the extracted index: An empty index would result in maximum precision, and an index containing all possible phrases in the universe would yield maximum recall. Therefore we will use the *F-measure* [MRS08] as a quality measure. The F-measure is a combined metric trading off precision versus recall. It can be parametrized to emphasize either precision or recall, but we decided to use the equally weighted *balanced F-measure*, which expresses a harmonic mean of the combined metrics:

$$fmeasure(d) = \frac{2 \cdot precision(d) \cdot recall(d)}{precision(d) + recall(d)} \in [0,1]$$

We have established a measure for evaluation of the effectiveness for a single document. It is useful to extend the metric to measure the effectiveness for the document collection D. For this purpose we will use the sum of all F-measure-values in the collection, divided by the collection size, which is called *macro-averaged* F-measure:

$$fmeasure_{macro}(D) = \frac{\sum_{d \in D} fmeasure(d)}{|D|} \in [0,1]$$

Therefore, the we will measure the effectiveness of a back-of-the-book indexing system $\mathtt{B}$ with the macro-averaged F-measure:

$$effectiveness(\mathtt{B}, D) := fmeasure_{macro}(D)$$

4.1.2 Matching

The matching function between index entries and gold entries is crucial for the calculation of the effectiveness, since it determines directly the number of true positives. When applying *exact matching*, $match_{exact}(e,g)$ would evaluate to true, when index entry e consists of the same sequence of characters like gold entry g. Where this would match identical entries, pairs like (*cannon*, *cannons*) would be not considered matches, since their character sequences are different. To capture morphological variations of entries, we follow Zesch and Gurevych [ZG09] and apply a weak form of *approximate matching*:

$match_{approx}(e,g)$ is true when:

- The *normalized* phrases e and g are identical, where normalization transforms some characters in the phrase following this rules:
  - All alphabetic characters are transformed to lower case.
  - Punctuation characters `.!?'¨` are removed.
  - Punctuation characters `-/_` are transformed to a single whitespace.
- The normalized phrases e and g are identical except for an s-character at the end of e or g.

This captures successfully spelling variations like (*token sequence*, *token-sequence*), quantity variations like (*cannon*, *cannons*), and other variations like (*Henry's*, *Henry*). We consider approximate matching an adequate matching strategy without being too tolerant.

4.2 Used Dataset

As described in section 2.2, the number of related works which provide transparent and reproducible evaluation possibilities is very limited. This is correlated with the vanishingly small number of available datasets to perform comparable evaluations in this domain. The only known researchers that have developed a dataset for back-of-the-book indexing and made it publicly available afterwards are Csomai and Michalcea. They created a dataset that should be used, in a modified version, as gold standard for the evaluation of their following automatic back-of-the-book-index extraction system [CM06] [CM07]. A second dataset following the same methodology was created later for the evaluation of a supervised system [CM08], but unfortunately only the first one is publicly available due to copyright issues.

In this chapter we will describe the properties of this first dataset, its documents and the derived index entry sets.

4.2.1 Properties of the Dataset

The dataset introduced by Csomai & Michalcea [CM06] in 2006 consists initially of 56 documents, taken from the collection of electronic books at the Project Gutenberg, a huge digital library consisting mostly of public domain material.[2] The books were collected manually by the researchers, trying only to retrieve documents that contain an index, while paying attention to a level of topical variance in the collection. The final collection covers different topics, not only from the humanities (which is dominant in the Gutenberg Library) like history, literature and psychology, but also from sciences like natural history, botany and physics.

In 2007 Csomai & Michalcea [CM07] introduce an unsupervised back-of-the-book indexing system (see page 14) and use a subset of their original dataset for its evaluation. They exclude all books from the initial collection from the early and mid 19-th century that used an outdated indexing style featuring longer explicative sentences. This reduced collection consists of 29 books.

Another dataset is created a year later [CM08] for the evaluation of a supervised system (see page 15) which comprises 289 books from the University of California Press.

Because this dataset is not publicly available due to copyright issues, we have to use the reduced dataset [CM07] for the evaluation of our proposed system. Fortunately this will allow us to directly compare the results of our system with the unsupervised system of Csomai and Michalcea.

Since this collection is the only dataset that is used by us for evaluation, we will refer to it as the *gold standard*.

4.2.2 Properties of the Documents

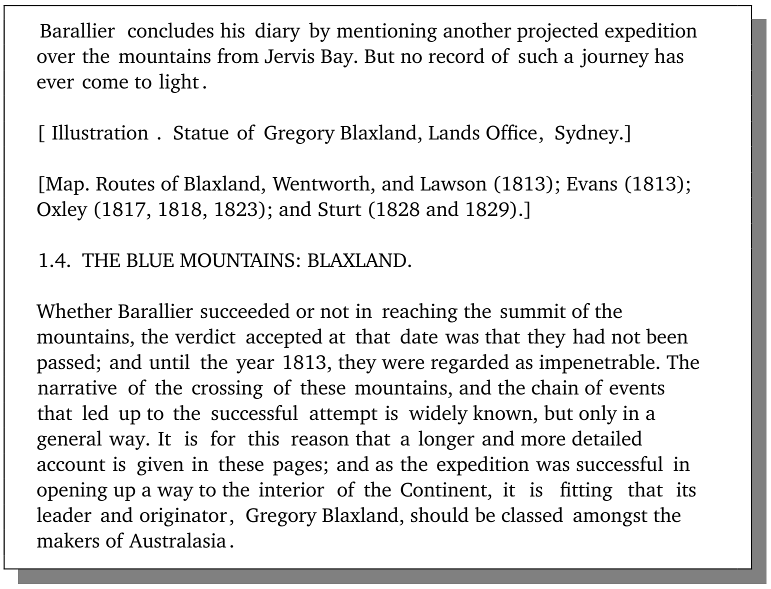
The gold standard dataset consists of a collection of documents and associated indexes for every document. The documents provided in the dataset contain the original text from Project Gutenberg without the back-of-the-book index, which has been manually removed from the document by the researchers.

The syntactical properties of the texts are established by Project Gutenberg: The text does not contain hyphenation, the lines are limited by a width of roughly 70 characters. All information about page numbering has been eliminated. The documents use additional line breaks to structure the text in chapters and paragraphs. Since the text is plaintext, underscores like in _*Institution*_ are used to symbolize emphasis. Many books utilize roman or arabic numbers for chapter enumeration, common are chapter

[2] `http://www.gutenberg.org`

headings usually printed in capitals. Illustration and figure captions, as well as footnotes, are placed inline in the text and annotated via brackets and keywords in the form *[Illustration. ...]*, *(FIGURE ...)*, *[Footnote ...]* or others. Translator notes are found in a at least one of the books, in the form *(... –Translator's Note.)*. Most of these annotations are not unified and vary from book to book.

Figure 4.1 shows an example excerpt, taken from one of the books in the collection.



Barallier concludes his diary by mentioning another projected expedition over the mountains from Jervis Bay. But no record of such a journey has ever come to light.

[Illustration . Statue of Gregory Blaxland, Lands Office, Sydney.]

[Map. Routes of Blaxland, Wentworth, and Lawson (1813); Evans (1813); Oxley (1817, 1818, 1823); and Sturt (1828 and 1829).]

1.4. THE BLUE MOUNTAINS: BLAXLAND.

Whether Barallier succeeded or not in reaching the summit of the mountains, the verdict accepted at that date was that they had not been passed; and until the year 1813, they were regarded as impenetrable. The narrative of the crossing of these mountains, and the chain of events that led up to the successful attempt is widely known, but only in a general way. It is for this reason that a longer and more detailed account is given in these pages; and as the expedition was successful in opening up a way to the interior of the Continent, it is fitting that its leader and originator, Gregory Blaxland, should be classed amongst the makers of Australasia.

Figure 4.1: Excerpt from a book in the dataset

For the introduced class of algorithms, these annotations and structuring tokens have different effects: Line breaks and underscores can be easily filtered, since they can be excluded from the set of tokens, that is the base set for all candidate sets. The additional line breaks have no negative side-effects, and can be furthermore exploited by text segmentation methods, which try to follow the natural structuring of the book like the on described in section 3.2.2.

Chapter numbering, figure and illustration captions, editorial and figure annotations can tempt to their erroneous classification as candidates. For instance, the tokens *Illustration* and *Translator* are nouns and will appear in the set of noun-phrase chunks and in the set of noun tokens. Therefore they are considered as unwanted noise.

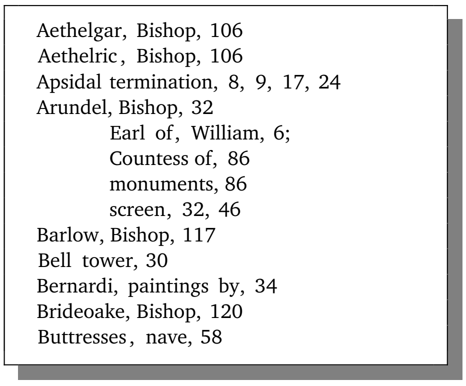
Apart from the body, the documents contain a front matter consisting of items like the caption of the *frontispice*, the title page (containing title and editorial information), the preface, acknowledgements, a list of contents and the list of figures. According to Mulvany, the front matter should not be indexed by a human indexer. For an automatic indexer the presence of the front matter can be seen as noisy material that needs special handling.

Since there are no clues that Csomai and Mihalcea apply any special procedures to deal with the described annotations or elements of the front matter, we suppose that they use the document *as-it-is* as input for the automatic indexer.

4.2.3 Properties of the Gold Index Sets

The original indexes are manually extracted from the Project Gutenberg documents. They differ in size, style and indexing strategy.

Figure 4.2 shows a original index in its raw form.



Aethelgar, Bishop, 106
Aethelric, Bishop, 106
Apsidal termination, 8, 9, 17, 24
Arundel, Bishop, 32
    Earl of, William, 6;
    Countess of, 86
    monuments, 86
    screen, 32, 46
Barlow, Bishop, 117
Bell tower, 30
Bernardi, paintings by, 34
Brideoake, Bishop, 120
Buttresses, nave, 58

Figure 4.2: Excerpt from an index from the dataset

This index contains locators and semicolons, but other indexes may also contain other unwanted tokens, including index-specific annotations like *seq.*, or *see also* as was displayed in figure 2.1 in chapter 2.1. The headings of the index have also been inverted, meaning a reorganization of the indexed phrase by the human indexer, so that the main heading can be used as a keyword for lookup. This inverted phrase will not likely be found in the original text, rendering the matching with any extracted phrase from the text impossible.

To achieve a unified and machine-readable style of the indexes, they are pre-processed eliminating all unwanted tokens and applying a standardized formatting. The second issue, the inversion of the headings, is addressed by an algorithm to reconstruct the original headings.

In the following sub-sections we will describe a total of three index sets that where constructed by the researchers based on the original index set.

The fine-grained Index

An inverted heading is a phrase, that begins with a noun phrase and is followed by phrase units, all of which are delimited by commas. Csomai & Mihalcea apply a heuristic reconstruction algorithm that tries to reorder the phrase units, so that the resulting phrase is likely to be found in the text, thus suiting the extractive approach:

The phrase is split into components using the comma as a delimiter. These components are scanned for a preposition at the beginning or the end, making the component a prepositional phrase.[3]

The new position of the component relative to the main heading is depending on the preposition and its position:

- **P1:** In the case of *as*, *from*, *of*, *among* at the beginning , the prepositional phrase is ordered before the main heading.
- **P2:** In the case of *to*, *in*, *for* at the end, it is ordered after the main heading.

In the absence of prepositions the following patterns are exploited:

- **C:** If the component begins with the conjunction *and*, it is placed before the main heading.

[3] Csomai & Michalcea do not provide any (Pseudo-)Code, we fund our interpretation of the algorithm on their informal description in the publication. Mapping to concrete rules provides by us.

- **A:** If the second component is a adjective or adverb, it is considered as a modifier and sorted before the main heading.
- **O:** In other cases, the component is placed after the main heading, as are all other components that are not matched by any of the rules.
- **H:** An index entry, which is broken into subentries, is reconstructed by attaching the main heading to each of the subheadings and processing each of the extended subheadings recursively.

Csomai & Michalcea further note, that all rules that are not based on propositional phrases are considered as "back-off solutions" and that the decision, which of those rules has to be applied, is made for every document individually. In practice they measure the frequency of the reconstructed entries in the document after the application of a rule and keep the set of reconstructed entries which has the highest frequency.

Table 4.1 shows examples provided by Csomai & Michalcea in [CM06]:

	Index entry	Reconstruction	Rule
1	*Acetate, of Ammonium Solution, Uses of*	*uses of Acetate of Ammonium Solution*	P1
2	*Goldfinch, American*	*American goldfinch*	A
3	*Goose, Domestic*	*domestic Goose*	A
3	*Cainozoic, term defined*	*cainozoic term defined*	O
4	*France,*		H
	- history of the use of subsidies in,	*history of the use of subsidies in France*	P1
	- the navigation laws of,	*the navigation laws of France*	P1
	- commercial treaty between England and,	*commercial treaty between England and France*	C
	- the Merchant Marine Act of,	*the Merchant Marine Act of France*	P1

Table 4.1: Examples for inversion reconstruction

Unfortunately there are some problems and inconsistencies with the applied algorithm when examining the dataset. Listed in table 4.2 are examples from three different books from the collection which show some exemplary shortcomings.

In (1) the second component ends with the preposition *of*, but is not placed before the main heading. The third component is leading the reconstructed heading, which is correct as a result, but not a correct application of rule *O*. The other expected subheadings *Countess of Arundel*, *Arundel monuments* and *Arundel screen* are not part of the fine-grained index, although they should have been – the phrases *Countess of Arundel* and *Arundel screen* are contained in the original text, and would have been legitimate gold index entries possible to be found by an extractive algorithm.

In (2) the algorithm misses the special form of the index used in the book. The first entry is an inverted heading which is followed by subheadings. The subheadings should be reconstructed using the main heading *Jackson*, but instead they are "flattened": All following entries are considered independent of the first index entry, which is defining the context, and raised to the the same level, which is heading instead of subheading. There are two things which are special about this particular index which raise the difficulty of reconstruction: Firstly, the subheadings have a logical, but no grammatical relationship with the main heading, thus they show a high degree of abstraction describing events and properties related to *Jackson* which are results of semantic analysis made by the indexer. Secondly, the index entry contains entries in its hierarchical structure, that serve as categories for following subheadings (f.e. *Appointments*). The category names are interpreted by the algorithm as main headings as well, having the drawback of adding very general entries to the gold index, which would be not found as headings in the index at all.

The problem shown in (3) is related to (2): The original index is in run-in style (see section 2.1.2), featuring subheadings with mostly logical relationship to the subheading. Instead of raising the subheadings to main headings, they are just ignored by the algorithm and not part of the fine-grained index.

	Index entry	Found reconstruction
1	*Arundel,*	
	— Earl of, William	*William Arundel Earl of*
	— Countess of,	-
	— monuments,	-
	— screen,	-
2	*Jackson, Thomas Jonathan, "Stonewall", [...]*	*Thomas Jonathan Jackson "Stonewall" [...]*
	— Advice overruled.	*Advice overruled*
	— Anecdotes of.	*Anecdotes of*
	Appointments:	*Appointments*
	— To Cadetship.	*To Cadetship*
	— First Regiment of artillery, U.S.A.	*First Regiment of artillery*
3	*Bismarck, Otto Eduard Leopold von,*	*Otto Eduard Leopold von Bismarck*
	— his birth,	-
	— ancestry,	-
	— at school in Berlin,	-
	— enters at Goettingen,	-
	— his personal appearance and character,	-

Table 4.2: Examples for inconsistencies of the inversion reconstruction algorithm

Since the exact procedure of the reconstruction is not described by the authors, it is difficult to find a plausible explanation for erroneous behavior as in (1), although there is a slight irregular indentation at the index position that could mislead the reconstruction. Problems as in (2) and (3) can be explained by bringing to mind the diversity of the index styles found in the collection. The original indexes are differing in layout, in the use of delimiters between subheadings etc. All those differences makes it difficult to find a unified procedure that handles all indexes with the same quality.

The index containing all reconstructed phrases is referred to as *fine-grained index*.

Even when the reconstruction works as described, the reversal of the inversion can yield ungrammatical phrases, since subheadings and main heading may not be grammatically related. How to deal with these phrases is discussed in the next section.

The filtered Index

Some examples of ungrammatical phrases which are created using the reversion are:

1	*decomposition of by vegetables Water*
2	*domestic Cat singular property of its fur*
3	*or orchard oriole orchard Starling*

Csomai & Michalcea apply a simple filtering strategy using a web-search engine. They measure the number of occurrences of each phrase on the Web using AltaVista[4], and drop every index entry that yields less then two search results. This threshold is determined empirically and results in reduction of

[4] http://www.altavista.com

roughly 25% of the fine-grained index set.[5] This method works quite well to eliminate obscure phrases efficiently. Obviously longer phrases are more prone to elimination then shorter ones.

The coarse-grained Index

Because there are some issues with the creation of the fine-grained index and the filtering can only solve the problem of ungrammatical phrases, a third index set was created. Each *coarse-grained index* contains only the main headings from the original index: From the headings in table 4.2 only the main headings *Arundel*, *Jackson* and *Bismarck* are included in the indexes, and all following phrase units and subheadings are dropped. This method is simple, but effective: The result is an set of headings which are most likely contained in the original text, and are relevant enough to be counted as index entries. There are obviously some problems in the approach, as these coarse index entries are by trend more general then the reverted headings (assuming they are reverted correctly). Information encoded in modifiers like adjectives and adverbs is lost, which can be pretty much looking at examples like *argillaceous Earths* and *calcareous Earths*, which could be correctly reconstructed from inversion and are both mapped to the very general main heading *Earths*. Looking further at the gold entries in the dataset, some imperfections become visible: In an coarse index the phrase *Bismarck* appears 15 times, while the terms *Arnim*, *Brandenburg* and *Vienna* appear twice. When these duplicate entries are not filtered in the evaluation, they will sneak in higher costs when classified as false negatives.

Comparison of the Indexes

As described, some indexes are transformations of other indexes. These relationships are depicted in figure 4.3.

Due to the transformations, there are significant differences in the numbers of index entries for every index type: The fine-grained index set is the biggest, followed by the coarse-grained index, while the filtered index is the smallest of the three.

Another important aspect already mentioned is the *presence* of index entries in the text. An entry is present when it is contained in the text – then it is possible for an extractive method to find a phrase which covers it. The upper bounds of presence form the upper bounds for recall as well. Table 4.3 shows the total size of the index sets and the percentage of the total entries that could be found in the text.[6]

So, which set is most suitable for evaluation purposes? Csomai and Michalcea chose the coarse-grained index, since it has the highest presence in the collection and therefore works best when applying the extractive approach. To maintain comparability, and because we see problems in the other branch of index sets, we will chose the same, but not without expressing some concerns about the coarse-grained index: Due to the trimming process it is strongly biased towards single nouns and short noun-phrases .

The problems of trimming adjective and adverbs prematurely was already discussed. Since many main headings are proper names, many named entities are to be expected in the coarse-grained index. It is the

[5] Csomai & Michalcea write of a reduction of "roughly 50%". It is not clear what is exactly reduced, but we have calculated the percentage of reduction based on the total counts of the index entries in the whole fine-grained and the whole filtered set as approx. 25%. See table 4.3 for counts.

[6] The numbers we achieve are quite different from Csomai & Michalcea which report presence for *fine-grained* = 30.34%, *filtered* = 54.78% and *coarse-grained* = 81.29%. They do not state if/what preprocessing they apply before the comparison. We use conversion to lower case as shown in the table and eliminate all line breaks in the original text to correctly match multi-words, even when they are separated with a line break. The comparison is implemented by simple substring comparison.

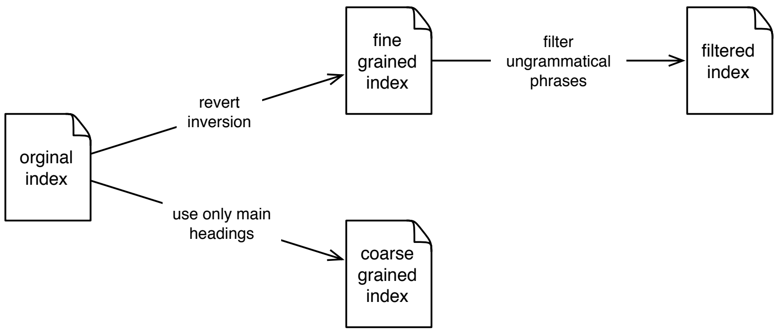


Figure 4.3: Relationships between the indexes

Index type	Suffix	#entries	Presence in text	
			Lower case	**Orig. case**
original index	.index	ca. 25000	n/a	n/a
fine-grained index	.index.full	16411	52.05%	13.99%
filtered index	.index.full.filtered	12602	64.71%	17.30%
coarse-grained index	.index.trimmed	13641	83.26%	23.36%

Table 4.3: Presence of Index Entries in the Text. The number of original index entries is a rough estimation, since the indexes are formatted differently and entries can't be counted automatically.

question, if such a trimmed index consisting of short phrases sufficiently reflects a humanly constructed index, which consists over half of phrases, which are longer then four tokens (see table 4.4).

It is comprehensible to use an algorithm to derive gold index phrases, since it scales better for an arbitrary number of books, compared to creating the gold entries manually, which does not scale at all. But we think, that a gold standard of unreversed entries, maybe created with constraints like a minimum presence in the text, would be a lot more realistic and meaningful, then the biased gold standard here — but more expensive too.

length	fine-grained	coarse-grained
1	16.3%	69.4%
2	18.5%	20.1%
3	15.1%	4.9%
> 4	50%	5.5%

Table 4.4: Distribution of index entry lengths (in tokens) in gold index. Percentages derived from the superset of our gold standard ([CM06]).

4.3 Evaluation Results

In the first section we will examine the potential of the different candidate sets introduced before. After choosing three candidate sets for the main experiments with our system, we will discuss the results, including the effects of segmentation and related topics in the second subsection. A comparison with benchmarks created by another system using the dataset is presented in the last subsection.

Note: We will apply approximate matching, if not stated differently.

4.3.1 Candidate Sets

There are a few quality criteria for candidate sets: Recall on the dataset is important, because it forms the upper bound for recall after the ranking- and extraction-by-threshold steps. The size of the set plays an important role as well, because huge candidate sets can be computationally difficult to handle. Size of a candidate sets often comes with low precision, as the set likely contains many low-quality candidates like non-grammatical phrases and such.

We will chose the best configuration out of three distinct candidate types for further experiments. The types were introduced in section 3.2.1 and comprise n-grams, POS-filtered tokens, named entities and noun-phrase chunks.

As addressed in section 3.2.1, we tested multiple filtering settings when creating n-grams with n in {1,...4}. The matching strategy was *exact*, the differences were quite subtle and tabulated in 4.5. The best setting yields the highest recall and F-measure:

Setting	P	R	F	#entries
No filtering	0.132	81.33	0.263	8306012
Filter single characters that indicate sentence endings	0.137	81.32	0.273	8002447
Filter single characters that are not alphanumerics	0.146	**81.63**	**0.292**	7510459

Table 4.5: Effects of filtering methods using n-grams (exact matching)

As shown before in table 4.3, the upper bound for the possible recall for extractive approaches is 83.26. An recall of 81.63 using n-grams with a maximum length of 4 tokens shows that the phrases consisting of more then 4 tokens are considerably rare, which is consistent with the findings in table 4.4.

We tried creating a set by filtering tokens with different part-of-speech tags. The results listed in table 4.6 were evaluated with exact matching, and comprise different settings trying to maximize recall. It was first surprising to see nouns having such high F-measure but after an analysis of the dataset (see section 4.2) it became consistent with the bias of the gold entry set towards nouns and noun phrases. Since the setting delivering the highest recall comes with a considerable low F-measure, we declare the simple noun set as interesting enough to use it in the forthcoming experiments.

Two different implementations of named entity recognition were tested: Our simple heuristic named entity recognition, and a state-of-the-art implementation based on *conditional random fields* (CRF) from Stanford University [FGM05]. As shown in table 4.7, the simple heuristic performs surprisingly strong. The CRF implementation yields better results, but needs also training effort and more processing time. Considering only the results, it is difficult to decide which of these sets is to prefer, but in the end we chose state-of-the-art over our effective heuristic and expect best results using this set.

Looking at table 4.8 listing the best candidate sets, the named entities outperform all other sets with their relatively high precision combined with a good recall. This fits to the thesis that the index entries in the gold standard consist to a large part of proper names. The n-grams yield the highest recall of all

Setting	**P**	**R**	**F**	**#entries**
Adjectives	0.437	1.47	0.673	45329
Nouns	5.917	64.16	**10.83**	146094
Nouns and adjectives	4.675	64.48	8.718	185811
Nouns and adjectives merged	2.956	68.84	5.668	313754
Nouns and adjectives merged (kept parts)	2.682	76.94	5.183	386500
Nouns and adjectives (kept parts, no single adjective)	2.977	76.62	5.73	346783

Table 4.6: Comparison of different POS-filtered token sets (exact matching)

Setting	**P**	**R**	**F**	**#entries**
NE Heuristic	15.01	45.47	21.36	48613
NE Stanford CRF	19.84	38.99	**24.53**	31069

Table 4.7: Comparison of different named entity recognition methods (exact matching)

candidate sets, but we fear the size of the set will cost to much processing time, while not paying off with good results, considering the low precision. Therefore we drop *n*-grams and declare the remaining noun-based candidate sets together with the named entities as superior.

Candidate Set	**P**	**R**	**F**	**#entries**
Noun-phrase chunks	3.95	69.06	7.39	287481
Nouns	7.25	62.86	12.68	146094
N-grams	0.27	**85.37**	0.53	7510459
Named Entities ([FGM05])	**19.84**	38.99	**24.53**	31069

Table 4.8: Comparison of the best candidate sets

Since the amount of index entries in a book must be limited somehow to mimic publishing standards, two thresholds where introduced in section 3.2.5.

Both thresholds represent a percentage of the extracted index-entry-candidates that is taken into account as final index entries. The percentage for T_{avg} is derived from the ratio of document size in the collection and the associated gold indexes in the given gold standard. For our gold standard we calculated the percentage of index entries to be retrieved per document with 0.372%. This corresponds to the results of Csomai and Michalcea [CM07] which estimate the average index length with 0.35%.

As shown in tables 4.9 and 4.10 the document based index length yields better results the averaged index length. This is not a big surprise, since knowing the index length of the gold standard is a valuable information increasing the chance of being on par in terms of precision and recall.

Since the improvement of the results is consistent for all candidate sets and rankers, we will stick to the averaged length, which also preserves a better comparability to Csomai and Michalcea, who apply a similar threshold. Therefore we will focus on the results in table 4.9:

To evaluate the impact of the rankers, the first row shows results achieved by extracting the specified amount of index entries without applying any ranking, which leads to the extraction of the first *n* candidates that occur in the document. Comparing the results of the rankers introduced in section 3.2.3, there is a hierarchy which holds for all candidate sets: Tf.idf is more effective then Web1T which performs better then TextRank (configured with co-occurrence window size 2). There was little experience with TextRank applied to long documents, which yields the worst results compared to other rankers. Yet it is surprising that both TextRank and Web1T get outperformed by the rather simplistic metric Tf.idf on all candidate sets. Another surprising result is, that rankers like Web1T and TextRank seems to worsen

Ranker	Nouns			Noun-Chunks			Named Entities		
	P	R	F	P	R	F	P	R	F
Position	12.01	11.16	10.55	8.48	9.11	7.99	22.50	21.69	20.37
Tf.idf	12.95	12.48	11.29	15.69	14.29	13.24	28.67	26.96	25.52
Web1T	10.60	10.57	9.42	13.52	12.78	11.66	26.94	24.44	23.61
TextRank (w=2)	6.71	6.31	5.87	11.25	11.93	10.36	26.66	24.91	23.83

Table 4.9: Baseline for unsegmented documents applying threshold T_{avg}

the performance, compared to position-ranking using the noun set. This can be explained with the fact that nouns, which occur in the beginning of the document, tend to be more important. As we showed in section 4.2, some documents begin with a table of contents containing nouns which are most likely indexed as well, thus leading to good results using position-ranking.

Ranker	Nouns			Noun-Chunks			Named Entities		
	P	R	F	P	R	F	P	R	F
Position	11.48	10.60	11.36	8.22	7.72	8.24	22.60	21.07	23.83
Tf.idf	13.88	11.69	12.65	15.96	13.23	14.43	28.08	26.27	27.08
Web1T	11.35	9.65	10.40	13.87	12.01	12.85	27.02	24.58	25.55
TextRank (w=2)	6.59	6.20	6.38	11.34	10.69	10.99	26.35	24.66	25.39

Table 4.10: Baseline for unsegmented documents applying threshold T_{doc}

4.3.2 Effects of Segmentation

To evaluate if segmentation improves back-of-the-book indexing, we will segment the documents to a number of equally sized parts with rising degree of segmentation. We will apply four different aggregation methods (see section 3.2.4) and compare the effectiveness with rising segmentation degree in $\{1, 2, 4, 8, 16\}$. The effectiveness as F-measure is plotted in figures 4.4, 4.5 and 4.6.

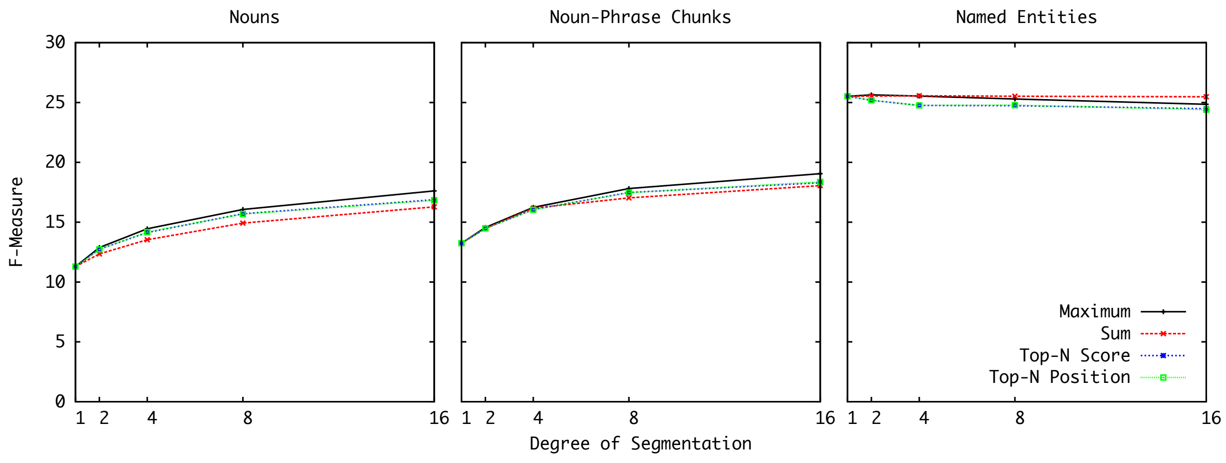


Figure 4.4: Symmetric segmentation using Tf.idf

All F-measure-graphs begin at the point $(1, y)$ with y being the baseline effectiveness for an unsegmented document introduced in table 4.9. Generalizing over all rankers and aggregation methods, the graphs for nouns and noun-phrase chunks have a clearly ascending slope, which is interpreted as a general positive effect of segmentation for these candidate types. The effectiveness improves continuously with increasing segmentation degree.

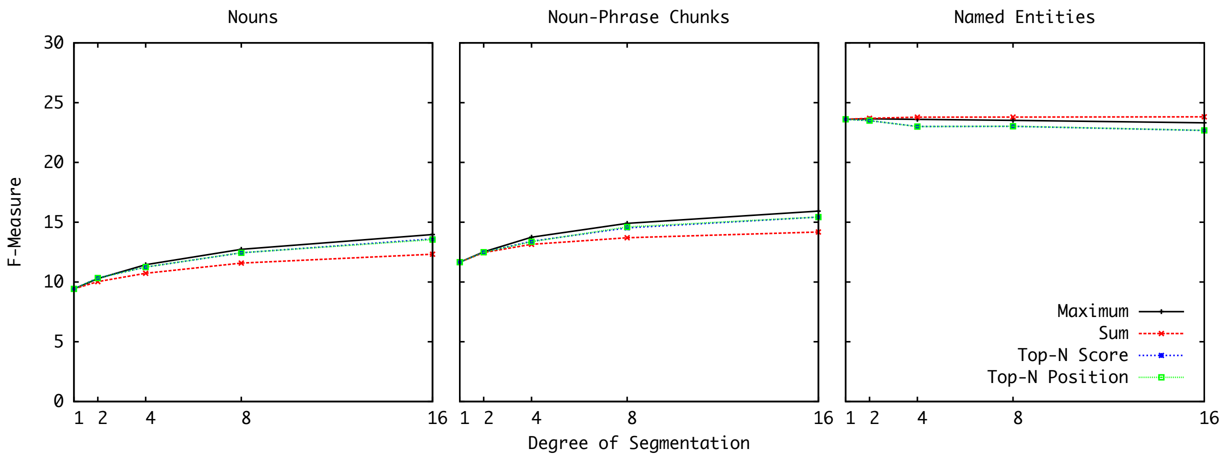


Figure 4.5: Symmetric segmentation using Web1T

Segmentation does not seem to have a positive effect on the named entity set. The graphs for the named entities have a slightly descending tendency except for TextRank, where the sum aggregation graph slightly rises above the baseline. The set of named entities seems to react very gently to segmentation compared to the noun-based candidates. Segmentation in conjunction with Tf.idf and Web1T worsens the effectiveness minimally. Yet we can notice an slight increase in F-measure with segmentation degree of two using TextRank together with the sum aggregation method.

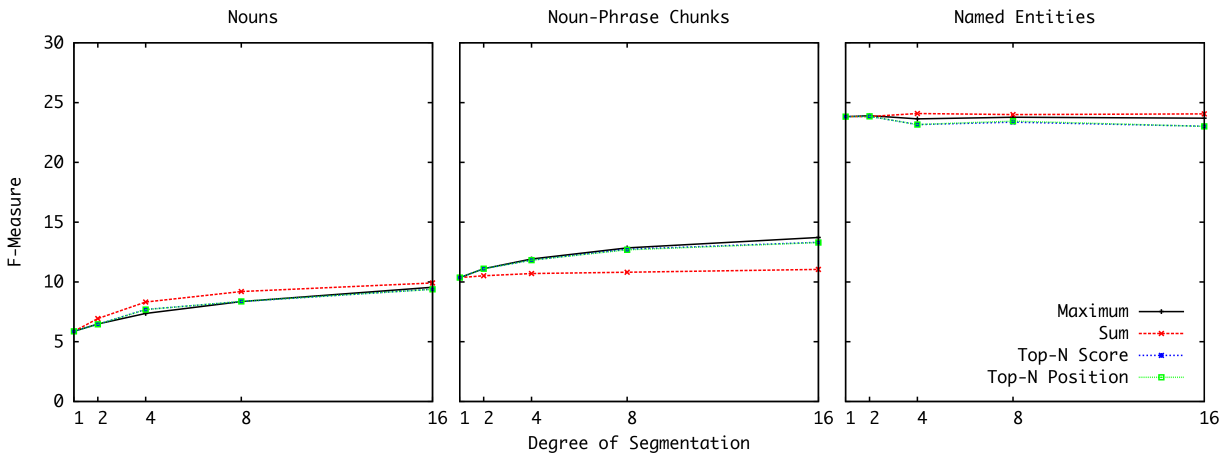


Figure 4.6: Segmentation by sentence using TextRank

By applying symmetric segmentation we started from an unsegmented document and increased the degree of segmentation. We also applied experiments in the other direction, starting from small segments and increasing the segment size. If an increasing segmentation degree leads to higher effectiveness, then a rather fine-granular segmentation could eventually correlate with a significantly high F-measure. To verify this thesis, we evaluated a partitioning in segments consisting of n sentences in the range $\{5, \ldots, 200\}$ using Tf.idf and TextRank as rankers (depicted in figures 4.7 and 4.8).

The graphs begin at point $(5, y)$ where y represents the F-measure using a partitioning into five-sentence segments. For the noun-based candidates, the global maximum is reached using maximum aggregation

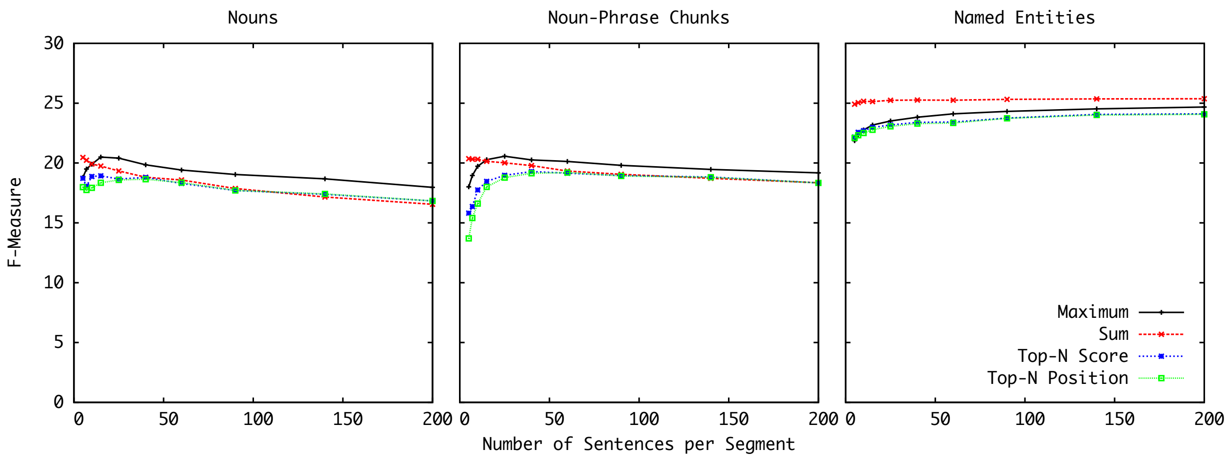


Figure 4.7: Segmentation by sentence using Tf.idf

for an x somewhere in $\{10,...,25\}$. After this point effectiveness decreases gently and monotonous. This trend will persist until reaching the baseline associated with the unsegmented document.

The influence of segmentation on the named-entity set is different compared to the noun-based candidates. At a segmentation of five sentences per segment the maximum- and TopN-aggregation measures are below the baseline for an unsegmented document and increase slightly, until they reach the baseline. We do not observe a local maximum in the area of 15 to 25 sentences per segment followed by a continuous descent, the trend heads rather into the opposite direction. The graph for the sum aggregation is almost flat except of few fluctuations as was observed applying symmetric segmentation. There is only conclusion we can take: Partitioning into small segments does not seem to improve effectiveness on the named entity set.

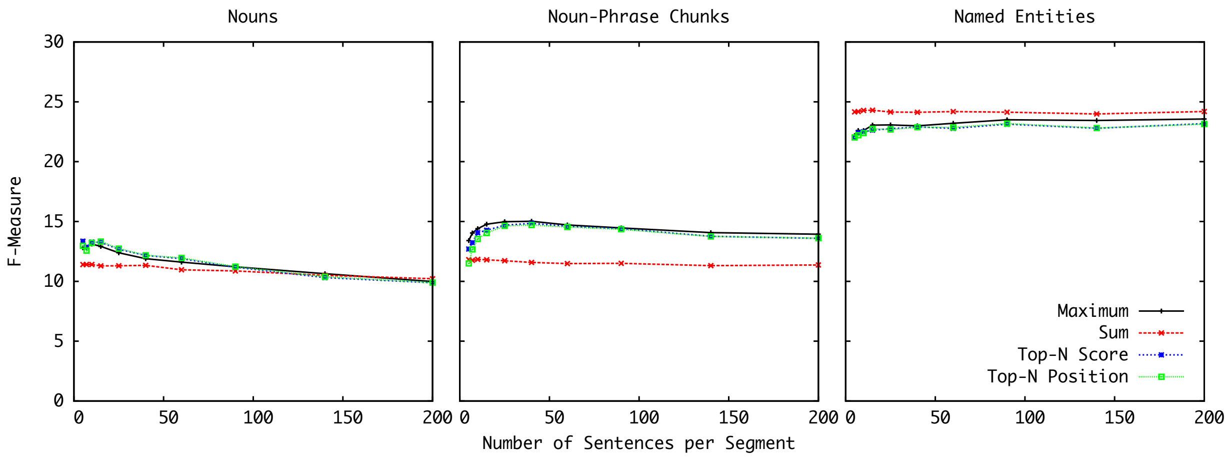


Figure 4.8: Segmentation by sentence using TextRank

Table 4.11 uses the same numbers as in the plots and compares the base effectiveness with the best effectiveness achieved by the means of segmentation. As in the graph, the results clearly support the thesis of positive effects on segmentation on basic candidate types like nouns and noun-phrases. For sophisticated types like named entities the positive effects are marginal, if even statistically significant. It is more likely that segmentation impairs effectiveness instead of improving it.

Candidates	Ranker	Eff.	F-measure	Segmenter	Aggregation
Nouns	Tf.idf	Base	11.293	Unsegmented	Max
		Best	**20.498**	Sentence (n=15)	Max
	Web1T	Base	9.422	Unsegmented	Max
		Best	17.650	Sentence (n=10)	Max
	TextRank	Base	5.868	Unsegmented	Max
		Best	13.357	Sentence (n=5)	TopN-Score
Noun-phrase chunks	Tf.idf	Base	13.242	Unsegmented	Max
		Best	**20.564**	Sentence (n=25)	Max
	Web1T	Base	11.655	Unsegmented	Max
		Best	17.562	Sentence (n=10)	Max
	TextRank	Base	10.364	Unsegmented	Max
		Best	15.017	Sentence (n=40)	Max
Named Entities	Tf.idf	Worst	21.863	Sentence (n=5)	Max
		Base	25.515	Unsegmented	Max
		Best	**25.636**	Symmetric (sd=2)	Max
	Web1T	Worst	21.363	Sentence (n=5)	Max
		Base	23.610	Unsegmented	Max
		Best	23.988	Sentence (n=10)	Sum
	TextRank	Worst	22.014	Sentence (n=5)	Max
		Base	23.825	Unsegmented	Max
		Best	24.298	Sentence (n=15)	Max

Table 4.11: Influence of segmentation on effectiveness. Since segmentation leads to improved effectiveness using the noun-based sets, the base (unsegmented) effectiveness is the worst effectiveness at the same time. This is not the case with the named entity set, where effectiveness can decrease with rising segmentation degree.

Discussion of Tf.idf

We observe, that using Tf.idf as ranker, combined with a partitioning into segments of 15 - 25 sentences leads to best results with noun-based candidates. But why is segmentation not improving results using named entities?

To answer this question, we will analyze one of the documents with a *phase distribution* plot (figures 4.9, 4.10) which shows the distribution of true and false positives in the document: In this plot the x-axis represents the position in the text, while ranked phrases are plotted on the y-axis. Given a document with a set of ranked candidates, each candidate is associated to an y-value, representing the rank according to its Tf.idf-score[7]. The global maximum has rank 1, the global minimum rank $|R|$, where R is the set of ranked candidates. A hit (true positive) is a ranked candidate, that appears in the associated gold index, whereas a miss (false positive) appears not.

4.9 shows the phrase distribution of one of the shorter documents in the collection using the set of noun-phrase chunks. As visible, the multitude of ranked candidates are false positives with the biggest portion of true positives in the upper part. Some true positives are scattered in the document having low ranks indicated by high y-values. The false positives are uniformly distributed across the whole range. An extraction of the top n candidates would mean the retrieval of all ranked candidates, where y is in $\{1, \ldots n\}$, including many false positives. A perfect ranker would separate the candidates in such way, that

[7] The difference between score and rank: Having candidates c_1 and c_2 with $score(c_1) = 1$ and $score(c_2) = 10$, this results in $rank(c_1) = 2$ and $score(c_2) = 1$.

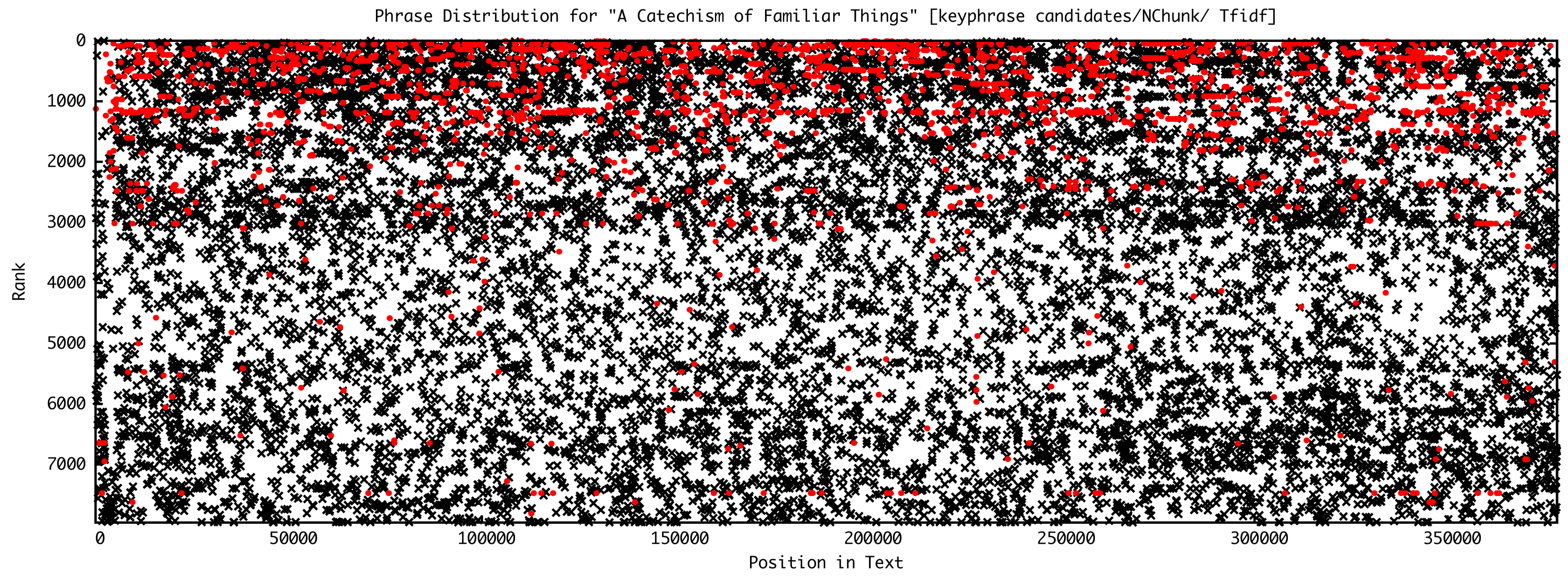


Figure 4.9: Phrase distribution plot of noun-phrase chunks ranked with Tf.idf. Segmentation leads to increase of effectiveness.

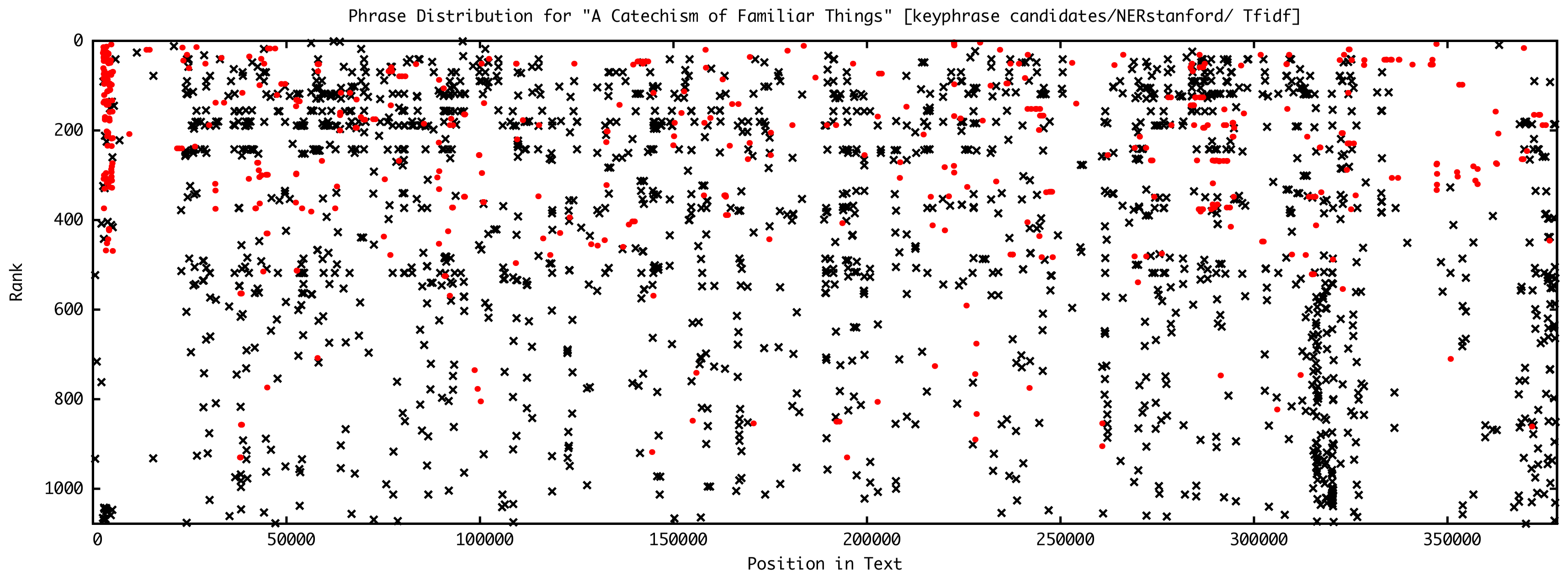


Figure 4.10: Phrase distribution plot of named entities ranked with Tf.idf. Segmentation leads to dicrease of effectiveness.

the upper part of the plot is consisting only of true positives and the lower part only of false positives. Then the extraction of a set of perfect index entries would be just a matter of choosing the right n.

Comparing the noun-phrase-based plot to the named-entity distribution in figure 4.10, the density of candidates is much lower. We hypothesize, that the decrease of F-measure by segmentation is due to the lower density of candidates and distribution of true positives in the document.

The chosen document has a sensitivity towards segmentation: The F-measure falls from 33.08 to 28.82 by applying a sentence based segmentation with $n = 25$. With the same segmentation the F-measure in the noun-phrase-chunk set increases from 19.27 to 36.78, which is even above the base performance of the named entity set.

Applying the average-base threshold will retrieve all phrases from rank 1 - 236, which does already include a considerably high number of true positives in the named entity set. The sequences of true positives have gaps, where we can discern ranges where the local maximum is a false positive. We think, that this may be the major reason for the impairment by segmentation: Maximum aggregation will retrieve a false positive as index entry. In such fragmented distributions, a global ranking seems to be the better choice.

Returning to the noun-phrase distribution, the explanation for a better performance is more difficult to find. The first 236 candidates mark a small area of crowded true and false positives in the upper top of the plot. Yet we observe over the half of the true positives below this area. This distribution may be the reason why the efficiency of an extraction method without segmentation may be low at the first place. We know by experiment result, that segmentation nearly doubles F-measure, which is based on an increase of precision from 34.32 to 62.71. The reason for the increase may be *phrase sequences* appearing in the document. Phrase sequences are multiple occurrences of phrases in a range of the document, denoted by many true positives or false positives in a row. Phrase sequences of true positives can be recognized by points on the same y-plane in short succession, ideally forming a closed line. Figure 4.9 shows many phrase sequences of true positives below the y-plane where $y = 236$. We hypothesize, that when segmentation have a positive effect on a frequency based ranking, it is by exploiting phrase sequences of true positives. When a occurrence sequence with a high density is contained in a segment, then it is likely that the occurring phrase is one of the candidates with the highest local score or possibly the local maximum. Table 4.12 shows an abstraction of this – although candidate a is the global maximum, candidates b, c and d would be chosen as index entries when applying maximum aggregation, since they are local maxima.

	d	s_1	s_2	s_3
$\#a$	6	2	2	2
$\#b$	5	**4**	0	1
$\#c$	4	1	0	**3**
$\#d$	3	2	**3**	0

Table 4.12: Local maxima in segments.

Discussion of Web1T

Comparing the highest efficiency of Tf.idf, Web1T and TextRank, Web1T is second best.

We see Web1T as conceptually similar to Tf.idf. It is based on term frequency, which is normalized by the relevance of the term in a larger document corpus. Although the calculation of the score is fundamentally different, we think that all insights that apply to Tf.idf regarding segmentation effects mentioned above should apply to Web1T as well.

One of the reasons, why Web1T performs inferior to Tf.idf could be the bias towards modern language. All books in the gold standard are between 80 to 150 years old, while the Google-*n*-grams capture the internet in the year 2006. This can have the effect, that common phrases from yesterday may be the rare phrases from today, which wrongly boosts the significance of the phrase, creating a false positive. Looking at the ranked phrases, Web1T seems to have the tendency of privileging very short noun-phrase chunks like *st, hat, pa, tin, ear*, which were the top phrases in a document we examined. It still has to be investigated what the exact reason for this behavior may be.

Discussion of TextRank

The prior experiments featured TextRank with a co-occurrence window set to two. This setting was empirically determined by Michalcea providing best results applied to abstracts from scientific journal papers. But the characteristics of our documents and its segments are different, so there is probably a setting which will provide better results. Therefore we evaluated TextRank with segmentation by sentence using over 800 different combinations of co-occurrence windows and segmentations. Table 4.13 shows the best and worst results: Although it was possible to find a setting to exceed the highest effectiveness of TextRank, we found using a window size of two, Tf.idf could not be outperformed and stays the best ranking method.

Candidates	Eff.	P	R	F	TR-Window	#Sents/Seg	Aggregation
Nouns	Worst	10.82	11.05	9.89	2	200	TopN-S
	Best	14.95	14.35	14.80	7	10	TopN-P
Noun-Phrase Chunks	Worst	10.67	11.19	9.79	12	3	TopN-P
	Best	16.71	16.68	16.93	3	20	Max
Named Entities	Worst	24.06	21.41	21	30	3	Max
	Best	27.73	25.62	24.69	12	40	Sum

Table 4.13: Best and worst results per candidate class for TextRank with segmentation per sentence

This results are inconvenient, since we hoped to outperform metrics like Tf.idf with more advanced methods for keyphrase extraction like TextRank.

To explain the low effectiveness of TextRank, we compared our results to Zesch and Gurevych's experiments with a noun- and adjective-based TextRank on different datasets [ZG09]: TextRank shows a variance in effectiveness using different document collections. The best results are achieved using a collection of short scientific documents, while longer documents lead to strongly decreased effectiveness. We have the possibility to diminish the size of documents by applying segmentation, so assuming the segmentation is sound, long documents should be no problem. But apart from document size the collections differ in *token/keyphrase ratio*, meaning some collections have a rather high frequency of keyphrases per document length compared to others. This can also be an explanation for the weak effectiveness, which we will examine using the numbers in table 4.14:

Zesch and Gurevych measure effectiveness in *R-precision* (R-p), which is defined as the precision extracting the same number of keyphrases as in the gold set. In our system this is analogous to evaluation with the threshold based on the index size (T_{doc}) and precision value. We notice that both R-precision and ratio for our dataset lie between the corresponding values in the DUC and SP datasets. To test if there is a correlation between R-p and keyphrase/token ratio we decide to calculate the Spearman-rank- and Pearson-correlation coefficients: Its easy to see that the Spearman rank-correlation is 1. We calculate the Pearson linear correlation with 0.921, which is a is significantly high value as well. This means, that there is a strong correlation between token/keyphrase ratio and effectiveness – which implies considerable low effectiveness, when applying TextRank for a collection like our gold standard.

Dataset	∅#Tokens	∅#Keyphrases	∅ratio	R-p$_{ex}$	R-p$_{ap}$
Inspec	138.6	9.84	0.07	31	36
DUC	902.8	8.08	0.008	21	23
SP	8491.6	8.31	0.0009	4	10
CM07	147236.2	464.52	0.003	-	14.7

Table 4.14: Numbers partly taken from [ZG09]. ∅ratio is the keyphrase/token ratio based on the averaged number of tokens and the averaged number of keyphrases. R-p$_{ex}$ applies exact and R-p$_{ap}$ applies approximate matching. The last line displays the properties for our dataset.

In the end it seems, that TextRank emerges to be not an appropriate method to deal with documents in the context of back-of-the-book indexing.

Effects of Discourse Segmentation

We applied various configurations of Freddy Choi's [Cho00] implementation of TextTiling [Hea94] as a paradigmatic discourse segmentation algorithm.[8] The results based on the Tf.idf ranker are listed in table 4.15, with ascending sorting by F-measure.

Nouns				**Noun-Phrase Chunks**				**Named Entities**			
F	**Step**	**Wind.**	**Agg.**	**F**	**Step**	**Wind.**	**Agg.**	**F**	**Step**	**Wind.**	**Agg.**
18.381	10	100	Max	17.488	10	100	Max	21.729	010	100	Max
18.551	50	250	Sum	19.424	50	250	Sum	22.290	020	120	Max
18.621	150	110	Sum	19.474	20	120	Max	22.814	020	240	Max
19.317	20	240	Sum	19.483	20	240	Max	23.336	050	250	Max
19.559	20	120	Max	19.723	150	110	Sum	23.801	150	110	Max
19.588	20	240	Max	19.825	20	240	Sum	24.742	010	100	Sum
19.618	50	250	Max	19.876	50	250	Max	24.907	020	120	Sum
19.980	150	110	Max	20.154	150	110	Max	24.998	020	240	Sum
19.991	20	120	Sum	20.242	20	120	Sum	25.234	050	250	Sum
20.329	10	100	Sum	20.342	10	100	Sum	25.251	150	110	Sum
20.498	Sent. (n=15)		Max	**20.564**	Sent. (n=25)		Max	**25.636**	Sym. (sd=2)		Max

Table 4.15: Best results of TextTiling compared to baseline. Tf.idf ranking.

To our surprise it was not possible for TextTiling to outperform the segmentation-by-sentence partitioning. For the noun-based candidates, using TextTiling is better then keeping the document unsegmented, but fails to exceed the baseline. We performed the experiment with TextRank as well, to verify if a semantically oriented segmenter works better with a sophisticated keyphrase extraction method. The results tabulated in 4.16 show, that TextTiling performs slightly better with noun-phrase chunks and named entities, but is inferior to a sentence-by-segment partitioning with $n = 10$ using nouns.

The differences are marginal, so the results are not really convincing either – the reason why TextTiling could not improve effectiveness significantly is surprising. We had expected, that the partitioning by TextTiling will create segments of such size, that the ranking of index entries by local maximum, combined with the maximum aggregation method, will be improved, regardless what ranker is used. But this does not seem to happen.

[8] The parameter-names differ from Hearst' paper: *step* is length of a step within a block (in tokens), and *window* means the length of a block (in tokens).

Candidates	Eff.	P	R	F	Method	Aggregation
Nouns	Base	**14.88**	**14.37**	**13.3**	Sent. n=10	TopN-P
	TextTile	14.61	13.91	12.93	s=25 w=100	TopN-P
Noun-Phrase Chunks	Base	16.4	16.17	14.72	Sent. n=15	Max
	TextTile	**16.67**	**16.53**	**14.97**	s=150 w=110	Max
Named Entities	Base	27.59	**25.7**	24.61	Sym. sd=2	Max
	TextTile	**27.62**	25.66	**24.65**	s=50, w=100	Sum

Table 4.16: Best TextTiling & TextRank results compared to best results of other segmenters. TextRank was configured with window size 4.

Discussion of Aggregation Methods

Considering all results, then all aggregation methods seem to participate in one of the best results. When limiting the results to experiments with the top-performing segmentation-by-sentence partioning, the interpretation becomes more definite: Nearly all configurations in table 4.11 achieve the best result with maximum aggregation, capturing only local maxima of score.

Sum aggregation leads in one case to the highest effectiveness with a small improvement over the baseline. As visible in the graph, this is not always the fact – in nearly all cases involving the noun-based candidate sets, the sum aggregation is significantly, or at least slightly, inferior. While other methods increase with segmentation degree, the gradient of sum aggregation can be flat, thus showing no positive effects of segmentation: Since rankers like Tf.idf are frequency-based, the result of summing the local frequency values in the document segments may not be affected much by a specific segmentation.

It is difficult to conclude on the general performance of the top-N-aggregation methods. We still think their underlying theoretical concept holds, but maximum aggregation outperformed top-N using the Tf.idf metric, and top-N could only achieve good results in conjunction with ranking- or segmentation-methods with relatively poor performance.

Since the concept of accumulating global information could not be proven wrong in general, we do not want to suggest to drop sum aggregation completely. There may be ranking methods which work best with the global accumulation from all segments. But considering the prior results, we can conclude that in conjunction with frequency-based rankers, maximum aggregation with appropriately large segment sizes is more predictable and works considerably well.

4.3.3 Comparison with Other Systems

We compare our best results with the unsupervised system of Csomai and Michalcea [CM07]. As mentioned in section 4.2, it is the only system providing a publicly available dataset with gold standard to date.

This system is structurally very similar to ours (see also section 2.2.1). It works on candidate sets including n-grams and named entities, and applies Tf.idf and a combined metric of χ^2-phraseness and χ^2-informativeness for ranking. Phraseness represents the degree to which the candidate can be considered as a grammatically valid phrase, while informativeness refers the degree to which the phrase is representative for the document. The χ^2-independence-test targets to rank candidate phrases by this properties. As a last step, a number of ranked candidates is extracted, which is based on the ratio of gold entries and extracted entries.

The results can be compared easily, since the dataset is the same. Quite difficult to anticipate is the effect of the paraphrase recognition system employed by the authors. The identification of synonyms and morpho-syntactic variations is supposed to have a stronger effect then the approximative matching, which only aims on minor spelling variations and recognition of plural. Unfortunately there is no comparison between an matching with and without the paraphrase recognizer to estimate the effects.

The calculation of F-measure in this work, as specified in section 4.1.1, differs from the formula Csomai and Michalcea use. They calculate F-measure as the harmonic mean of the averaged precision and recall. What averaging they apply is not exactly stated, but we expect macro-averaging. We will abbreviate this measure with the symbol $F_{P,R}$ to stay consistent in notation.

Csomai and Michalcea use the candidate types *n-gram* and *named entity*, with a named entity recognition based on the LingPipe toolkit[9]. As was already shown, named entities without any ranking prove to be very good index entries, since proper names are frequently indexed in books and the gold standard is biased towards short noun phrases and proper names. So it is no surprise, that the two systems with best results in all metrics are based on named entities (see table 4.17).

	Method	**P**	**R**	**$F_{P,R}$**	**F**
[CM07]	N-grams, χ^2-Inf, χ^2-Phrase	20.65	21.25	20.95	-
	NE, Tf.idf	25.95	25.27	25.60	-
	NE, χ^2-Inf, χ^2-Phrase	26.70	26.37	26.53	-
This work	Nouns, Tf.idf, Sent. (n=15)	23.81	21.59	22.65	20.5
	Noun-phrase chunks, Tf.idf, Sent. (n=25)	23.98	21.78	22.83	20.56
	NE , Tf.idf, Sym. (sd=2)	**28.91**	**26.94**	**27.89**	25.64

Table 4.17: Comparison with the unsupervised system of Csomai and Michalcea [CM07]

Csomai and Michalcea's best system is measuring a $F_{P,R}$ of 26.53. Mainly due a advantage in precision of our system, we are able to exceed this value with a $F_{P,R}$ of 27.89. It is important to admit, that the quality of the candidate set plays an important rule in the final effectiveness of the system. When we compare the named entity recognition systems alone (see table 4.18), then it is easy to see that the recognizer we use is superior. It is quite possible, that Csomai and Michalcea would outperform our system, if they would use it in theirs.

System	**P**	**R**	**$F_{P,R}$**
LingPipe	15.79	39.10	22.49
Stanford CRF [FGM05]	19.84	38.99	26.3

Table 4.18: Comparison of different named entity recognition systems based on gold standard

Assuming that a χ^2-independence-based ranking might be superior to Tf.idf, it is interesting to see that the two noun-centered candidate sets with Tf.idf and segmentation show a higher effectiveness compared to n-grams with χ^2-informativeness and -phraseness. Considering the simplicity of the approach just involving chunking (or POS-tagging), Tf.idf and basic segmentation in n sentences, the result is respectable. Although it has no high priority in this context, we expect a better computational performance, since the candidate set is significantly smaller, and the calculation of Tf.idf with maximum aggregation is considerably cheap compared to the more complex statistical tests on huge numbers of n-grams.

[9] http://alias-i.com/

5 Implementation

It was required, that the system is written in Java using the UIMA-framework[1]. We will quickly introduce UIMA and continue with a overview over the implemented components.

5.1 UIMA

Our back-of-the-book indexing system was developed funding on the Apache UIMA framework [GS10]. UIMA stands for *Unstructured Information Management Application* and aims at applications for analysis and information discovery from large amounts of unstructured multi-modal data.

As depicted in figure 5.1, UIMA builds on a pipelined concept of organization data processing. Documents are read from disk and augmented subsequently with annotations which represent metadata resulting from analysis. The container that is passed in the pipeline, holding the document and the associated analysis results, is called *Common Analysis Structure* (CAS).

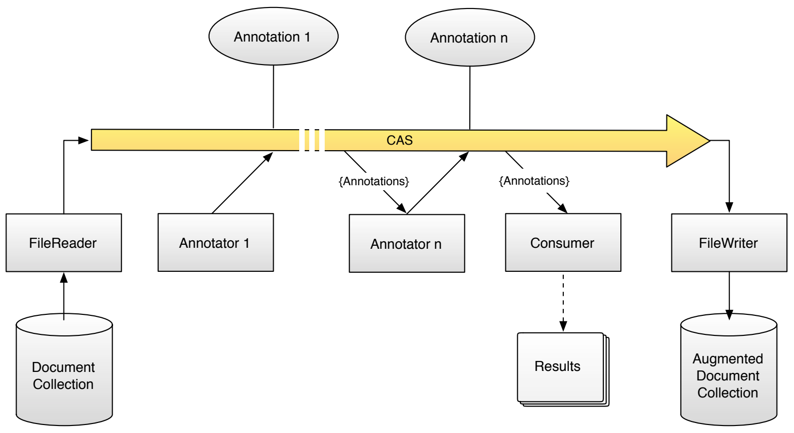


Figure 5.1: Generalized pipeline consisting of n Annotators and a consumer that produces a result. A document is read from the collection using a file reader and represented as CAS. The CAS is augmented with annotations such that previous annotations may be used for calculations by subsequent annotators. The augmented CAS may be serialized to disk using a file writer at the end of the pipeline.

Many of the used components are provided in the *Darmstadt Knowledge Processing Repository* (DKPro) [GMM^{+}07][2]. Next to these publicly available components the UKP Lab[3] supplied additional components which are still at research status. For convenience we will refer to all components associated with the UKP Lab as DKPro.

[1] http://uima.apache.org
[2] http://www.ukp.tu-darmstadt.de/research/projects/dkpro
[3] http://www.ukp.tu-darmstadt.de/people

Additional helpful functionality is provided by uimaFIT[4], which greatly simplifies the configuration of annotators in UIMA and eases testing [OB09]. The Apache Commons Project [5] supplies robust utility methods that are frequently used as well.

All other used libraries and implementations are mentioned in the text below.

5.2 Pipelines

A generic experiment, starting from a document collection and ending with a evaluation result, is divided in four distinct pipeline processing steps, as shown in figure 5.2. The documents are preprocessed (1), annotated with candidate annotations (2), segmented and ranked (3), and finally evaluated against a gold standard (4). The results of the steps 1-3 are augmented CAS-collections which are stored to disk as Xmi-representations. The evaluation results are stored in the file system as plain text.

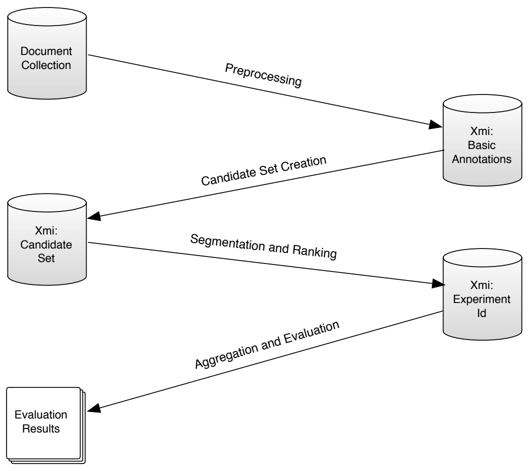


Figure 5.2: Sequence of pipelines representing the processing from a document to the evaluation result.

5.2.1 Preprocessing

The preprocessing pipeline subsumes all basic processing steps that are needed in subsequent stages. It includes tokenization, sentence splitting, lemmatization and POS-tagging. All used components are provided in DKPro, which includes a wrapper for TreeTagger [Sch95], responsible for the lexical and morphological analysis needed for lemmatization and tagging. The augmented CAS is stored as Xmi-representation for later reuse.

5.2.2 Candidate Set Creation

We created nine candidate sets, not including subtle tunings like f.e. applied to the *n*-gram-set. Figure 5.4 shows the generic setup and figure 5.5 the setup involving a wrapper annotator for the Stanford Named Entity Recognizer.

[4] `http://code.google.com/p/uimafit`
[5] `http://commons.apache.org`

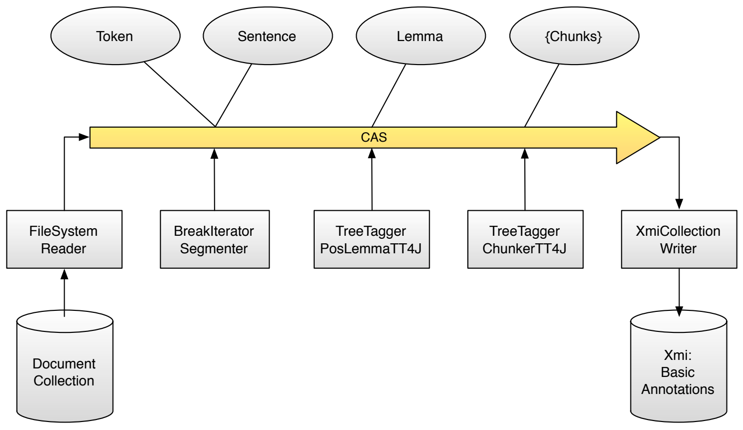


Figure 5.3: Pipeline for creation of basic annotations

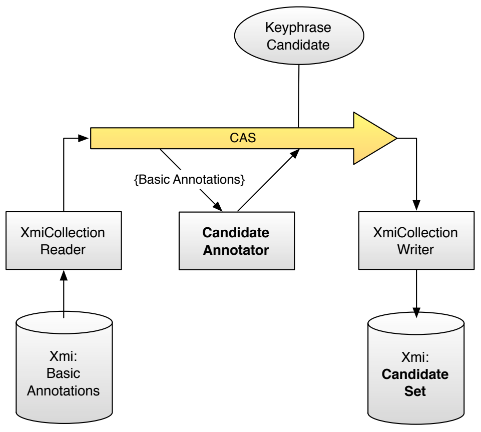


Figure 5.4: Pipeline for candidate annotation. Generic components in bold.

Most of the components could be taken from DKPro, we implemented the HeuristicNamedEntityAnnotator as specified in section 3.2.1, and a new NGramAnnotator which allows exacter filtering of tokens that are assembled into *n*-grams.

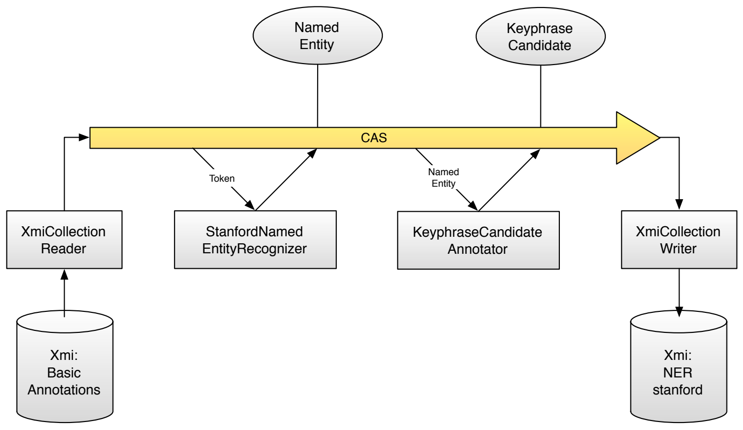


Figure 5.5: Pipeline for creation of a candidate set using the StanfordNamedEntityRecognizer

5.2.3 Segmentation and Ranking

The implementation concept of this stage is to enable a partitioning of the augmented document into its segments, in such way, that these segments can be processed as if they where conventional documents. This allows to work with standardized annotators and consumers, without wrapping their implementations into special segment-handling components. DKPro offers a component called `SegmentMultiplier`, which allows the creation of new CASes for every segment in an encountered CAS. These sub-CASes can be processed by a segment-processing sub-pipeline, after which the CASes are de-multiplied back into one CAS. This CAS contains all data created by the system associated to a concrete experiment and is serialized as Xmi to disk.

Figure 5.7 shows the concrete setup of the multiplication-pipeline. It includes a generic segmenter and a generic ranker which have to be substituted with a concrete segmenters and rankers specified in sections 3.2.2 and 3.2.3. The purpose of the sub-pipelines is to create `Keyphrase`[6]-annotations from keyphrase candidates, which store the score created by the ranker in this pipeline. Figure 5.6 shows the configuration of annotators that form TextRank.

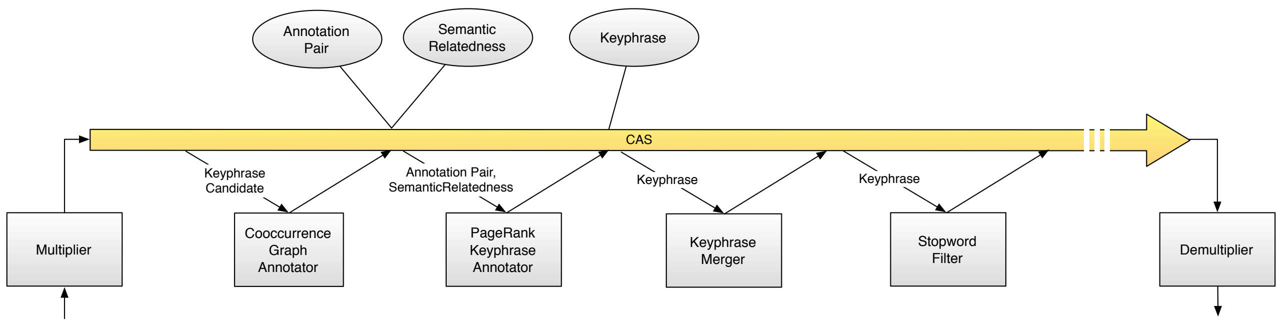


Figure 5.6: Segment-processing sub-pipeline forming TextRank. A CAS containing KeyphraseCandidate-annotations is received from the Multiplier. A weighted co-occurrence graph is build represented by AnnotationPairs with weights modeled by SemanticRelatedness-annotations. The PageRankKeyphraseAnnotator consumes the previous annotations and creates Keyphrases with scores set to PageRank. Subsequent keyphrases are merged up to a length of eight and trailing stop words are filtered, before the pipeline continues.

The technique of storing the intermediate experiment results before evaluation has the benefit of changing evaluation methods afterwards creating the results more quickly. A big drawback is the huge amount of disk space that is taken by the Xmi-representations. But nowadays disk space is no real problem, so we think this procedure is reasonable.

Segmenters

All segmenters needed to conduct the segmentation experiments where implemented by us, except of the `JTextTileSegmenter`, a wrapper of Freddy Choi's TextTiling implementation [Cho00] which is included in DKPro.

[6] Since DKPro uses many annotators that are expecting annotations of the `Keyphrase`-type, we have submitted to this and model index entries as keyphrases. Since keyphrases are phrases with an associated score, this is conceptually flawless and the introduction of an additional type is not necessary.

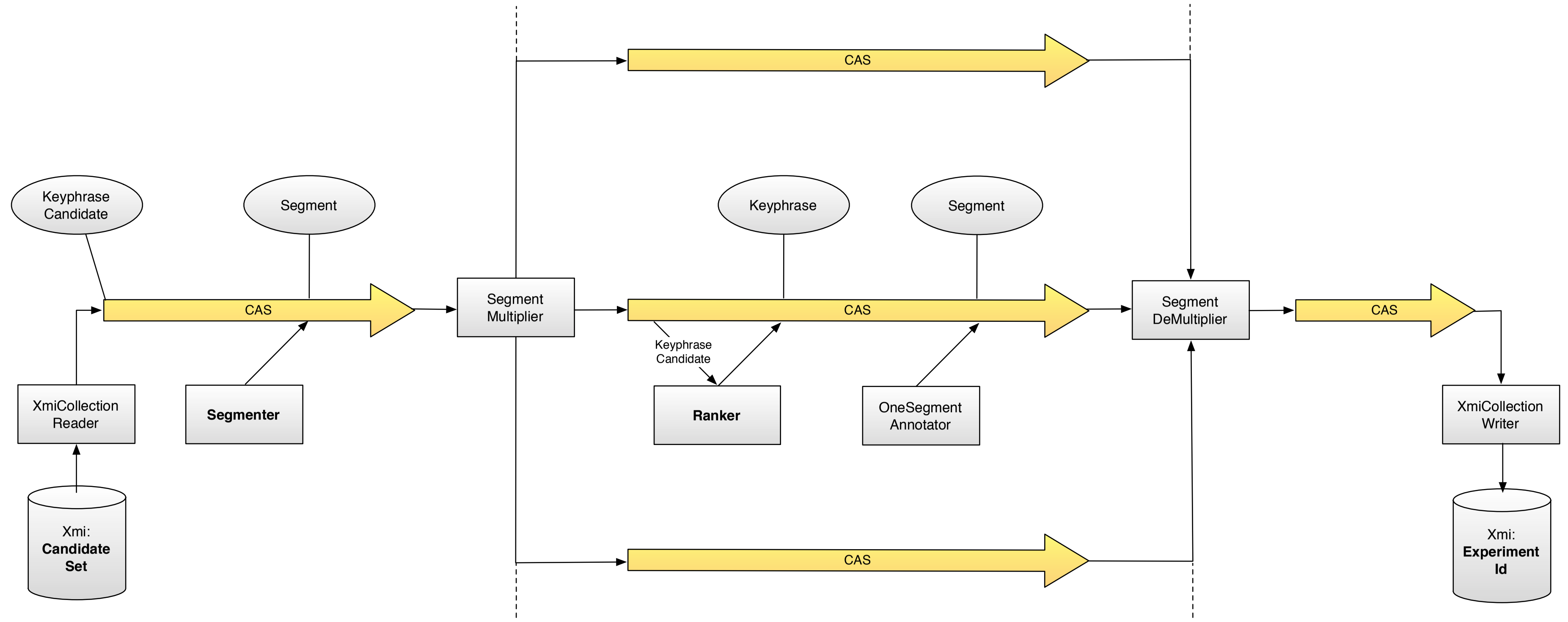


Figure 5.7: Multi-segment pipeline with ranking, generic components in bold. A document is read from disk containing Keyphrase-Candidate-annotations. The CAS is augmented with segment annotations and passed to a multiplier. This component creates new CASes for every segment, which contain all annotations covered by the segment annotation. Each of this sub-CASes is processed by a segment processing step consisting of a ranker and a OneSegmentAnnotator. The ranker adds Keyphrase annotations, which contain the score for the corresponding candidate. Since all Segment annotations get lost in multiplication, the OneSegmentAnnotator has to add a new Segment annotation spanning the range of the sub-CAS. The document is passed to the DeMultiplier who merges all sub-CASes into one CAS. This CAS containing the former document with added Keyphrase and Segment annotations is stored to disk with an Id that represents the configuration of candidate set, segmenter and ranker.

Rankers

The components needed for TextRank are taken from DKPro, the Annotator needed to apply ranking by Tf.idf is implemented by us. The existing `TfidfAnnotator` had problems handling the large documents in conjunction with huge candidate sets. Our new version solves this problems and provides an alternate look-up routine for occurrences of keyphrases in the documents, which searches in the text directly (cached to increase performance). `Web1tAnnotator` implemented by us is a facade for a slightly modified implementation of an API for the Web1 T corpus called jWeb1T[7] provided by Claudio Guliano. This implementation is based on binary search and returns the frequency counts in logarithmic time. The full uncompressed Web 1T corpus has to be installed on the machine which takes approx. 90 GB of space. It was recently shown, that a frequency lookup for this corpus can be implemented much more efficiently by reducing the the size to approx. 8.76 GB with constant look-up time [GH10]. For our purposes, jWeb1T worked just fine.

5.2.4 Aggregation and Evaluation

In the last stage, all Xmi-collections representing an back-of-the-book indexing experiment are consumed by an AggregatingPhraseEvaluator. This evaluator expects CASes containing keyphrases and Segment-annotations and needs to access the gold index collection consisting of plain text files to retrieve the gold index entries for evaluation. Experiment results are stored to the file system. The evaluator is configurable, which means that it can be setup with different matching- and aggregation-methods. In practice we used a single pipeline consisting of four cascaded evaluators configured with the four aggregation methods to avoid reloading Xmi-collections for every aggregation method. The pipeline further contains a component, which logs statistical data about the segment counts and dimensions during collection processing.

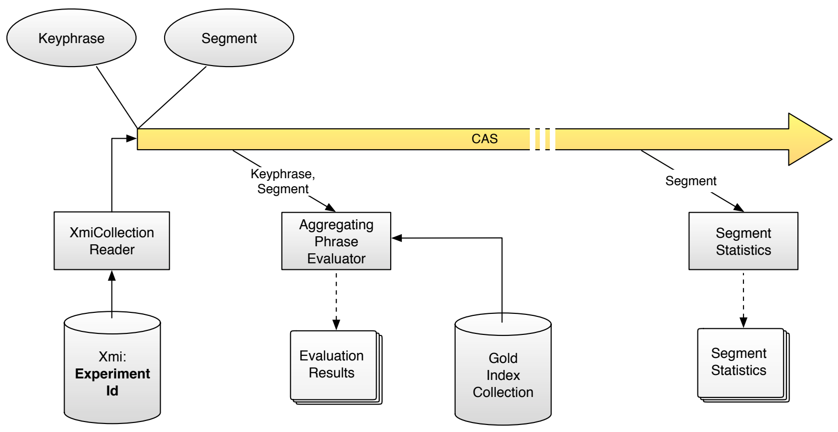


Figure 5.8: Excerpt of the evaluation pipeline showing an AggregatingPhraseEvaluator and the SegmentStatistics-component. The implemented pipeline contains four cascaded AggregatingPhraseEvaluators configured with different aggregation methods.

For the implementation of the aggregation methods we used a data structure modeling a table from the Guava Libraries[8]

[7] http://hlt.fbk.eu/en/node/81
[8] http://code.google.com/p/guava-libraries/

6 Summary

The contributions of this thesis were threefold:

- We gave an **exhaustive overview** over the existing approaches in automatic back-of-the-book indexing. The presented systems include methods from different areas of natural language processing including syntactic analysis, collocation discovery, keyphrase extraction, paraphrase recognition, named entity recognition, word sense disambiguation, semantic relatedness measures and discourse comprehension.
- Looking for methods, that let us evaluate our own automatic-indexing approach, we discovered a **gold standard**, which played a central role in design and evaluation of the system. In this work we analyzed it thoroughly and made the following findings:
  - The **inversion algorithm** used to reverse the inverted index entries shows some flaws and inconsistencies.
  - The coarse-grained index created without inversion of index entries displays strong **bias towards short noun-phrases**. The question arises, whether it does sufficiently resemble back-of-the-book index created by humans.
- We implemented an unsupervised back-of-the-book indexing system involving text segmentation. It submits to the common **extractive approach**, meaning the aggregation of small original text fragments instead of the generation of new potential index phrases. The system extracts a set of candidate phrases from the text and partitions the document into segments. The candidates in the segments are associated with a score using a ranker and aggregated into a set of locally ranked phrases in the end. A subset constitutes the final set of index entries. We experimented with different candidate sets, having these insights:
  - State-of-the-art **named entities** are the best set of candidates reaching the F-measure of followed by named entities created by a simple heuristic.
  - **Noun-based candidates** like noun-phrase chunks and noun-tokens seem to be a good trade-off between candidate-set size and high recall.

  We experimented with different text segmenters, which led to these conclusions:

  - Segmentation **improves** a back-of-the-book indexing system when it is using noun-based candidates.
  - Segmentation affects a system only marginally, when it is based on **named entities**, it can slightly improve the effectiveness, but will most likely impair it.
  - **Aggregation** of locally maximal candidates usually works best, assuming the segment is sized appropriately.
  - To our surprise, **discourse segmentation** implemented by TextTiling does not perform better then simple segmentation by sentence.

  We have tried three rankers, including a ranker utilizing the Web 1T corpus consisting of *n*-grams derived from Google web searches. The insights were:

  - Tf.idf is the **best ranking method**, followed by Web1T and TextRank. This is true for configurations with segmentation or without, and for all candidate sets.

- **Web1T** may be biased towards modern language, since the gold standard comprises books from the last centuries.
- **TextRank** shows surprisingly inferior performance. It is positively affected by segmentation but is not able to excel the baseline of Tf.idf. It may generally work bad on datasets with a low keyphrase density.

We could compare our system with a unsupervised system, which was evaluated using the same gold standard. We were able to **exceed the existing benchmark** for this gold standard, using a combination of named entity recognition, Tf.idf and simple segmentation. This success is mostly due our more advanced named entity recognition. Nevertheless, the configuration using nouns and noun-phrase chunks could also achieve good results, outperforming the existing benchmark for *n*-grams.

Future Work

Since the experiences with TextRank were not satisfactory, it should be tried if a different keyphrase extraction method could excel the baseline created by Tf.idf. Since the dataset is too small for supervised systems the keyphrase extraction system has to be unsupervised.

A partitioning into pseudo-sequences of *n* tokens should have been tried, instead of our segmentation-by-sentence approach. It is difficult to estimate the impact of variance in the segment sizes, so segments with the same size would have been methodically cleaner.

We would not opt for trying more segmentation methods, before examining the possibilities of exploiting phrase sequences as introduced in section 4.3.2. Since segmentation combined with the aggregation of local maxima is optimal when a phrase sequence of true positives is fully contained in the segment, it should be tried to recognize these sequences directly.

It would be very interesting to see, if the extracted index entries from the different candidate sets could be combined to create a better index set. This could by tried by weighted intersections or union operations on the sets.

We think the quality of the initial candidate set is crucial. Seeing, how basic heuristics, as implemented in the heuristic named entity recognizer, can reduce the number of false positives should motivate for searching for other ones.

Web1T was implemented by ourselves for this thesis, so we have not much experience with its performance on other tasks. We were not satisfied with its effectiveness, it could be tried to adjust the implementation and/or to verify if the bad performance was due the bias towards modern language.

TextTiling was performing inferior to segmentation-by-segment. This could be correlated with the increased variance of segment sizes.

We quickly implemented a segmentation-by-paragraph. This worked so bad that we even haven't mentioned it in the evaluation. But we don't opt for a retry before investigating in phrase sequences.

Further examination, if TextRank is sensitive to keyphrase density could be interesting.

When the task of automatic back-of-the-book indexing shall be taken seriously, then it is necessary to create a new gold standard. We think, the standard available is not sufficient, as outlined in the thesis.

A List of Books contained in the Gold Standard

```
A Catechism of Familiar Things
Bell's Cathedrals- Chichester
Bismarck and the Foundation of the German Empire
Bramble-Bees and Others
Different Forms of Flowers
English Literature
Experimental Researches in Electricity, vol1
Fabre, Poet of Science by Legros
More Hunting Wasps
Pathology of Lying
Peace Theories and the Balkan War
Sequential Problem Solving
Socialism and American ideals
Stonewall Jackson And The American Civil War
Studies in the Psychology of Sex Volume 1
Studies in the Psychology of Sex Volume 2
Studies in the Psychology of Sex Volume 3
Studies in the Psychology of Sex Volume 4
Studies in the Psychology of Sex Volume 5
Studies in the Psychology of Sex Volume 6
The Effects of Cross & Self-Fertilisation in the Vegetable Kingdom
The Evolution of Man V2
The Explorers of Australia and their
The Journal of Sir Walter Scott
The Sources and Analogues of A Midsummer-night's Dream
The Wonders of Instinct
The Worst Journey in the World - Antarctic
Theory Of Silk Weaving
Winter Sunshine
```

List of Figures

List of Tables

Bibliography

[AH06] S. Aubin and T. Hamon. Improving term extraction with terminological resources. *Advances in Natural Language Processing*, pages 380–387, 2006.

[AN06] T. Aït El Mekki and A. Nazarenko. Using NLP to build the hypertextual network of a back-of-the-book index. *ArXiv Computer Science e-prints*, September 2006.

[BF06] T. Brants and A. Franz. Web 1T 5-gram Version I, 2006.

[BMSW97] D. M. Bikel, S. Miller, R. Schwartz, and R. Weischedel. Nymble: a high-performance learning name-finder. In *Proceedings of the fifth conference on Applied natural language processing*, pages 194–201, Morristown, NJ, USA, 1997. Association for Computational Linguistics.

[Bor70] H. Borko. Experiments in book indexing by computer. *Information storage and retrieval*, 6(1):5–16, 1970.

[BP98] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. *Computer networks and ISDN systems*, 30(1-7):107–117, 1998.

[Cho00] F. Y. Y. Choi. Advances in domain independent linear text segmentation. In *Proceedings of the 1st North American chapter of the Association for Computational Linguistics conference*, pages 26–33, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[CM06] A. Csomai and R. Mihalcea. Creating a Testbed for the Evaluation of Automatically Generated Back-of-the-book Indexes. *Computational Linguistics and Intelligent Text Processing*, pages 429–440, 2006.

[CM07] A. Csomai and R. Mihalcea. Investigations in unsupervised back-of-the-book indexing. *Proceedings of the Florida Artificial Intelligence Research Society*, pages 211–216, 2007.

[CM08] A. Csomai and R. Mihalcea. Linguistically motivated features for enhanced back-of-the-book indexing. In *Proc. Ann. Conf. of the Association for Computational Linguistics, ACL/HLT*, volume 8, pages 932–940, 2008.

[DDF$^+$90] S. Deerwester, S. Dumais, G. Furnas, T. Landauer, and R. Harshman. Indexing by latent semantic analysis. *Journal of the American society for information science*, 41(6):391–407, 1990.

[Dil82] M. Dillon. Thesaurus-based automatic book indexing. *Information Processing & Management*, 18(4):167–178, 1982.

[DL83] M. Dillon and K. LAURA. Fully automatic book indexing. *Journal of documentation*, 39(3):135–154, 1983.

[FGM05] J. Finkel, T. Grenager, and C. Manning. Incorporating non-local information into information extraction systems by gibbs sampling. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 363–370. Association for Computational Linguistics, 2005.

[G$^+$95] J. Grossman et al. *The Chicago manual of style*. Univ. of Chicago Press, 1995.

[GH10] D. Guthrie and M. Hepple. Storing the Web in Memory: Space Efficient Language Models with Constant Time Retrieval. *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, 2010.

[GMM$^+$07] I. Gurevych, M. Mühlhäuser, C. Müller, J. Steimle, M. Weimer, and T. Z. and. Darmstadt knowledge processing repository based on uima. In *Proceedings of the First Workshop on Unstructured Information Management Architecture at Biannual Conference of the Society for Computational Linguistics and Language Technology*, page electronic proceedings, Tübingen, Germany, 2007.

[GS10] T. Gotz and O. Suhre. Design and implementation of the UIMA Common Analysis System. *IBM Systems Journal*, 43(3):476–489, 2010.

[Hea93] M. Hearst. TextTiling: A quantitative approach to discourse segmentation. 1993.

[Hea94] M. A. Hearst. Multi-paragraph segmentation of expository text. In *Proceedings of the 32nd annual meeting on Association for Computational Linguistics*, pages 9–16, Morristown, NJ, USA, 1994. Association for Computational Linguistics.

[Hea97] M. A. Hearst. Texttiling: segmenting text into multi-paragraph subtopic passages. *Comput. Linguist.*, 23(1):33–64, 1997.

[JK95] J. Justeson and S. Katz. Technical terminology: some linguistic properties and an algorithm for identification in text. *Natural language engineering*, 1(01):9–27, 1995.

[JM08] D. Jurafsky and J. H. Martin. *Speech and Language Processing (2nd Edition) (Prentice Hall Series in Artificial Intelligence)*. Prentice Hall, 2 edition, 2008.

[Juo05] P. Juola. Towards an automatic index generation tool. *In Abstracts of ACH/ALLC*, pages 102–04, 2005.

[Kin98] W. Kintsch. *Comprehension: A paradigm for cognition*. Cambridge Univ Pr, 1998.

[Luk06] S. Lukon. A machine-aided approach to intelligent index generation: Using natural language processing and latent semantic analysis to determine the contexts and relationships among words in a corpus. Master's thesis, Duquesne University, 2006.

[Mil95] G. Miller. WordNet: a lexical database for English. *Communications of the ACM*, 38(11):39–41, 1995.

[MRS08] C. Manning, P. Raghavan, and H. Schtze. *Introduction to Information Retrieval*. Cambridge University Press New York, NY, USA, 2008.

[MT04] R. Mihalcea and P. Tarau. TextRank: Bringing order into texts. In *Proceedings of EMNLP*, pages 404–411. Barcelona: ACL, 2004.

[Mul05] N. Mulvany. *Indexing books*. University of Chicago press, 2005.

[NBB03] M. Neff, R. Byrd, and B. Boguraev. The Talent system: TEXTRACT architecture and data model. In *Proceedings of the HLT-NAACL 2003 workshop on Software engineering and architecture of language technology systems-Volume 8*, pages 1–8. Association for Computational Linguistics, 2003.

[OB09] P. Ogren and S. Bethard. Building test suites for UIMA components. In *Proceedings of the Workshop on Software Engineering, Testing, and Quality Assurance for Natural Language Processing (SETQA-NLP 2009)*, pages 1–4, Boulder, Colorado, June 2009. Association for Computational Linguistics.

[OD08] D. Olson and D. Delen. *Advanced data mining techniques*. Springer Verlag, 2008.

[PBB02] Y. Park, R. Byrd, and B. Boguraev. Automatic glossary extraction: beyond terminology identification. In *Proceedings of the 19th international conference on Computational linguistics-Volume 1*, pages 1–7. Association for Computational Linguistics, 2002.

[PS06] P. Pecina and P. Schlesinger. Combining association measures for collocation extraction. In *Proceedings of the COLING/ACL on Main conference poster sessions*, pages 651–658. Association for Computational Linguistics, 2006.

[RG00] P. Rayson and R. Garside. Comparing corpora using frequency profiling. In *Proceedings of the workshop on Comparing corpora*, pages 1–6, Morristown, NJ, USA, 2000. Association for Computational Linguistics.

[RLJ10] K. Reinholt, S. Lukon, and P. Juola. The role of visual exploration in machine-aided back-of-the-book index generation. Poster at Chicago Colloquium on Digital Humanities and Computer Science, November 2010.

[Sal88] G. Salton. Syntactic approaches to automatic book indexing. In *Proceedings of the 26th annual meeting on Association for Computational Linguistics*, pages 204–210, Morristown, NJ, USA, 1988. Association for Computational Linguistics.

[SB88] G. Salton and C. Buckley. Term-weighting approaches in automatic text retrieval. In *INFORMATION PROCESSING AND MANAGEMENT*, pages 513–523, 1988.

[Sch95] H. Schmid. TreeTagger—a language independent part-of-speech tagger. *Institut f
"ur Maschinelle Sprachverarbeitung, Universit
"at Stuttgart*, 1995.

[Sta88] H. Stark. What do paragraph markings do? *Discourse processes*, 11(3):275–303, 1988.

[ZEMAN06] H. Zargayouna, T. El Mekki, L. Audibert, and A. Nazarenko. IndDoc: an aid for the back-of-the-book indexer. *The Indexer*, 25(2):122–125, 2006.

[ZG09] T. Zesch and I. Gurevych. Approximate matching for evaluating keyphrase extraction. In *Proceedings of the 7th International Conference on Recent Advances in Natural Language Processing (electronic proceedings)*, pages 484–489, Borovets, Bulgaria, Sep 2009.