Multi Document Summarization
Multi Document Summarization

A N T O N  H Ä G E R S T R A N D

Master’s Thesis in Computer Science (30 ECTS credits)
at the School of Computer Science and Engineering
Royal Institute of Technology year 2011
Supervisor at CSC was Lars Arvestad
Examiner was Stefan Arnborg

TRITA-CSC-E 2011:090
ISRN-KTH/CSC/E--11/090--SE
ISSN-1653-5715

Royal Institute of Technology
School of Computer Science and Communication

KTH CSC
SE-100 44 Stockholm, Sweden
URL: www.kth.se/csc
Abstract

This thesis describes the development and deployment of a search based, automatic, multi-document summarizing system, and shows that technologies regularly used in search engines are applicable to the task of automatic summarization. The summarizer works by first indexing a set of documents as well as extracting sentences from them. The summarizer can then summarize document collections, which are dynamically constructed from the documents in the index.

This is done as an addition to the Solr search server, which provides an interface to the Lucene index; the summarizer uses Lucene to find sentences which it can use to create summaries. The summarizer uses methods similar to search engines, and enjoys the same benefits in speed that modern search engines provide when summarizing documents.
Referat

Sammanfattning av multipla dokument

Den här rapporten beskriver utvecklingen av ett sökbasert system för automatiska sammanfattningar av multipla dokument, och visar att metoder som vanligtvis används i sökmotorer är användbara vid automatisk sammanfattning. Summeraren fungerar genom att först indexera ett antal dokument och extrahera alla meningar ur dessa. Summeraren kan sedan summera dokumentsamlingar som har konstruerats av dokument från indexet.

Detta görs via sökservern Solr, som tillhandahåller ett gränssnitt mot indexet Lucene; summeraren använder sig av Lucene för att hitta meningar som kan användas för att skapa sammanfattningar. Summeraren använder metoder som liknar dem i moderna sökmotorer, metoderna är lika snabba i summeringstillämpningen som vid sökningar.
# Contents

## 1 Introduction
1.1 Background ................................................. 1
1.2 Acknowledgements ........................................... 2
1.3 About the Method ........................................... 2
1.4 Glossary .................................................. 2

## 2 Relevant Summaries
2.1 Measuring Relevance ........................................ 5
  2.1.1 Human Relevance ....................................... 5
  2.1.2 ROUGE ................................................ 6
2.2 What to summarize .......................................... 7
2.3 Sentence Extraction .......................................... 7
  2.3.1 Query-oriented summarization ......................... 8
  2.3.2 Topic Grouping ....................................... 8
2.4 Scoring Function ........................................... 8
  2.4.1 Term relevancy ........................................ 9
  2.4.2 Sentence Positioning .................................. 10
  2.4.3 Avoiding Redundancy ................................ 11
2.5 Sentences to summary ....................................... 12
  2.5.1 References in sentences ............................... 12
  2.5.2 Ordering of sentences ................................ 12

## 3 Solr and Lucene
3.1 Search engines ............................................. 15
  3.1.1 Document representation ............................... 15
  3.1.2 Inverted Index ........................................ 15
  3.1.3 Scoring ............................................... 16
  3.1.4 Document representation ............................... 17
3.2 Using Solr and Lucene ..................................... 17
3.3 Documents and fields ....................................... 18
3.4 Queries in Lucene and Solr ................................. 18
Chapter 1

Introduction

1.1 Background

Summaries of documents have been used for presenting the most relevant information from one or several documents for a long time. This is a task traditionally performed by humans, since it requires understanding of natural language as well as an understanding of what information the readers of the summary are interested in. Humans are however expensive, and any text which is to be summarized must first be read by a human. This is not a big problem for single document summarization, since it is likely that the document has been written by a human; the amount of extra work and time spent on summarizing the document is small compared to the amount of work and time spent on writing it.

Summarizing of static collections of documents is not much more cumbersome than summarizing of single documents, provided that any document is only allowed to be a part of a single collection. If a document is allowed to be a part of multiple collections much more work is involved, since every collection needs a separate summary. If the collections are also allowed to be dynamic, a new summary would be needed for every change. For dynamically generated collections of documents, human summarization becomes infeasible; the time and work needed for summarizing is the same, or almost the same, as reading the documents.

Automatic summarization of multiple documents is therefore of value, if it can be done well enough to provide relevant summaries. Good summaries of documents should contain the most relevant information contained in the documents which were summarized. In this aspect summarization is very similar to searching using search engines, since both rely heavily on extracting relevant information from large amounts of text. There have been several attempts at creating good summarizers of multiple documents, using several different methods and assumptions. This report presents a search based approach to the problem of summarization of multiple documents.

This thesis will try to answer the following question:
“Is it possible to build a multi-document summarizer using a search based approach?”

1.2 Acknowledgements

The job requester of this thesis was Findwise AB, a consultant firm specializing in search driven solutions. My supervisor at Findwise was Simon Stenström and my supervisor at CSC was Lars Arvestad. Without these persons as well as other people at Findwise my work would had been much harder.

1.3 About the Method

In order to see if it is possible to build a summarizer using a search based approach, one was built. The choice of technologies was done after a study of existing information in the field of automatic summarization of multiple documents. The study gave a hint towards what kind of techniques could be used in order to build such a summarizer.

The built summarizer was then tested against a dataset which is well known, and well used, in this field. No larger user tests were done; instead the summaries were tested against other summary systems, using an automatic evaluator. This evaluator is also well known, and widely used.

The choice of method was a simple one. In order to see if one can build a summarizer, one can try to build one. If it works, it is possible. If it is not, one might have to try again.

The choice of evaluation method is a bit less obvious; user testing would be good, but user testing of a summarizing system requires a lot of work. Indeed, if the testing was done rigorous enough, it would be work enough for another thesis. The automatic evaluation method used here has however been widely used in past work in this field, so the choice to use it instead of manual evaluation was easy.

1.4 Glossary

Document A document can be any written coherent text. The content of a document consists of natural language, but documents may also have meta-data such as title, author or the date on which it was written. A document may for example be a book, a scientific report or a web page.

Document Collection A document collection is a collection of documents; no other requirements are put on the collection. A document collection could for example be every book in a library or every page on the internet.

Document Cluster In the field of Information Retrieval, document clustering is a technique for grouping documents which are similar. Since documents in
clusters have something in common, the clusters are relevant to summarize. It is of no interest to summarize documents in a single summary if they are not similar; if this is the case it is more relevant to summarize the documents separately. A document cluster could for example be all books in a library about the country of Switzerland.

**Term** In the field of Information Retrieval, a term is the basic unit to consider when retrieving information. Most terms are words in some kind of lemmatized or stemmed form. Lemmatization and stemming are different methods for removing grammar from terms, in short it works by converting any occurrence of a word, in any form, to a single word form. Terms might also be for example names. Names can of course be split into several words, where the individual words have a different meaning. For example “New York” is different from “New” and “York”.

**N-Gram** Generally, n-grams are a sequence of items of length n. In this thesis the n-grams are always built of terms. N may be any integer, for example 1-gram, 3-gram or 10-gram, and “n-gram” may also be used to describe a sequence of arbitrary length. 1-grams, 2-grams and 3-grams are also called unigrams, bigrams and trigrams.

**Sentence Extraction** Sentence extraction is a method for generating summaries of a document, or document set, by extracting sentences from the original document(s) and using those to generate a summary.

**Summary Relevance** This thesis chooses to define the relevancy of a summary to be the relevance it has for a human reader of the summary, in respect to the content of the documents summarized. This is a rather vague definition, but it is also a rather realistic one.
Chapter 2

Relevant Summaries

The goal of a multi document summarizer is to deliver relevant summaries for a set of documents. For any set of documents, there may be many different views on what a relevant summary should contain. A summarization algorithm will have to be designed in a way so that it is able to produce relevant summaries. However, in order to measure relevance, we need a way to compare summary relevance.

2.1 Measuring Relevance

Measuring relevance is a hard task where a subjective experience should be converted to an objective measure. Since real objectiveness is very hard to achieve, simplifications are made. Either by letting a representative group of people evaluate summaries, or by comparing summaries to summaries which are considered relevant.

In either case, it must be remembered that these measurements are simplifications. Most importantly, optimizing a summarizer for these tests is not the same as optimizing it for summary relevance.

2.1.1 Human Relevance

The concept of relevance can be viewed from different viewpoints. Humans can generally decide whether a summary is valid for a document cluster by reading the summary. Indeed, a summary is relevant to any person if that person decides that it is relevant. The relevance of a summary is specific to the persons information need and to the document cluster currently being summarized [14].

Different people may have different views on the relevancy of a summary of the same document cluster; if they have different information needs they will find different information relevant. This have both positive and negative consequences for development of a summarizer. If a summarizer is aware of for what information need the summary is produced for, it may use this knowledge in order to produce more relevant summaries, which of course is very positive.
CHAPTER 2. RELEVANT SUMMARIES

However, since different users have different user needs, a large number of user tests need to be done in order to evaluate the relevance of summaries produced by a summarizer. Good user tests will therefore require many people to spend time reviewing summaries. User tests are therefore expensive in both time and money [12].

2.1.2 ROUGE

ROUGE is a summary evaluation implementation which evaluates summarizers by comparing the summaries they produce with known good summaries [12]. Good summaries are often attained by letting humans summarize text; this of course requires human effort, but it is sufficient to summarize a document once. This enables the creation of data sets which can be used to evaluate any document summarizer with little human effort. ROUGE has been used to evaluate many of the algorithms for summarization which have been suggested in later years.

ROUGE is based on an evaluation of summary evaluation protocols, and the main measures are different co-occurrences between the summary to be evaluated and a known good summary. There are some variations of ROUGE which measure co-occurrences in different ways:

**ROUGE-N**  ROUGE-N is a measure of n-gram co-occurrences between two summaries. N can be any number, and even unigrams have been shown to be a good measure.

**ROUGE-L**  ROUGE-L is a measure based on the longest common subsequence between sentences of the summaries. A subsequence of a sentence is a sequence of words which occur in the sentence in the same order, but not necessarily without other words in between.

**ROUGE-W**  ROUGE-W is a variation of ROUGE-L with weighting. The weighting is done based on how close the words in the subsequence are to each other in the original sentence: closer is better.

**ROUGE-S**  ROUGE-S is based on the number of co-occurrences of so called skip-bigrams in sentences. Skip bigrams are bigrams where other words may appear between the two words of the bigram. For example, the sentence “Roses are red.” have the skip-bigrams “Roses are”, “Roses red” and “are red”. ROUGE-S may also be specified with a maximum number of words to skip over, for example ROUGE-S4.

**ROUGE-SU**  ROUGE-SU is a variation of ROUGE-S, where sentences’ score also get boosted if they have similar unigrams. For example, the sentence from above (“Roses are red.”) compared to “Red are roses.” with ROUGE-S would get a score of zero, which is as low as any sentence. With ROUGE-SU these two sentences
2.2. WHAT TO SUMMARIZE

would get a higher score than if you for example compared the first one to “Violets are blue”. ROUGE-SU may also be specified with a maximum number of words to skip, for example ROUGE-SU4.

2.2 What to summarize

When summarizing document sets, one needs to keep in mind what kind of document sets are worth to summarize together. Document sets which contains documents which are not related in any way are hard to summarize as a set; one can just as well summarize the individual documents and put the short summaries together.

In this thesis, we look at how documents which have something in common can be summarized. Document sets consisting of documents which have something in common are in this thesis called “Document Clusters”.

The things the documents have in common should be something that makes them worth summarizing together. For example, two novels might have the same author; this does not necessarily make them worth summarizing together, since the novels might be about entirely different stories. On the opposite, a number of news articles written about an event might be worth summarizing together, since they have similar content.

2.3 Sentence Extraction

Summaries of document clusters should contain relevant information from the documents in the document cluster. The summaries produced by a summarizer must also be coherent written text. The coherency can be achieved either by generating text from scratch, which is a hard task, or by using parts of the text in the document cluster. The parts of text used need to be able to stand on their own, for example sentences. This method is called sentence extraction, and it has recently been used in a lot of proposed summarizers.

Summarizers which use sentence extraction must focus on finding relevant sentences to use in a summary, rather than finding relevant information in general.

As shown in figure 2.1, sentence extraction is done in two steps: first sentences

![Figure 2.1. Basic sentence extraction methodology.](image-url)
are extracted from documents in the cluster, then a summary is constructed using these sentences.

Finding sentences to use in relevant summaries is far from an easy task. The sentences used in the summaries produced by the summarizer must together contain the most relevant information contained in the document cluster. Looking at individual sentences within document cluster, a scoring system may set a score on each sentence. This score can be used to calculate which sentences are the most relevant to use in a summary.

Different ways of calculating scores for sentences have been proposed by different researchers. It seems that there are two main criteria for sentences score: how well its content represents the document cluster and that is not too similar to any other sentence used in the summary. The second criterion is of greater importance in summarization of document clusters than it is in summarization of single documents, since a single document is less likely to repeat itself. The sentence scoring can be seen as a function, which we call the sentence scoring function.

2.3.1 Query-oriented summarization

Query-oriented summarization is a kind of summarization applicable when the summarizer does not only know what documents are to be summarized, but also a user query. For example, if a search is done in a search engine on the web the returned documents can be seen as a cluster. When summarizing such a cluster a summarizer can use the user query in order to create a summary which contains information relevant to the query. Summarizers using this technique have been able to deliver good results, of course granted that they get a query [21].

2.3.2 Topic Grouping

Some summarizers use a model assuming that a sentence is written on a specific topic, using the gained knowledge to include the most relevant topics in the summary. The topics are hidden, but knowledge of the probability of a sentence belonging to a specific topic are used in calculating the sentence score.

Topic Grouping can be useful for finding synonyms and dealing with them. Synonyms should have similar probabilities for ending up in topics, and if the synonyms have a high relevancy in the document the topic grouping will grant them an even higher score.

2.4 Scoring Function

Much of the research concerning document summarization has lately been focused on finding a good scoring function for sentences. Many of them rely on term frequency measures, some using the TF-IDF measure described above.

The methods for designing a scoring function vary in their approach. Some rely solely on relatively simple ways of combining term frequency measures, such as the
2.4. SCORING FUNCTION

product, average or sum of the term score [19]. Other approaches represent the text as relations, trying to understand how terms relate to each other grammatically [15]. There have also been suggestions using clustering of sentences, using the cluster density in order to find important sentences[20].

2.4.1 Term relevancy

A possible basis for a sentence scoring function is the relevance of the terms in it. The relevancy of a term is specific for a certain document cluster; terms that are important for a document cluster have a high relevancy in that cluster. For example, the term “Zurich” might be very relevant for a cluster of books about Switzerland, but not so relevant for cluster of books about ancient Chinese culture.

Measuring the relevancy of a term can be done by looking at how often it is used in a document cluster compared to how often it is used in general. In a library, the term “Zurich” will probably occur with high frequency in books about Switzerland, but not with as high frequency generally. This measure of relevance assumes that terms which are highly relevant for a document cluster occurs many times in that cluster compared to how often they occur in general text. This might not be true for every case, but it works well enough for practical usage. For example, many major search engines rely heavily on term frequency measures.

This report includes two different measures using term frequencies, described below.

**TF-IDF**

TF-IDF (Term Frequency - Inverse Document Frequency) is a measure used to see how many times a term occurs in a specific document, compared to how often it occurs in the total number of documents. It will however work as well for looking at how many times a term occur in a document cluster compared to a total document collection.

A variation of the standard TF-IDF measure is a weighted measure, which penalizes multiple occurrences of terms. The full calculation looks like this:

\[
\begin{align*}
\text{For a term } t \text{ and a document } d: \\
\quad w_{t,d} &= w_{t,d} \times idf_t \\
\text{where } \quad w_{t,d} &= \begin{cases} 
1 + \log (tf_{t,d}) & \text{if } tf_{t,d} > 0 \\
0 & \text{else}
\end{cases} \\
\quad idf_t &= \log \frac{N}{df_t}
\end{align*}
\]

In this calculation, \( tf_{t,d} \) is the term frequency in a specific document, i.e. the number occurrences of a term in the document, \( N \) is the total number of documents
and $df_t$ is the document frequency of a term, i.e. the number of documents a term occur in. The total measure is a combination of the sub measures $idf_t$, which is a weighted inverse of the document frequency, and $wf_t$, which is a weighted variant of the term frequency in a document[17].

When several terms are matching a document, the scores from the different terms can be summed up to produce a score for the entire document. This sum is then the similarity of the documents, where higher score means that they are more similar.

**Kullback-Leibler Divergence variations**

*Kullback-Leibler* divergence (KL-divergence) is an asymmetric similarity measure which measures the divergence between two sets of data. The modifications of KL-divergence described below have been used by L. Hennig in [10] to create a well performing summarizer, and the methods are applicable in this thesis. The divergence between two documents, $d$ and $e$, would be computed as follows:

$$KL(d||e) = \sum_{t \in T} tf_{t,d} \log \frac{tf_{t,d}}{tf_{t,e}}$$

(2.4)

$$KL(e||d) = \sum_{t \in T} tf_{t,e} \log \frac{tf_{t,e}}{tf_{t,d}}$$

(2.5)

where $T$ are all terms in either $e$ or $d$.

Here, $KL(d||e)$ is the how much $e$ diverges from $d$, and $KL(e||d)$ is how much $d$ diverges from $e$. As seen, this divergence is not symmetric, $KL(d||e) \neq KL(e||d)$, so it is not a good distance measure. This can be fixed with the so called “symmetric KL-divergence” which simply adds up the KL-Divergences:

$$SymKL(d,e) = KL(d||e) + KL(e||d)$$

(2.6)

Here, $SymKL(d,e)$ is a combination of the divergences between $d$ and $e$; this measure is symmetric, since $SymKL(d,e) = SymKL(e,d)$. However, this measure is a distance measure, where as similarity is what we want to calculate. This can be fixed by normalizing the divergence to be in the range $[0,1]$ and subtracting the normalized divergence from 1. We now have a similarity measure instead of a divergence measure[10].

**2.4.2 Sentence Positioning**

Apart from terms in a sentence, the sentence’s position in the original text can be used to determine if it is an important sentence or not [14]. Sentences in the beginning or the end of a document generally have more relevant information. If
2.4. SCORING FUNCTION

one knows where in a paragraph or subsection of a document the sentence is, one can also use this knowledge in the same manner as for full documents.

Sentence positioning is largely dependent on the type of document to be summarized. For example, a news article often have many relevant sentences in the leading paragraph, while the leading paragraph of a novel might not at all reflect the rest of the content.

2.4.3 Avoiding Redundancy

Having two very similar sentences in the same summary is annoying for the reader; a summary should not contain repetitions of the same information in different words. When summarizing single documents, this comes naturally, since a single document rarely repeats itself. Document clusters might very well contain several almost identical sentences.

It is clear that when multiple sentences are too similar, only one of them should be used in the summary. There are different methods of removing duplicates, for example one could start building the summary, and when adding new sentences heavily penalize sentences which are very similar to the ones already in the summary. [13] Detecting similarity can be done using methods similar to the ones used to calculate sentence scoring, by comparing two sentences instead of a sentence to a collection.

Synonym detection might be a problem here, since it is likely that different documents uses different words for the same thing. Topic grouping from 2.1 might be used to detect synonyms, since synonyms should have a similar topic.

Maximum Marginal Relevance

The Maximum Marginal Relevance (MMR) method is a method which aims to reduce redundancy in summaries [3]. Works by selecting sentences using an unspecified method, such as any of those described earlier, and adding them to the summary. However, it also rescores each sentence as the summary grows. Carbonell and Goldstein describes the method in more detail, but the following formula is a basic variant of it [3].

\[
Score_S = \lambda Sim(S, Q) - (1 - \lambda) Sim(S, Sum)
\]  

Where \(\lambda\) is a weight factor, \(S\) a sentence which is not yet in the summary and \(Sum\) the summary. The symbol \(Q\) is easiest described as a hidden query, and \(Sim(S, Q)\) can be seen as a score representing how good the sentence is to be used in a summary (without taking redundancy into account). The second part of the formula subtracts score if the sentence is too similar to the existing summary.
CHAPTER 2. RELEVANT SUMMARIES

2.5 Sentences to summary

Once sentences have been extracted for usage in a summary, some processing might be needed to form these into a coherent summary. There are two main issues to consider when doing this processing: references and ordering.

2.5.1 References in sentences

Words in a sentence can refer to things explained in different sentences, as well as to other concepts thought to be known by the reader from the context in which the text occurs. This matters when constructing a summary from a set of sentences, since the sentences might contain references to things which may or may not be present in other sentences used for the summary. Handling this might lead to better summaries.

For example, in the sentences “Martin is hungry. He wants a hamburger.” the word “He” refers to the person “Martin”. When the sentences stand together the reference is clear, but consider if only the sentence “He wants a hamburger” ended up in a summary. Then the reference refers to something which might not even be in the summary. If the reference refers to something elsewhere in the summary, the summary must be constructed so that the references are understandable.

Another kind of reference is when the sentence contains references to something that is presumed to be known by readers of the text, given the context of it. Time references are of this kind, as well as references to titles etc. References to time can for example be “Last Tuesday”, and when summarizing multiple documents the words might not refer to the same actual date.

When referring to titles of persons, the surrounding sentences might give a clue to the reference. For example, in “Obama was born in Hawaii. The president attended law school.” it is rather obvious to most readers that “The President” refers to Barack Obama, current president of the USA. However, this assumes that the reader knows that there is a person named “Obama” which is currently a president.

A document, or a document collection, might contain several title references, where the same title might be given to different persons and vice versa.

2.5.2 Ordering of sentences

Sentence ordering should be done in a way such that the final summary is coherent for a reader. References should always refer to something, and time lines should be kept. The sentence ordering is of course strongly dependant on the references contained in the sentences selected to form a summary.

The methods proposed for sentence ordering in existing research vary in ambition and results. Most of the work in sentence extraction summarizing have been focused on the extraction, not on the ordering of the extracted text. Many summarizers, which perform very well in the extraction part, have very simple ways of ordering
sentences (they are aware of this). For example, ordering sentences according to their relative position in their original document have been used, as well as simply putting the highest scoring sentence first.

D. Bollegala et al. [2] have focused on sentence ordering using a more advanced approach, while not constructing the sentence extractor. Rather they assumed that the extraction had been done, and ordered extracted sentences according to their own method; they used probabilistic methods building summaries bottom up, beginning with pairing then pairing pairs etc. Their results show that their method performed very well. It also shows that ordering of sentences is a subject worth exploring in order to produce more readable summaries.
Chapter 3
Solr and Lucene

Solr is a search server which is built on top of the Lucene index. Both systems are open source and Solr is very easy to modify by writing your own code, and then use it as a module in Solr. The Lucene index contains a lot of functionality for free text search, making development of modules requiring such functionality possible.

3.1 Search engines

I chose to approach the summarizing task using a search based approach. The reasons for this are based on that search engines which support free text search are similar to summarizers in several aspects. The most important similarity for this thesis is that both kinds of systems are used to find relevant data in large quantities of text using text as input. The text input varies though, but we’ll get back to that later. First we need to understand how search engines work.

Generally, search engines work by first indexing a number of documents, in order to more easily search in these documents.

3.1.1 Document representation

Documents in search engines are generally treated as one or several fields which contain data. The fields usually have names, and there are often requirements of at least one field which uniquely identifies the document. Fields may, or may not, be searchable.

3.1.2 Inverted Index

Most modern free text search engines build upon the concept of inverted indexes. In free text search engines, an inverted index works by mapping words to the documents where they appear, as shown in figure 3.1.

Prior to searching, the search engine must index all the documents which are to be searchable. It is during indexing that the mapping between terms and documents
is done. After indexing, the search engine can return documents which contain the terms; since the mapping is already done, this can be done very fast.

**Multiple search terms**

When searching for several terms, the index can be used to first find a document list for each term, and then combining the lists. The manner in which the lists are combined differ, depending on the type of query the search engine got as input. The most common is to have either a “all” or a “any” type of query, or combinations of those. In a “all” query, all terms must be present in the returned results, while a “any” query only requires one of the terms.

### 3.1.3 Scoring

Some queries may have a large number of document hits, and then it is of interest to know which of the documents are the most important ones. This can be done by giving a score to each hit, and then order the results accordingly. One way of scoring documents which is commonly used is the TF-IDF measure described in section 2.4.1. Both the Term Frequency and the Inverse Document Frequency values can be calculated when indexing. The TF-IDF values are separate for each document, and for each term. A score can be assigned to a document depending
3.2. USING SOLR AND LUCENE

on the TF-IDF values the document has for the terms in the query. If the query contains multiple terms, each should be taken into account.

3.1.4 Document representation

Search engines in general represent documents in a “bag-of-words” representation, which means that they do not consider the layout of the document, just the words in it and their position in the document. The position in the document is used when a query is placed indicating that a sequence of terms should be matched, but not used for scoring.

For this thesis this is very relevant; it means that any operations involving document layout will have to be done prior to indexing.

3.2 Using Solr and Lucene

There are rather many search engines, but few which are really good, and even fewer which are both good and suitable for this thesis. Here we use Solr and Lucene as the basis for the task of summarizing, for a number of reasons. The main ones were:

Existing functionality Lucene and Solr together contains a wide array of tools which makes processing of large quantities of text easier. Lucene is built for searching, and the index keep track of several things important to text processing. It is also relatively easy to add new data as well as new data types to the index, and to use this data.

Speed Lucene is built to be fast when it comes to searching; when the index has been built, retrieving matches for searches is fast. While speed is rarely calculated and even more rarely tested in existing summarizing solutions, a summarizer using an index to create summaries should be faster than a summarizer which creates summaries directly from text. The downside here is of course that the documents which are to be summarized need to be indexed first.

Modularity Solr can be told to use different modules for different types of requests. This can be used in order to create your own modules, and using the existing architecture of Solr to handle requests. This makes the coding needed to apply the summarizer considerably less, while still allowing for much freedom when it comes to coding the summarizing modules.

Open Source Lucene and Solr are both open source, making existing code available for study and modification. Being stuck with something closed to modification would not had been good.
3.3 Documents and fields

In Lucene, documents are represented as a number of fields; what kind of data the fields should contain can be specified.

For example, a book might contain a field with the book’s title, a field containing its ISBN number (International Standard Book Number, an international and widely used unique identifier for books), a field containing the publish date and finally a field with its content, i.e. the actual text. Each of these fields might be used for searching for books, one might for example want to find all books containing the text “cats are brown”. The date field might be used in order to order search results by date, something which is different from ordering text.

The storing of the fields in the index is also something which might be considered; some fields does not have to be stored as continuous text in the index. For example, the full content of the book might be unnecessary information for the indexer; it might suffice to store the term occurrences of the content in the inverted index.

The fields as well as the field types are declared in one of Solr’s configuration files, which is called “schema.xml”. In order to explain how fields are configured, a small sample of the the field declarations are included below.

```xml
<field name="title" type="text" indexed="true" stored="true" />
<field name="isbn" type="string" indexed="true" stored="true" />
<field name="publish_date" type="date" indexed="true" stored="true" />
<field name="content" type="text" indexed="true" stored="false" termVectors="true"/>
```

The title is a field with the type of text, it should be indexed and stored. Stored fields can be retrieved when searching, one could for example use the title of a book when listing results. The ISBN number is stored as a string rather than text; string fields are not processed before being indexed, while text fields are processed using linguistic methods. The publish date is stored as a date type, which will enable searching for dates which are not necessarily written in the same format. The content field is of the text type; but it should not be stored. It will therefore not possible to easily retrieve the entire book as a single text; it might however be possible to do this with a large number of queries, if not stopped. [17]

Note that there are a parameter telling Solr to store the term vectors. This option saves the term vectors for the “content” field for every document. The term vectors are stored as the Term Frequency values for each term in the content field.

3.4 Queries in Lucene and Solr

In Lucene and Solr, queries are regarded as document request; Solr receives queries via HTTP requests, while Lucene represents queries as Java objects. In general, Solr will build a Lucene query from its input request, which Lucene then executes and then Solr convert the results from Lucene to whatever format it was asked to
return the results in. The formats supported when returning the results are XML and JSON.

Queries in Lucene have to be able to return a Scorer (via a couple of steps); a scorer should be able to score documents. A Scorer does not necessarily have to give scores to documents depending on the content of the document’s fields; it is fully possible to create a scorer which scores documents depending on how much rainfall Zurich has had in the last 3 days. The scorers included in Lucene are however designed to return documents which matches the original query.

One of the most used queries in Lucene is the “TermQuery”, which basically says that a single term should occur in a document, and the query’s scorer scores the document depending on the number of occurrences of that term in the document. The term query’s scorer weights the score in accordance to the TF-IDF measure discussed in 2.4.1.

Likely the most used query in Lucene is the “BooleanQuery”, which is used to combine different queries into a single query. The BooleanQuery contains an unlimited amount of sub queries as input, and its scorer returns a slightly normalized sum of the scores from the sub scorers (provided by the sub queries). The BooleanQuery also keeps track of whether each of its sub queries “should”, “shall” or “shall not” appear. Combined with the fact that a BooleanQuery can be a sub query of another BooleanQuery, it is possible to construct very complex and precise queries.

3.5 More Like This

“More Like This” is a functionality in Lucene, which Solr also provides an interface to. It is built to find documents similar to a specific document, and this is a task which is very similar to the task of finding relevant summaries. Both tasks involve finding documents in an index which are similar to some other document. Lucene’s “More Like This” requires that the document for which to find similar documents exists in the index.

The “More Like This” functionality takes, among other parameters, a document identifier as input, and its task is to find similar documents in the index. This is done by finding the most relevant terms in the input document (the number of relevant terms can be set), and then construct a search query from these. The terms in this query are given a higher relevance proportional to the number of times they occurred in the input document.

The constructed query is then handled as a regular search query, and the results are returned as output; these are the documents which are considered similar to the input document.
In this chapter, the construction of the summarizer is explained. The goal when building the summarizer was to make it as general as possible, in order to be able to use it to summarize documents from different sources. Different document sources have different preconditions, and the deployment of the summarizer for different sources will have to include building certain parts separately. Thus this chapter describe the different parts of the summarizer, while not locking in to any specific data source.

**Figure 4.1.** Architecture of the summarizer.
4.1 Architecture

When using Solr and Lucene, one has to accept the existing architecture of the systems. The good thing about this architecture is that it can be expanded very easily. Three different parts needed to be built in order to use Solr and Lucene as a basis for a summarizers. In addition to these three parts, the Solr configuration was changed so that Solr would use the parts built.

The three parts were the Preprocessor, the Summarizer Component and the Summary Fetcher. They fit together according to figure 4.1. The three parts are very loosely connected; the Sentence Splitter and the Summary Fetcher are in reality only helping parts which should be adapted to documents coming from different sources or to when the summary should be delivered to a special source.

4.2 Preprocessor

The preprocessor has three main tasks: to split documents into sentences, to calculate parts of what relevance a summary would have in a summary and to insert the sentences and documents. Since Lucene stores no data regarding the layout of the documents or sentences in the index, every computation which use the layout as input needs to be done before indexing.

The preprocessor is highly dependant on the input data; likely, different preprocessors will have to be written for different sources. This is partly due to the fact that there is no standard for how documents are to be represented, and partly due to that different kinds of documents have different layout. It is therefore important to make the preprocessor as loosely connected to the Summarizer Component as possible, as this makes the process of writing different preprocessors easier.

4.2.1 Splitting

For splitting, the naïve approach to this task is to simply split on periods; however, this approach will produce strange output when dots are used inside sentences. For example, consider abbreviations; the sentence “Animals, e.g. cats, can move” would with this naïve approach be split into “Animals, e.” , “g.” and “cats, can move”. This is of course not a good set of sentences to use in a summary. Depending on data input, splitting on dots may or may not be a good way of splitting a document into sentences. Since the splitting is highly dependent on the input data, this has do be done for every deployment.

4.2.2 Boosting

Sentences which have certain positions in a document have been shown to be more relevant for summaries. It would be foolish to ignore this fact, so some work is put into marking certain sentences as better than others for summaries.
4.3. SUMMARIZER COMPONENT

This is done by utilizing a feature of Lucene, which is called “Document Boosting”. In short, this function applies a boost to the scoring function on a document basis. The boost is set when indexing, and is the same for the document regardless of what query is posed. This allows sentences to be given boosts, since they are added to the Lucene index as documents.

Sentences with high relevance are often found at the end and beginning of documents, and in the end and beginning of paragraphs. Measuring a sentence’s position relative to a document is easy; when parsing the document, just keep a count of how many sentences have been seen. In order to extract positioning within a paragraph, it is necessary to know where the paragraph starts and where it ends. If the documents contain explicit information about where paragraphs begin and end, this is trivial. Even if the documents do not contain this kind of explicit mark up, the document can contain implicit information regarding paragraph splits. For example, line spacing or line indentation might be different at the start or end of paragraphs. Again, this information is specific to different input data, and each source needs to be handled separately.

4.2.3 Inserting

After extracting sentences and boosting them, both documents and sentences will have to be inserted into the index. Both documents and sentences are treated as documents in the index, and are added similarly. It is however important that sentences and documents are indexed in such a way that the information about if they are sentences or documents are not lost in the index. This can be done using different fields for the content, or by simply adding another field which specifies what kind of document it is. How this is done is up to each deployment of the system.

The fields used by Solr is configured in a configuration file, which is located in the directory in which Solr is located. The implementation of the insertion needs to be aware of what fields are used for what, and set these fields accordingly.

4.3 Summarizer Component

The summarizer component is a component written to be used in Solr, and is built out of existing elements from both Solr and Lucene. The summarizer component can be used as other components in Solr, and is called by setting an extra parameter when searching for documents. The summarizer component works by the following algorithm:

1: Search for document(s) from a input query
2: Construct query from the best of the found documents
3: Search for sentences from the constructed query
4: Create a summary from the best found sentences
5: Return the summary
By constructing the summarizer component this way, one can use the Lucene search to choose which documents should be summarized. This enables summarization of both dynamical document clusters from search hits, as well as from static clusters; one needs simply to set a Solr field to contain an identification for which cluster(s) a document belongs to, and search for this field with a specific value.

### 4.3.1 Disclaimer

The summarizer component was built by modifying the “More Like This” functionality described in 3.5. Much of the code found in the implementation of this functionality has been reused, although re-factorized to provide a more dynamic environment to develop new functionality in. This thesis does not claim credit for the code which have been re-used; but some changes are far from trivial, the functionality described in this thesis is novel and the parts of the implementation which have not been re-used are as well.

### 4.3.2 Creating a summary query

The creation of a summary begins with a user (or some other external source) stating a search query, searching for documents in the index. This query is a regular search query, and the search is done in the same manner as if the summarizer component had not been used. The summarizer component will then look at the top scoring documents, and summarize these documents. The number of documents which are summarized is configurable, but defaults to the top ten documents.

The documents which are to be summarized are then processed in order to retrieve their term vectors, i.e. the number occurrences of each term in a document. Depending on configuration, the term vectors for the documents might be stored in the index, in which case the term vectors can simply be retrieved from the index. If the term vectors are not stored, the document is instead parsed, and the number of term occurrences are counted. This of course requires the index to store the content of the field, something which is not always done.

The term frequencies from all documents which are to be summarized are added together, thus constructing a term-frequency mapping which corresponds to the combined term distribution of the documents to be summarized. Several fields can be used when extracting term frequencies, and if that is the case the term frequencies from the different fields are added together. The different fields can be given different boosts, if this is the case the term frequencies from the different fields are summarized with weights corresponding to the boost.

Now we have a mapping of terms frequencies which have been added together from different documents and fields. The frequencies does not any longer have to be integers, the weighting may have turned them into fractions; we will however continue treating them as the frequency of occurrences of a term.

A query is then created from the term-frequency map. The query consists of a number of sub queries, and every term in the term frequency map is a sub query.
4.3. SUMMARIZER COMPONENT

Sub queries can be assigned weights, and we can use this in order to use the term frequencies in the term-frequency map in the search. This is done by assigning weights to each sub query, depending on the frequency of the term in the term-frequency map. This means that terms which occur often in the documents which are to be summarized will have a high weight in the constructed query.

We have now constructed a query which will be used to search for a good summary.

4.3.3 Searching and scoring sentences

When a query has been constructed, sentences matching this query is searched for in the index. This is done by existing functionality in Lucene, so no code for parsing the index needs to be written; one can however specify what kind of scorer should be used for the search. The scorer is the component which gives scores to documents, and each query have its own scorer. Sub queries are also queries, so they also have scorers. The scorers then assign scores to sentences, and the top scoring sentences will be returned. The number of sentences to be returned can be set; in this implementation, the default is to return 1000 sentences.

Two different methods of scoring were used; one was implemented using only scorers provided with Lucene, and one was implemented with custom scorers.

TF-IDF Scoring

The first kind of scoring used was implemented using only existing scorers from Lucene. The query types used for this scoring method are the BooleanQuery and the TermQuery mentioned in section 3.4. As mentioned earlier, these scorers are used very often throughout Lucene and Solr; this kind of scoring is thus the same as regular searching in Lucene.

The scoring is done quite simply; for each term in the query, the TF-IDF score of that term is added together. This score is then the score returned by the scorer.

KL-divergence Scoring

The second kind of scoring was done according to a variant of the KL-divergence method described in section 2.4.1. Remember that the KL-divergence is a divergence measure, which can be negated in order to create a similarity measure. In order to score the documents according to this method, the query was seen as a document; the more similar a query is to a document, the higher score the document in the index gets.

The goal was to use the symmetric KL-divergence; it did however not really work out. The KL-divergence is designed to measure differences between probability distributions, and no probability in any of the distributions are allowed to be zero. In our case, that means that the index must contain all the terms in the query and that the query must contain all the terms in the index, with a weight larger than zero.
CHAPTER 4. BUILDING A SUMMARIZER FROM A SEARCH ENGINE

While it is very probable that all the terms in the query exist in the index, having all the terms in the index in the query is not applicable for anything but very small indexes; the query would take a very long time to run.

Instead of using the symmetric KL-divergence, a variant of the regular KL-divergence was used. The formula used was this:

\[
Score = 1 - \frac{\sum_{t \in \text{Query}} \text{Weight}_t \times \log \left( \frac{\text{Weight}_t}{P_t} \right)}{\sum_{t \in \text{Query}} \text{Weight}_t \times \log \left( \frac{\text{Weight}_t}{10^{-5}} \right)}
\] (4.1)

where

\[
P_t = \begin{cases} 
\text{Freq}_t & \text{if } \text{Freq}_t > 0 \\
10^{-5} & \text{else} 
\end{cases}
\] (4.2)

That is, subtract the normalized divergence from one to get a similarity measure. Since the KL-divergence would not be defined if \( \text{Freq}_t \) was zero, a small value is set \((10^{-5})\) to emulate the behaviour of the divergence measure for very small probabilities.

It is also assumed that no \( \text{Freq}_t \) is lower than this small number; this is however not very probable; terms seldom occur less than \(10^{-5}\) times in a document, even with boosting counted. This assumption enables normalization via the fact that the maximum divergence for a single term occurs when \(P_t\) is very small.

4.3.4 Constructing the summary

When the top scoring sentences have been found, they are used to construct a summary. In this implementation, no regard is taken to the sentence ordering in the summary; the purpose of this step is to avoid redundancy in the summary.

This was done using a variation of the MMR method described in section 2.4.3, modified for performance. The MMR method does in its standard form require that you re-score every sentence each time you add a sentence to the summary. This is of course not very efficient, since the index might contain a very large number of sentences.

Instead, only the sentences returned by the original search are considered; this is the reason why the default amount is so large; the MMR can now afford to discard those which are too similar to the summary, while not considering the entire index. This is of course a greedy approach, and it might miss out on some sentences.

The summary constructor starts with selecting the best scoring sentence and setting it as the start for the summary. It then iterates through the rest of the sentences returned by the search, re-scoring them according to the MMR method and then pushing them into a priority queue.

The priority queue orders the documents according to a comparator; the comparator compares two sentences by how much score they have after the re-scoring.
4.4. SUMMARY FETCHER

The top sentence in the queue is then added to the summary and removed from the queue. Then the remaining sentences in the queue are re-scored and the queue is re-built to match the new scores.

4.4 SummaryFetcher

The summary fetcher’s job is to fetch the summaries from the summarizer component. Since the summarizer component is built as a component for the Solr search server, the communication is done using web requests. The fetching is done using web requests, and the format of the results can be requested to be either JSON or XML. Solr also provides a Java API, which can be used to ease the communication by handling all the requests within the API, controlled using Java methods.

The fetching is done by posing a query, which is then sent to the Solr server as a HTTP request. If the query contains a parameter specifying that a summary should be generated, the Solr server will use the Summarizer Component to generate one. The summary will then be included in the response returned to the requester.

How the summary is used is then up to the requester; it might be shown on a website or written to a file.
Chapter 5

Deploying the summarizer

This chapter describes the deployment of the summarizer on a dataset, which comes from the Document Understanding Conference. The deployment is done separately for each data source; this part may be re-done for another data source, using the same Summarizer Component but with other parameters.

5.1 Document Understanding Conference

The Document Understanding Conference (DUC) was a series of conferences which aimed at creating systems which can process large amount of information. The first conference was held 2000, and the latest 2007. A summarization task has been a part of the conference, and participants of the conference have been able to take on the challenge of automatic summarization.

For this task a new dataset have been put together each year. The dataset of 2006 and 2007 consists of a number of documents, which are pre-categorized news articles from different sources, marked up using a simple mark up language similar to XML. Each category contains a number of documents, and each category also has a short narrative describing its content.

One interesting thing to note here is that the narrative have been used by other summarizers when creating summaries of the documents in the category, and thus it is also used for this thesis. Using an existing narrative of a cluster of documents is of course not applicable when summarizing dynamically generated clusters. Something similar to a narrative, or at least some terms, connecting documents in a dynamically generated cluster might be found if you look at how the cluster has been put together. For example, if the documents are searched for the query might be an indicator. This is however not applicable for the DUC-07 summarization task, since no queries were provided, so the narrative is used.

Each article in the every cluster contains a unique identifier, a headline, the date it was published in addition to the text of the article. The text in the article is also divided into paragraphs.

In order to evaluate the summarizer built for this thesis, it was deployed on
the DUC-07 dataset. This was done while building the summarizer, and used as a testing basis for looking at what would make the summarizer better or worse. However, in hindsight, this might have been a bad decision. The participants in the DUC-07 summarizing task did not get to evaluate their summarizers against the real dataset while developing them; this of course gives the summarizer produced for this thesis a slight advantage. However, the advantage is not that great; the other summarizers could have been evaluated against the DUC-06 dataset, which contains data similar to the data of the DUC-07 set.

5.2 Preprocessing

The preprocessing of the data is the part of the summarizing which is most specific for each data source. The DUC-07 dataset contains a lot of meta-data, which might help for creating good summaries. Each document was indexed by itself with a number of fields, and also split into sentences which were also indexed; the fields were indexed using different field types. The field types used were “text” and “string”. Remember that strings are simply sequences of letters, while texts are fields which is also processed with different linguistic tools.

The documents were indexed using the following fields:

**Document number** This is a unique identifier for each article, and is not used for anything else than identifying. This field was stored as a string type.

**Headline** The headline for each article might give a good hint to what the article is about. This field was stored as text type.

**Content** This is the text of the article. All markup in the text (paragraphs) were removed from the text. This field was stored as text type.

**Category Label** This is a label for the category, used to identify which category a document belongs to. This field was stored as string type.

**Category Title** This is the title of the category to which the document belongs. The category briefly describes the content of the category. This field was stored as text type.

**Category Narrative** This is a short narrative describing the content of the category in more detail than the category title. This field was stored as text type.

Note that the date field is not used; it was assumed not to be important for the extraction part of the summarizer, since it uses term frequencies as its main method of scoring. It might have been good to include the date for sentence ordering, but the ordering of sentences were not taken into consideration.

Each article was indexed using these fields, and also split into sentences. The sentence splitting is done by first splitting up the text in paragraphs, which were
marked up in the dataset, and then splitting the paragraphs into sentences. It seems that the DUC-07 dataset have been produced with sentence splitting in mind, since it is really easy to split sentences in it; simply splitting on dots is good enough (very few or no splitting errors).

As mentioned earlier, sentences are generally more important if they appear early or late in a document or a paragraph. In order to incorporate this in the summarizer, sentences appearing in such places are given a boost. The boost is calculated according to this formula:

\[
\text{Boost} = \omega \times |\text{Pos}_p - \sigma| + (1 - \omega) \times |\text{Pos}_d - \sigma|
\] (5.1)

Where \( \text{Pos}_p \) is the relative position of the sentence in the paragraph, and \( \text{Pos}_d \) is the relative position in the document, both normalized to \([0, 1]\). The constants \( \omega \) and \( \sigma \) are weight factors; the value \( \omega \) affects which of the positions is given the most importance, while \( \sigma \) controls if early occurrences or late occurrences are the most important. The value of \( \omega \) was set to 0.7 and the value of \( \sigma \) was set to 0.5 for the DUC-07 dataset.

When the sentences have been boosted, they are inserted into the index. For the sentences only two fields are used:

**Sentence Number** This is a number generated from the document which the sentence was taken from, combined with the position of the sentence in the document (not normalized). This is only used as a unique identifier for each sentence. This field was stored as *string* type.

**Sentence Content** This is the content of the sentence, extracted from a document. This field was stored as *text* type.

### 5.3 Summarizer Component

The Summarizer Component was not adapted in more than modifying some parameters, which is done when requesting a summary; this is done in the fetcher.

### 5.4 Summary Fetcher

Two fetchers were built for the DUC-07 dataset. One was a very simple HTML page which fetched the summary using JSON, while one was a small application which wrote the summaries to files. These files were written in a way so that they could easily be evaluated against the participants of the DUC-07 summarizing task participants.

The fetcher which was built as a HTML page was just a proof of concept, showing that it can be done. In figure 5.4 one can see the interface of the summary fetcher. It works by showing an interface where a user can pose a query, and also say how
many words should be included in the summary. Remember that the search query is an actual search query of the index; so the summarizer component will return a number of documents matching the query and a summary of those documents, and present them to the user.

The fetcher used to write files for evaluating against other summaries creates its own queries. Since the summary task is about summarizing each category, the query requests all documents in a single category. They will of course then be summarized, and written to a file.

When comparing sentences to other summarizers, no regard is taken to the speed at which the summaries are generated (since it is not for the others). Therefore a very large number of sentences were fetched to use in the summary, out of most of course are filtered out and not used.

The summary challenge allows a maximum of 250 words to be used in the summary, so that is the size of the summaries produced for the task.
Chapter 6

The Summaries

6.1 Evaluation

The performance of the summary was, as mentioned earlier, measured against other summarizer in the DUC-07 summary task. This was done using the ROUGE system, which calculates the difference between a generated summary and a known good summary. The DUC-07 dataset came with a set of such good summaries, produced in advance by humans.

The comparison was done using the ROUGE-SU4 and the ROUGE-2, discussed in section 2.1.2. The score used for comparison is an average from evaluation of a lot of summaries produced from a number of different categories.

Included in the DUC-07 dataset was a series of top scores in these areas, so comparing the score of the summarizer was as easy as comparing the score given to the summarizer by ROUGE with the top scorers for each ROUGE method.

Different components of the summarizer were tested for their efficiency; every possible combination of components were however not tested. The ones tested were the ones which yielded the best results, as well as components deemed to be of special interest. The components are described in greater detail earlier, but a short description is given here:

**Positioning** Indicates that the sentence’s position in the document has been used to affect summaries.

**Narrative** Indicates that the narrative of a category have been used when creating summaries.

**TF-IDF** Indicates that TF-IDF has been used when searching for sentences for summaries.

**KL-Divergence** Indicates that the Kullback-Leibler method for scoring was used when searching for sentences for summaries; this is mutually exclusive with the TF-IDF method.
CHAPTER 6. THE SUMMARIES

MMR Indicates if the Maximum Marginal Relevance method has been used when creating summaries.

6.1.1 DUC-07 Results

The results are presented in table 6.1, compared to the top scoring system in DUC-07 (DUC-07 Top) and to a reference system from DUC-07 (Duc-07 Ref). The reference system works by selecting the leading sentences for documents in a category, up to the maximum length.

Table 6.1. The results attained by ROUGE evaluation of summaries from the DUC-07 dataset. Higher scores indicate higher average similarity to known good summaries, making them considered better.

<table>
<thead>
<tr>
<th>Method</th>
<th>ROUGE-2 Score</th>
<th>ROUGE-SU4 Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>DUC-07 Top</td>
<td>0.17528</td>
<td>0.21892</td>
</tr>
<tr>
<td>Narrative, TF-IDF</td>
<td>0.1003</td>
<td>0.15624</td>
</tr>
<tr>
<td>Positioning, Narrative, TF-IDF</td>
<td>0.10000</td>
<td>0.15384</td>
</tr>
<tr>
<td>Positioning, TF-IDF</td>
<td>0.09107</td>
<td>0.14408</td>
</tr>
<tr>
<td>Positioning, Narrative, TF-IDF, MMR</td>
<td>0.08459</td>
<td>0.13892</td>
</tr>
<tr>
<td>Positioning, Narrative, KL-Divergence</td>
<td>0.07255</td>
<td>0.12876</td>
</tr>
<tr>
<td>DUC-07 Ref</td>
<td>0.06039</td>
<td>0.10507</td>
</tr>
</tbody>
</table>

6.1.2 Meaning of the results

There are several remarkable things with these results. Neither the sentence positioning nor the MMR method had a positive impact on the produced summaries. The reason why the MMR method did not work out well may be that the evaluation system ROUGE takes no explicit regard to repetition of information in the summaries; thus not giving any benefit to summarizers which strives to create a summary without repetition. The reason why the sentence positioning did not work out is of course a combination of the formula used to boost sentences according to position and of the structure of the documents to be summarized. It might work out better with another formula, or with other texts.

The Kullback-Leibler divergence method did not work out well either; it is however a more novel approach to searching than the more usual TF-IDF measure. One should remember that the KL-divergence comes from probability theory, and it maybe it was a bad move to try it, but it might also be so that more refinement is needed to use it in a search engine.

The narrative did however have a positive effect, but that is to be expected; using a summary when generating a summary gives you a considerable advantage.
6.2. RELATION TO EXISTING WORK

6.1.3 Comparing results with others
There were more participants in the DUC-07 challenge than those presented in table 6.1. The results varied, but most were around the same score as the best results produced by the summary produced for this thesis. If the summarizer had been a participant, it would had been at position number 22 out of 43 in the ROUGE SU-4 top score. Several of the top scorers did however use machine learning and clustering algorithms, more suitable for the controlled environment of the DUC-07 summarization task.

6.2 Relation to existing work
This thesis builds upon results from other researchers, several methods have been shown to be successful before. For example, both the ideas of sentence positioning and the Maximum Marginal Relevance method have been used to make the summarizer better. The main idea of this thesis is however that term frequency measures is a good measure to follow when using sentence extraction for summarizing.

6.2.1 Similarities to existing work
The only real book written on the subject of automatic summarization ("Automatic Summarization", published in 2001) heavily implies that TF-IDF measures is not a good measure for automatic summarization of multiple documents [16]. Others have however been able to use the idea of relative frequencies to create summarizers, showing that this method is far from trivial [19]. Something is rather interesting though: probability theory has entered the field of automatic summarization, using probabilities to find out what sentences are relevant.

This thesis have been influenced by the work of Henning and Labor in choosing the similarity measures used. [10] The difference is rather that they as well as other see it as hidden “topics” to which a sentence has a certain probability to belong. The number of topics are constant, set for each summarization process.

Different sentences have different chances of making it into the summary, depending on how large their probability is of belonging to the same topic as the documents which they are supposed to make a summary of. With a bit of fantasy one can see that there is a connection between these topics and the term frequencies.

The number of occurrences of terms in a document should indicate which topic the document is likely to belong to. The same is true for the sentences; the terms which occur an unusually large number of times in a sentence should indicate what topic it would belong to.

6.2.2 Novel Parts
The summarizer described in this thesis uses these two measures, and can thus be seen as a variant of the method involving hidden topics. However, previous solutions
seem to require a set number of topics, this might be a problem if the document collection changes. In the built summarizer, no explicit number of topics are needed.

This might not matter much when evaluating the summarizer in a small, controlled environment. However, if one would like to dynamically create summaries for documents in a set of data which might change over time, this is a great strength. For example, if one would index a large encyclopaedia many users could dynamically request summaries of several documents in the encyclopaedia. This could be done with very little computational effort, resulting in fast responses to the users.

If the encyclopaedia grows or changes, the index can be updated with the changes; one of the beauties of Lucene is that adding, updating and removal of documents are relatively fast operations.

This is what is novel about this summarizer; it can be used in an entirely different fashion than most other summarizers. The features which make this summarizer different from existing summaries might be something which can change the way we look at summarizer systems.

6.3 Future Work

The summarizer described in this thesis is a prototype, and from here there are at least two tracks which might be explored. One is how to make better summaries, while still using a search based approach, and the other is to research how this kind of summarizer might be used in real-world applications.

Making the produced summaries better are done by improving the method used to select viable sentences, working with relevance models which are applicable in search engines.

Researching how to use the summarizer might be done by deploying it on a real life dataset, for example an encyclopaedia.

6.4 Conclusions

The summarizer produced for this thesis does not outscore existing summarizers, yet it shows a different way of viewing the summarizing task by splitting up the work in two parts; it uses a search engine to do the summarizing. The connection between searching and summarizing have been shown to be non-trivial; summarizers may be built using methods used in search engines.

By using an established search server to encapsulate the summarizer, the summarizer is easily deployable in existing environments.

The problem formulation of this thesis was: "Is it possible to build a multi-document summarizer using a search based approach?”. The answer to this question is: Yes, it is possible.

The novel thing about this approach is that a large amount of the work is done before the summarizing takes place. This makes the summarizer fast when searching, making the summarizer usable in a different manner than existing systems.


