NTNU
Innovation and Creativity

# Implementation and evaluation of Norwegian Analyzer for use with DotLucene

**Bjørn Harald Olsen**

Master of Science in Computer Science
Submission date: June 2006
Supervisor: Tore Amble, IDI
Co-supervisor: Stein Joar Gran, InfoFinder Norge AS

Norwegian University of Science and Technology
Department of Computer and Information Science

## Problem Description

DotLucene is the .NET port of the information retrieval (IR) library Lucene, which is developed by Apache Software Foundation. Lucene is a full featured IR library, with support for indexing and searching. Lucene is open source and is distributed under the Apache License. InfoFinder Norge AS desires an implementation of a Norwegian Analyzer for DotLucene.

An analyzer is a module which extracts words, discards punctuation, removes accents from characters, performs case folding, removes common words, and normalizes words. The goal of this thesis is to develop and test an as complete as possible analyzer for the Norwegian language. The analyzer must be implemented in C# and .NET.

Assignment given: 20. January 2006
Supervisor: Tore Amble, IDI

## Abstract

This work has focused on improving retrieval performance of search in Norwegian document collections. The initiator of the thesis, InfoFinder Norge, desired an Norwegian analyzer for DotLucene. The standard analyzer used before did not support stopword elimination and stemming for Norwegian language. Norwegian Analyzer and standard analyzer were used in turns on the same document collection before indexing and querying, then the respective results were compared to discover efficiency improvements.

An evaluation method based on Term Relevance Sets were investigated and used on DotLucene with use of the two analyzer approaches. Term Relevance Sets methodology were also compared with common measurements for relevance judging, and found useful for evaluation of IR systems. The evaluation results of Norwegian analyzer and standard analyzer gave clear indications that use of stopword elimination and stemming for Norwegian documents improves retrieval efficiency.

Term Relevance Set-based evaluation was found reliable by comparing the results with precision measurements. Precision was increased with 16% with use of Norwegian Analyzer compared to use an standard analyzer with no content preprocessing support for Norwegian. Term Relevance Set evaluation with use of 10 ontopic terms and 10 offtopic terms gave an increased *tScore* of 44%. The results show that counting term occurrences in the content of retrieved documents can be used to gain confidence that documents are either relevant or not relevant.

# Preface

This thesis is the final work of my master studies at the Department of Computer and Information Science at the Norwegian University of Technology and Science in Trondheim.

## Acknowledgments

I would like to thank several persons for giving me assistance throughout the work. First of all I thank Jeanine Lilleng for co-operation regarding Term Relevance Sets and feedback on the report in general. I also thank Tore Amble my supervisor. Thanks to InfoFinder Norge AS represented by Stein J. Gran who gave me feedback, and Håvard Stranden who recommended me to InfoFinder so I could write this thesis. Øyvind Vestavik receives thanks for giving me access to a large Norwegian document collection.

Bjørn Harald Olsen
June 15, 2006

# Contents

**Preface** iii

**List of Figures** ix

**List of Tables** xi

**I Setting** 1

**1 Introduction** 3

1.1 Introduction . . . . . . . . . . 3

1.2 Motivation . . . . . . . . . . 3

1.3 Goals . . . . . . . . . . 4

1.4 Thesis outline . . . . . . . . . . 4

**II Background** 7

**2 Information retrieval systems** 9

2.1 Introduction . . . . . . . . . . 9

2.2 Information retrieval . . . . . . . . . . 9

2.2.1 Information versus Data retrieval . . . . . . . . . . 10

2.3 Relevancy . . . . . . . . . . 11

2.4 Indexing and searching . . . . . . . . . . 11

2.4.1 Inverted file index . . . . . . . . . . 12

2.5 Retrieval models . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12
2.5.1 Boolean models . . . . . . . . . . . . . . . . . . . . . . . . . . . 12
2.5.2 Vector space models . . . . . . . . . . . . . . . . . . . . . . . . 13
2.5.3 Probabilistic models . . . . . . . . . . . . . . . . . . . . . . . . 14
2.6 Improve retrieval efficiency with text preprocessing . . . . . . . . . . . . 15
2.6.1 Suffix stripping: Stemming versus lemmatization . . . . . . . . . 17

**3 Lucene information retrieval library 19**

3.1 Introduction . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 19
3.2 Lucene usage . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 19
3.3 Indexing and searching . . . . . . . . . . . . . . . . . . . . . . . . . . . 21
3.4 The QueryParser . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 22
3.5 Ranking in Lucene . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 23
3.6 Analyzers in Lucene . . . . . . . . . . . . . . . . . . . . . . . . . . . . 24
3.7 Conclusion . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 25

**4 Evaluation of IR systems 27**

4.1 Introduction . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 27
4.2 Retrieval evaluation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 27
4.2.1 Recall and precision . . . . . . . . . . . . . . . . . . . . . . . . 28
4.2.2 TREC collections . . . . . . . . . . . . . . . . . . . . . . . . . . 29
4.3 Evaluation using Term Relevance Sets . . . . . . . . . . . . . . . . . . . 31
4.3.1 Introduction . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 31
4.3.2 Goals . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 31
4.3.3 Understanding Trels . . . . . . . . . . . . . . . . . . . . . . . . . 32
4.3.4 Exploration of Trels . . . . . . . . . . . . . . . . . . . . . . . . . 34
4.3.5 Results . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 37
4.3.6 Discussion . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 40
4.3.7 Conclusion . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 41

**III Implementation 43**

**5 Implementation of Norwegian Analyzer 45**

5.1 Introduction . . . . . . . . . . 45

5.2 Components . . . . . . . . . . 45

5.3 NorwegianStemmer: Stemming using Snowball . . . . . . . . 47

5.4 Verification of NorwegianStemmer . . . . . . . . 48

**6 Implementation of Trels-based evaluation 51**

6.1 Introduction . . . . . . . . . . 51

6.2 Evaluation procedure . . . . . . . . . . 51

6.3 Open and parsing content of documents . . . . . . . . 52

6.4 Counting terms . . . . . . . . . . 52

**7 Implementation of Trels-based evaluation framework for DotLucene 55**

7.1 Introduction . . . . . . . . . . 55

7.2 Components . . . . . . . . . . 56

7.2.1 IR system based on DotLucene . . . . . . . . 56

7.2.2 Evaluation of IR system based on DotLucene . . . . . . 57

**IV Results 59**

**8 Results 61**

8.1 Introduction . . . . . . . . . . 61

8.2 Evaluation procedure . . . . . . . . . . 61

8.3 Deciding cutoff point . . . . . . . . . . 62

8.4 Document collection . . . . . . . . . . 62

8.5 Term Relevance Sets . . . . . . . . . . 63

8.6 Results . . . . . . . . . . 64

8.7 Discussion . . . . . . . . . . 65

8.8 Conclusion . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 67

**9 Conclusion and further work 69**

**V Appendices 71**

**A Fields in Lucene 73**

**B Query subclasses in Lucene 75**

**C Common words in the Norwegian language 77**

**D Indexing benchmark 79**

**E Term Relevance Sets used for exploration of Trels-based evaluation 81**

**F Term Relevance Sets used for evaluation 85**

**G Results of Trels-based evaluation experiments 91**

**H Norwegian stemmer in Snowball 95**

**Bibliography 99**

# List of Figures

2.1 Information retrieval system . . . . . . . . . . . . . . . . . . . . . . . . . . . 10

2.2 Boolean model Venn diagram [3] . . . . . . . . . . . . . . . . . . . . . . . . 13

3.1 Lucene and application responsibilities [12] . . . . . . . . . . . . . . . . . 20

3.2 Query terms used with appropriate query-methods . . . . . . . . . . . . 22

3.3 Formula used by Lucene to determine document score based on a query [11]. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 23

3.4 Using analyzer when adding documents to the index . . . . . . . . . . . 24

3.5 Using analyzer when retrieving documents . . . . . . . . . . . . . . . . . 24

3.6 Analyzer is a collection of operations . . . . . . . . . . . . . . . . . . . . 25

4.1 Illustration of document subsets used for precision and recall calculations [3]. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 28

4.2 Sample recall-precision curve from TREC 2005 [19]. . . . . . . . . . . . 29

4.3 Document level scoring ($tScore(d_i, q)$) and collection scoring ($tScore(D_q)$) 33

4.4 Calculation of result list score $tScore(D_q)$ on relevant documents . . . . 38

4.5 Experiments on relevant, irrelevant and fifty-fifty document collection . 39

4.6 Changing priority of terms . . . . . . . . . . . . . . . . . . . . . . . . . . 39

5.1 Components of default StandardAnalyzer . . . . . . . . . . . . . . . . . . 45

5.2 Components of NorwegianAnalyzer . . . . . . . . . . . . . . . . . . . . . 46

5.3 Documents are first converted to a stream of tokens, filtered and conflated into terms in index . . . . . . . . . . . . . . . . . . . . . . . . . 46

5.4 Relevant classes and their relations . . . . . . . . . . . . . . . . . . . . . 47

6.1 Calculating score for a collection of retrieved documents $D_q$ . . . . . . . 52

6.2 Parsing supporting different document formats. . . . . . . . . . . . . . . 52

7.1 Trels-based evaluation framework overview . . . . . . . . . . . . . . . . . 55

7.2 Trels-based evaluation framework processes . . . . . . . . . . . . . . . . . 56

8.1 Abstract evaluation procedure for NorwegianAnalyzer and StandardAnalyzer. 62

G.1 Normalized for document size in relevant document . . . . . . . . . . . . 91

G.2 Mean of random selected term pairs . . . . . . . . . . . . . . . . . . . . . 92

G.3 Normalized for document size in irrelevant documents . . . . . . . . . . 93

G.4 Mean of random selected terms in irrelevant documents . . . . . . . . . 93

G.5 Fifty-fifty relevant and irrelevant documents . . . . . . . . . . . . . . . . 94

G.6 Mean of random selected terms in fifty-fifty relevant and irrelevant documents . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 94

# List of Tables

2.1 Sample inverted file index [31] . . . . . . . . . . . . . . . . . . . . . . . . . 12

2.2 Document and query vectors for a given vocabulary [31] . . . . . . . . . 14

2.3 Conditional probabilities [31] . . . . . . . . . . . . . . . . . . . . . . . . . 15

3.1 Examples of companies and search engines that are using Lucene [12]. . 20

3.2 Lucene scoring factors [12]. . . . . . . . . . . . . . . . . . . . . . . . . . 23

4.1 Sample TREC topic [26] . . . . . . . . . . . . . . . . . . . . . . . . . . . 29

4.2 Sample Qrels [18] . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 30

5.1 Components of Norwegian Analyzer in executed order . . . . . . . . . . 46

5.2 Sample words with their stemmed equivalent . . . . . . . . . . . . . . . 49

7.1 Associated classes . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 57

7.2 Components of Trels-based evaluation framework . . . . . . . . . . . . . 58

8.1 Boost factors for documents of different size . . . . . . . . . . . . . . . . 63

8.2 Evaluation results of StandardAnalyzer with and without stopword elimination, and NorwegianAnalyzer . . . . . . . . . . . . . . . . . . . . . . . 64

8.3 Increased effectiveness using different analysis operations . . . . . . . . . 65

A.1 An overview of different field types, their characteristics, and their usage [12] . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 73

B.1 Query subclasses in Lucene [12] . . . . . . . . . . . . . . . . . . . . . . . 75

# Part I

# Setting

# Chapter 1

# Introduction

## 1.1 Introduction

This chapter will give an introduction to the topic of this thesis and describe the task, aimed goals, and chapter layout of the thesis.

## 1.2 Motivation

The idea of automatic access of large amount of stored knowledge was given already in 1945 [5]. The principle of searching is quite trivial, but the solutions facilitating searching are complex computer systems called information retrieval (IR) systems. The goal of an IR system is to find the information item(s) satisfying an information need often expressed by a query. Based on what the system know, usually only a query, items which satisfies the assumed information need is located and returned to user. It is the task of the system to return the items with the highest probability of being interesting to the user.

The initiator of this thesis is InfoFinder Norge AS. InfoFinder aims at facilitating search in business data. An enterprise may have many different data stores; documents, spread sheets, presentations, PDF-files, e-mail, drawings, scanned documents or telefaxes. Their goal is to make all possible data sources available through a single search interface.

InfoFinder uses a open source system for indexing and searching called DotLucene which is the .NET port of Lucene. InfoFinder choose not to implement software performing indexing and searching on their own. This gives them the opportunity to focus on creating smart interfaces and tools for information source organization which increases functionality and quality of search results.

Many of InfoFinders customers are seated in Norway and the content of the data sources is often written in Norwegian. For better storing and searching it common to use language dependent preprocessing of text content. Because of this, they requested a module performing content preprocessing for Norwegian. The module should fit the existing system used for search in business data.

My approach is to implement a module for preprocessing Norwegian text documents performed before indexing and searching for Lucene. After this module is found to satisfy given specifications, it will be focused on evaluation of the resulting system when the module is put into practice. The evaluation methods will be investigated through experiments and finally used to compare the new approach for preprocessing Norwegian documents with the module previously used.

## 1.3 Goals

The two main goals of this thesis are to

- Adapt DotLucene for use on Norwegian documents collections.
- Investigate and use Term Relevance Sets for evaluation of Norwegian Analyzer with use of DotLucene.

Norwegian Analyzer for DotLucene will be implemented according to given methods for text preprocessing of Norwegian documents. The methods are standard operations from the DotLucene library, stopword elimination, and an stemming algorithm for Norwegian. A simple IR system will be created with use of DotLucene and both the previous standard analyzer used, and Norwegian Analyzer created. Performing the same evaluation with use of both analyzers should give an indication of how much Norwegian Analyzer increases the quality of search results. The evaluation method will first be investigated and finally used for evaluation of DotLucene with use of Norwegian Analyzer.

## 1.4 Thesis outline

The report describes topics first generally and secondly especially for practical usage. The main focus will be information retrieval (including analysis of Norwegian) and evaluation. The following structure can be expected:

*Background*
Chapter 2: Information retrieval in general
Chapter 3: Information retrieval with use of Lucene
Chapter 4: Evaluation of information retrieval systems in general and with use of Term Relevance Sets

*Implementation*
Chapter 5: Implementation of Norwegian Analyzer

Chapter 6: Implementation of relevance measurement based on Term Relevance Sets
Chapter 7: Implementation of evaluation framework based on Term Relevance Sets

*Results*
Chapter 8: Evaluation and results

# Part II

# Background

# Chapter 2

# Information retrieval systems

## 2.1 Introduction

People are not interested in searching, but *finding*. Information technology let us store knowledge digitally and once data stores grow its getting harder to browse through the documents in order find an answer to the question we might have.

IR systems performs organization of document collections and make it possible to access documents satisfying the needs of a user. Throughout this thesis the definition of a document is used as follows:

> "A document is a unit of text that is returned in response to queries. It might be a paragraph, a section, a chapter, a Web-page, an article, or a whole book" [3].

This chapter will discuss topics which are supposed to improve efficiency of IR systems. An efficient IR system is a system which manages to present documents to the user, which satisfies the needs. Documents satisfying information needs of a user are called *relevant* documents.

To improve efficiency we use retrieval models which are algorithms that use available information to locate relevant documents among all the stored documents. Additionally, the content of documents can be processed before storage, so that the effect of retrieval models increases. This chapter will discuss classic models in information retrieval and common text processing techniques used to increase efficiency of IR systems.

## 2.2 Information retrieval

The different tasks encapsulated in an IR system has together one common goal; to retrieve *relevant* documents in short time. Figure 2.1 shows basically the interface between the user and the information. Time efficient retrievals is of great importance

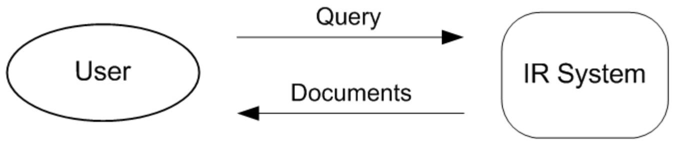


**Figure 2.1:** Information retrieval system

and many techniques are developed in order to achieve both time and space efficient data structures. These subjects are interesting, but they will not be discusses in detail through this thesis, except that it will be mentioned at relevant occasions. The main focus will be on how well IR systems perform retrieval of *relevant* information.

Its not always easy to know what is relevant for the users querying for something. For example in a music library when a user searches using a sub string of either a artist, song title or genre, the domain is limited which makes it easier to know that retrievals are relevant. The World Wide Web consists of many information domains which make the search much more complicated. Searching for "zeppelin", would make it hard for the search engine to if the users means the rock band "Led Zeppelin" or "Zeppelin airships".

This chapter will describe information retrieval more in detail and common methods used to achieve as relevant results as possible.

### 2.2.1 Information versus Data retrieval

Data retrieval must not be confused with information retrieval. The goal of data retrieval is more definite than information retrieval, which means it does not have to handle the big question of *relevancy* that's a big issue related to information retrieval. Data retrieval aims at retrieving data which satisfies clearly defined conditions, such as regular expressions or a relational database. The query language is often more specifically defined, in contrast to information retrieval which handles *natural language* queries[3].

In the context of IR systems, data retrieval consists mainly of finding documents which satisfies a specific keyword or a term. This does not mean the documents retrieved satisfying the subject or topic which the user is looking for. The task of finding which of these documents that's relevant and how they should be ranked is related to information retrieval. This task is quite undetermined and related to content "interpretation" (syntactic and semantic) and user context analysis. There are many techniques used to find out what's relevant to the user, and will be discussed in the following sections.

## 2.3 Relevancy

As stated before, *relevancy* is quite relatively. Determining what is relevant changes depending on what kind of information retrieved for, as well as the context of the user. This means, if the same user searches in two information sources, that is different of nature, the system should modify parameters of its ranking model for each search performed.

If a search is performed in an electronic mailbox its highly relevant to retrieve *all* e-mail, both outgoing and incoming, that matches for example a specific e-mail address. If a single e-mail is judged irrelevant by the IR system, an important part of a conversation could be left out. This is a situation where we want to find everything related to a query. In the scope of information retrieval we measure this kind of retrieval using the unit *recall*.

On the other hand, in an e-commerce environment users searching for a specific product are interested in *few* documents matching the exact product name which are low priced. Now we want to find only those documents that relates to a given query. This is called *precision*.

These examples represents two different goals of information retrieval and different methods and ranking models are used to achieve them. While precision and recall are discussed further in section 4.2.1, we now go on discussing the methods and ranking models.

## 2.4 Indexing and searching

Indexing and searching are core functionalities in IR systems and choosing the right methods is crucial for time efficient retrieval as well as performing relevant document retrieval. Before going into details they will be defined shortly.

- *Indexing*: processing the original data into a highly efficient cross-reference lookup in order to facilitate rapid searching. The result of indexing is called an index. An index is basically the same as an index of a book. When looking up a word you get references to pages where the word appears. An searchable index consists of references to different documents which the word appears [12, 31].
- *Searching*: the process of looking up words in an index to find documents where they appear [12].

There may different ways of creating indices. Specially for text the most suitable structure is an *inverted file index* which is the structure used by Lucene [31, 12]. The next section contains a simple example of an inverted file index and how its used for searching.

### 2.4.1 Inverted file index

Inverted file index is a general concept which is used as basis for the indexing core of many IR systems, where Lucene is one of them [12]. A simple example is created below to explain the most important aspects of inverted file index.

Table 2.1 shows a sample index where "Term" is the indexed word and "Documents" the positional information for the terms. List of appearances or positions for each term is using the notation $\langle r; (x; y_1, y_2, ...), ...\rangle$. Where a term appears in $r$ documents and for each document $x$ the term appears at position $y_1, y_2, \ldots$.

| Number | Term | Documents |
|---|---|---|
| 1 | about | $\langle 2; (1; 4), (2; 3)\rangle$ |
| 2 | always | $\langle 2; (1; 2), (2; 1)\rangle$ |
| 3 | best | $\langle 1; (1; 6)\rangle$ |
| 4 | losers | $\langle 2; (1; 1), (2; 4)\rangle$ |
| 5 | whine | $\langle 2; (1; 3), (2; 2)\rangle$ |
| 6 | their | $\langle 1; (1; 5)\rangle$ |

**Table 2.1:** Sample inverted file index [31]

This sample have information both about term appearance in document and position inside the document. This can be used to to search for phrases, since its possible to find the distance between terms inside a document. Different levels of detail can be used. In contrast to previous example, we could only include information of which documents terms occurs, and not the position inside documents. This would result in an index consuming less disc space, but decrease the level of search complexity possibilities, such as phrase distance.

## 2.5 Retrieval models

> "A ranking algorithm operates according to basic premises regarding the notion of document relevance. Distinct sets of premises (regarding document relevance) yield distinct information retrieval models" [3].

Retrieval models are used to find relevant documents in a collection based on some kind of evidence. Most used is query evidence, where the terms in a query are the basis for selecting relevant documents. This section guide through the three classic retrieval models used to predict relevant documents.

### 2.5.1 Boolean models

This is the simplest way of processing a query. It is the oldest of the three classical models and already introduced in the 50s [23]. Each term in the query are combined with the logical connectives AND, OR, and NOT which means the intersection, union and complement, respectively. Documents satisfying the logical expression are retrieved to the user. Lets say we have the query below, where $t_x$ are query terms.

$$(t_1\, AND\, t_2)\, AND\, NOT\, t_3$$

If $D_t x$ are documents satisfying a query term $x$ we would retrieve the documents marked by the shaded area in figure 2.2. The nature of the boolean model implies either

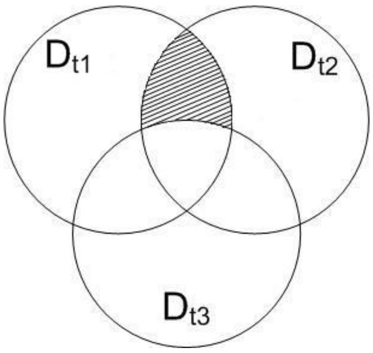


**Figure 2.2:** Boolean model Venn diagram [3]

exclusion or inclusion of documents. There are no way or weighting query terms so for example half of the documents matching a query term are considered as a possible relevant result. Hence, the choice stand between retrieving many documents (OR) or few documents (AND). Which means high precision or high recall, respectively [31].

Boolean retrieval systems, despite its limits, were the primary mechanism used to access information for more than three decades [3]. Currently, more featured methods have been developed, featuring weighting of terms in contrast to inclusion or exclusion. To day boolean model have more or less been substituted by the model in next section.

### 2.5.2 Vector space models

Vector space models are also called *similarity models* since they measure the similarity between vectors of weighted terms. Queries or documents can be expressed as vectors. The vector shows what words of a vocabulary occurring in a query or document. A similarity measure of a query vector of terms and a document vector of terms shows how close they are in the vector space. Measuring the similarity between a document vector and query vector, gives us an idea of how relevant a document is to a given query. If a query is close to a document in the vector space, we assume that the document is relevant to the query.

Similarity measure might be calculated using the inner product of the two vectors, or alternatively an inverse function of the angle between the corresponding vector pairs [24]. A simple example will explain it more clearly. Lets say we weight terms based on how many times they occur in the text. Considering table 2.2 we have two vectors, one for document $D$ and query $Q$ defined as

$$\begin{aligned} D &= \text{"Pease porridge hot, pease porridge cold"} \\ Q &= \text{"hot porridge"} \end{aligned}$$

| Vocabulary | cold | hot | pease | porridge |
|---|---|---|---|---|
| Document vector, $D$ | 1 | 1 | 2 | 2 |
| Query vector, $Q$ | 0 | 1 | 0 | 1 |

**Table 2.2:** Document and query vectors for a given vocabulary [31]

Lets say we use the inner product to measure similarity between vector $Q$ and document $D$, the similarity is expressed as

$$M(Q, D) = Q \cdot D$$

The inner product of two vectors $X = \langle x_i \rangle$ and $Y = \langle y_i \rangle$ with $n$ dimensions is defined to be

$$X \cdot Y = \sum_{i=1}^{n} x_i y_i$$

Using the query and document vectors from table 2.2 the similarity measure is

$$M(hot\,porridge, D) = (0,0,0,1,0,0,0,0,1,0) \cdot (1,0,0,1,0,0,0,2,2,0) = 3$$

In contrast to the boolean model, which would have returned the document as relevant for "hot AND porridge", the vector model returns a scalar which tells *how* relevant the document is, based on term occurrences [31].

### 2.5.3 Probabilistic models

There are many ways of using probability for ranking documents in information retrieval. The classic way is based on the *Probability Ranking principle* [20], and the models attempt to rank documents by their probability of relevance given a query [14, 21]. There were early found benefits from putting fundamental theory of probabilistic retrieval together with indexing and searching methods [22].

This section will focus on the classic probabilistic model where ranking is found based on relevance probability based on documents and a given query. Currently, a lot of research focus on probabilistic models with *query independent* evidence [7, 28, 29]. Yet, the following classic probabilistic model example uses query evidence and is based on Bookstein&Swanson [4].

A specific term (typically from a query) is supposed to either prove that a document is relevant or not relevant. First we need to establish a weight for each term which will be used to prove the relevancy. Secondly, the probability for each term appearing in the document, are multiplied in order to find total weight for a document. High weight means high probability for the document to be relevant.

| | Number of documents | | |
|---|---|---|---|
| | Relevant | Nonrelevant | Total |
| Term t present | $R_t$ | $f_t - R_t$ | $f_t$ |
| Term t absent | $R - R_t$ | $N - f_t - (R - R_t)$ | $N - f_t$ |
| Total | $R$ | $N - R$ | $N$ |

**Table 2.3:** Conditional probabilities [31]

The creation of term weights conditional probabilities are estimated based upon known relevance judgments. Lets say we have a collection of $N$ documents, where $R$ documents are relevant. $R_t$ of the relevant documents contain term $t$ and $f_t$ are the documents in the collection where term $t$ appears. These values are set from a training set of documents which relevance judgments have already been decided [31].

Table 2.3 are used to estimate the conditional probabilities;

$$\text{Pr[relevant} \mid \text{term } t \text{ is present]} = R - t/f_t$$

$$\text{Pr[irrelevant} \mid \text{term } t \text{ is present]} = (f_t - R_t)/f_t$$

$$\text{Pr[term } t \text{ is present} \mid \text{relevant]} = R_t/R$$

$$\text{Pr[term } t \text{ is present} \mid \text{irrelevant]} = (f_t - R_t)/(N - R)$$

Then term $t$ weight $w_t$ is derived using Bayes' theorem:

$$w_t = \frac{R_t/(R - R_t)}{(f_t - R_t)/(N - f_t - (R - R_t))}$$

where values greater than one indicate that term $t$ supports that the document is relevant. Weights less than one indicates that the appearance of term $t$ supports that the document is not relevant [31].

Total weight of a document is estimated on

$$weight(D_d) = \prod_{t \in D_d} w_t$$

## 2.6 Improve retrieval efficiency with text preprocessing

In general, text preprocessing and lexical analysis is very central in the scope of information retrieval and in order to achieve relevant results. Mainly the preprocessing operations are; extracting words, discarding punctuation, removing accents from characters, case folding, removing common words (stopwords) or normalization (suffix stripping) [3, 12, 2, 31].

*Stopword elimination* is one of the operations included in Norwegian Analyzer. Its no doubt that elimination of common words, or semantically insignificant words, increases the performance of IR systems. For example would the query "President of United States" match the document string "President of the United States" if "the" was removed. Elimination of common words is widely used and accepted among experts.

How do Google handle common words? According to Google they ignore common words when preprocessing queries [15]. However, they do not say anything about the indexing phase. A simple test shows that they actually do not remove common words before indexing. The query "to be or not to be", with and without quotes shows this[1]. Without quotes shows that *not* is the only word considered while all the others are thrown away. With the quotes documents about Shakespeare are retrieved. Hence, common words are not removed during indexing [12]. Google needs a lot of storage in order to index all common words of large parts of the Web. For those who do not have this ability, eliminating stopwords make the size of the index smaller and less storage consuming.

*Suffix stripping* has also become more and more a natural part of IR systems. Basically, it means elimination of plurals, past participles, genitive endings etc. Its important to note that suffix stripping is assumed to be done both on user query and on document content during indexing. For example would a document containing

```
CONNECT
CONNECTED
CONNECTION
```

after suffix stripping be

```
CONNECT
CONNECT
CONNECT
```

Since "connect", "connected", and "connection" now are transferred to the same term "connect" in the index, a query for one of them would match all three. The consequence is that documents containing terms closely related to the query terms are returned.

A second effect of suffix stripping is the that the total number of terms in index is reduced. This results in less disc space consumption. If the index search algorithm for example has an asymptotic run time of $\Theta(n)$, term reduction would also decrease search latency.

Suffix stripping basically means to find the basic form of a word. Two approaches are

[1]The "to be or not to be"-test on Google were done 12-05-2006

common for doing this; lemmatization and stemming, and will be discussed in following section.

### 2.6.1 Suffix stripping: Stemming versus lemmatization

Stemming and lemmatization in information retrieval, basically, aims at the same goal; suffix stripping. Lemmatization can yet imply prefix stripping, for example complements for words; visible contra invisible. Use of this in information retrieval would result in the opposite of the information need, thus this feature is not desired.

*Lemmatization* is the process of finding the *lemma* of a given word. Since it aims at finding the right meaning of a word, its required to determine the part of speech of the word, therefore lemmatization is also called part-of-speech tagging [8]. The disadvantage of this approach is the need of grammar knowledge and large amount of implementation effort.

*Stemming* has several advantages compared to lemmatization. Stemming operates at one word at a time and the result is called the *stem* of a word. This means no grammar knowledge requirement and no need to consider part-of-speech for each word. There are two well-known stemming algorithms; KSTEM [13] and Porters algorithm for suffix stripping [17], where the latter are relevant for this thesis. Section 5.3 describes Porters algorithm for Norwegian in detail.

# Chapter 3

# Lucene information retrieval library

## 3.1 Introduction

Lucene is an high performance, scalable Information Retrieval (IR) library [12]. Lucene provides text indexing and searching capabilities for applications needing search features, which are many.

Lucene is currently a member of the Apache Jakarta family of projects [12] and licensed under the liberal Apache Software License [30]. Lucene is free, open-source and originally implemented in Java. Note that Lucene is not a *search engine*. Its a software library or a tool kit which can be used by others to create full-featured search applications.

The original Java implementation is ported to several other programming languages; Perl, Python, C++ and .NET [12]. The Lucene port for .NET will be used during implementations later on and therefore referred to as DotLucene.

## 3.2 Lucene usage

Providers of search solutions may choose degree of effort putting into developing search algorithms and indexing infrastructure. Some companies, such as Fast Search & Transfer[1], has created their own technology and algorithms for ranking and indexing documents [2].

There are examples of companies finding it unnecessary to develop machinery for indexing and searching. Instead of putting a great deal of effort into developing already efficient algorithms, they focus on other tasks. Intelligent use of different data sources

[1]FAST - Solution provider developing their own technology (http://www.fastsearch.com/)

| | |
|---|---|
| Nutch.org | Open-source search engine for the World Wide Web. |
| jGuru.com | Community-driven web-page for Java developers. |
| SearchBlox | Search tool focusing on companies. |
| Michaels.com | Michaels Stores, Inc. is an arts and crafts retailer with more than 800 stores in the United States and Canada. |

**Table 3.1:** Examples of companies and search engines that are using Lucene [12].

and intelligent presentation of results to the user, are tasks that have a greater potential for improvement than index and search algorithms. Often this results in a two-layered IR systems; the application layer and the layer providing machinery for indexing and searching.

In figure 3.1 we see that Lucene takes care of the most basic tasks; indexing and searching. While a given application performs gathering of data and presentation of results. The application probably have more knowledge about the actual user, and can perform these tasks so that they fit a given scenario.

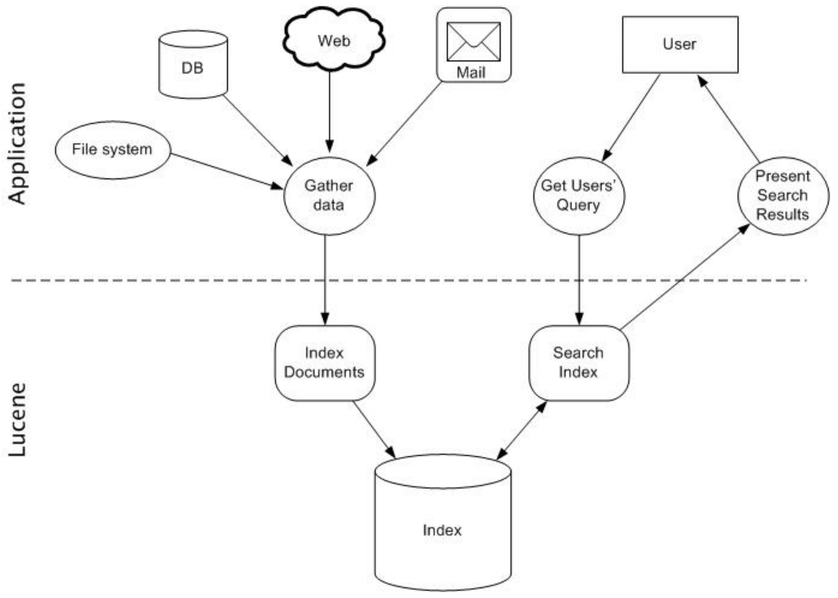


**Figure 3.1:** Lucene and application responsibilities [12]

InfoFinder, as mentioned, uses Lucene as indexing and search machinery and adds intelligent software on top to improve relevancy of results. Table 3.1 shows several other companies and search engines that are using Lucene as bottom IR machinery.

## 3.3 Indexing and searching

This section will in a simplified and understandable way describe how to index a document and search for it in Lucene. Four main steps have to be done; create index, add document to index, search index, and explore hits (results are called hits).

First an index is created using the class IndexWriter. Irrelevant properties are marked with dots.

```
IndexWriter writer = new IndexWriter(indexDirectory, ... );
```

Now an empty index has been created. Lets create a document an add it to the same index. The content of the document are added to a text field called "contents". A document may consist of several *fields*. Its up to the developer to divide documents into appropriate fields. For example could the title and body of a document be separated into two fields. If many documents are indexed with use of the fields "title" and "body", we have the possibility of retrieving or searching among the title or body of all documents. An advantage is that we have the opportunity to put more weight on the title when searching among all documents. (for more fields and explanations see appendix A).

```
Document doc = new Document();
doc.add(Field.Text("contents", new FileReader(documentFile)));
writer.add(doc);
```

The index contains now one document and is searchable. Below a IndexSearcher is instanced, a simple query created, and search performed. TermQuery is one of several Query-subclasses supported by Lucene. More information about TermQuery and other subclasses can be found in appendix B. There will also be discussed more about query-classes in the later section 3.4. Now, lets see how we search and retrieve some results.

```
IndexSearcher searcher = new IndexSearcher(indexDirectory);
Query query = new TermQuery(new Term("contents", "mason"));
Hits hits = searcher.search(query);
```

The container Hits will now consist of all documents that have the word "mason" in a "contents" field. Each hit is also given a score between 1 and 0, which tells how relevant Lucene believes the document is to the query.

Its actually not harder than this. Yet, it does not mean that it *can not* be done harder. As mentioned in section 3.2 the application facilitating the collection of data sources and interaction with user has the possibility to perform intelligent decisions in order to fit a given environment and users. Some of the classes mentioned above have different factors and configurations, such as "boost" for Query, Field and Document. "Boost" can be used to weight content that is believed to be specifically relevant. Construction of queries can also be done in a more advanced way. Different kinds of queries can be used in combination in order to achieve appropriate retrievals. Next section will discuss the query issue more in detail.

## 3.4 The QueryParser

Different query methods supported by Lucene are listed in appendix B. Developing a search application gives the opportunity to use these query methods to create nested queries fitting a give scenario. This section will give an example of how query methods can be nested for a book scenario. For example could a user type the query "book 1-932394-28-1". It consists of the term "book" which indicates that we are looking for a book, and the ISBN number of the book. In this case it would be appropriate to use a query method which retrieves the exact match of the ISBN number. The query term "book" could probably be weighted less because a book record would consist of author, title, publisher, etc.

QueryParser can be used by IR systems which prefers automatic nesting of query methods. Lets go into an example of a nested query. Three such query-methods supported by Lucene are; TermQuery, PhraseQuery and BooleanQuery. TermQuery matches exact terms in the index, and is useful for retrieving keys (such as ISBN). PhraseQuery matches terms within a certain distance of each other. BooleanQuery use AND, OR and NOT to logically combine its containing clauses, which may be other queries. So, how would we use these in combination for a specific query? Lets say we have the

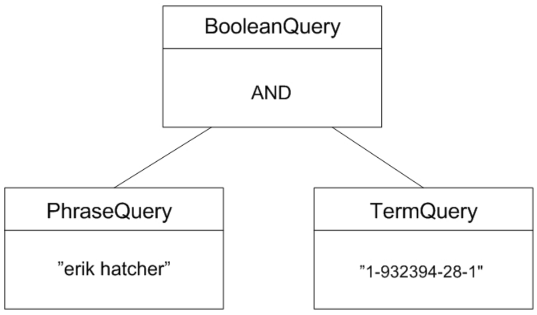


**Figure 3.2:** Query terms used with appropriate query-methods

query "erik hatcher 1-932394-28-1". First, its pretty obvious that TermQuery is appropriate for the ISBN number. Secondly, we know that "erik" and "hatcher" would be to separate terms in the index, and it would be appropriate to use the PhraseQuery. In the end, BooleanQuery could be used to combine these two with an AND operator, as illustrated in figure 3.2.

This example shows how different query-method could be used in combination for a given query. Its not a fact that QueryParser would chose to do it this way, but probably something in that direction. QueryParser is not right for every scenario. Some IR developers may prefer to instance the query manually, to be sure that the query is appropriate for the specific needs.

## 3.5 Ranking in Lucene

Searching may result in a number of documents believed to satisfy the question or information need expressed in a query. Its a crucial task to develop algorithms that finds these documents. Earlier in section 2.5, three classic models for ranking and retrieval was described. Lucene uses a model based on vector space models or similarity models [9]. This section describes the retrieval model of Lucene. Note that Lucene is under constant construction and the model may develop. The model described here is based on the Lucene version as in benchmarks under appendix D

Each document retrieved receives a *score*. The score explained here is the *raw score*. Its important to note this since the score returned for a document hit is not necessarily the raw score. Lucene guarantees that all scores are 1 or less. If the top-ranked document scores greater that 1, all document hit scores will be normalized such that the top-ranked document receives score 1 [12].

$$\sum_{t\,in\,q} tf(t\,in\,d) \cdot idf(t) \cdot boost(t.field\,in\,d) \cdot$$
$$lengthNorm(t.field\,in\,d) \cdot coord(q,d) \cdot queryNorm(q)$$

**Figure 3.3:** Formula used by Lucene to determine document score based on a query [11].

The formula in figure 3.3 shows how score is calculated based on terms $t$ in the query $q$ and a document $d$. The score is computed for each document matching a query [12]. Table 3.2 explains the factors used in the formula.

| Factor | Description |
|---|---|
| tf(t in d) | Term frequency factor for the term (t) in the document (d). |
| idf(t) | Inverse document frequency of the term. |
| boost(t.field in d) | Field boost, as set during indexing. |
| lengthNorm(t.field in d) | Normalization value of a field, given the numbers of terms within the field. This value is computed during indexing and stored in the index. |
| coord(q,d) | Coordination factor, based on the number of query terms the document contains. |
| queryNorm(q) | Normalization value for a query, given the sum of the squared weights of each of the query terms. |

**Table 3.2:** Lucene scoring factors [12].

Boosting is featured so that the score can be influenced manually. Boosting can be done for the whole document or specific fields. Such as document titles could be boosted so they would influence the score significantly. If a query term is found in the title of a document, we assume this document to be extra relevant. Queries may also be boosted, either a specific term in a multiple-clause query or the whole query. A boosted query results in that documents hitting this query are boosted [12].

## 3.6 Analyzers in Lucene

A topic of this thesis is to implement and evaluate an analyzer for Norwegian. In section 2.6 it was explained how text preprocessing is used to improve efficiency of IR systems. Lucene have something called analyzers, which is a collection of text preprocessing operations such as stopword elimination and stemming described in section 2.6. Lets define analysis and an analyzer.

> "*Analysis*, in Lucene, is the process of converting field text into its most fundamental indexed representation; *terms*. An *analyzer* is an encapsulation of the analysis process" [12].

This means that an analyzer performs the conversion of document text into terms, so the document can be placed into an index. Basically, conversion is to divide text into pieces or words, perform text processing operations on each piece or word, and finally call it a term.

An IR system have two main text input resources, (1) documents which is indexed and made searchable, and (2) queries typed by users which is used to search the index for documents. Both text inputs are preprocessed or analyzed. Figure 3.4 shows analyzers used when documents are added to the index. Figure 3.5 shows analyzers in use when a query is typed by a user and search is performed.

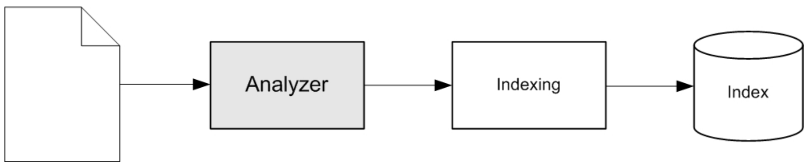


**Figure 3.4:** Using analyzer when adding documents to the index

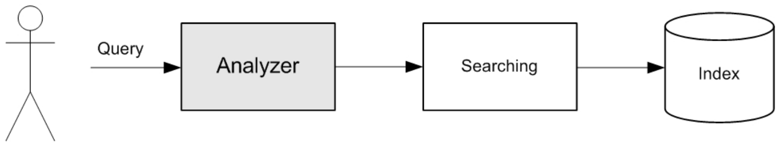


**Figure 3.5:** Using analyzer when retrieving documents

The operations performed by the analyzer may be different depending on the language of the text. Some of the operations are though standard operations, such those dividing text into words separated by whitespaces. Others are created to do linguistic operations for a given language. In this thesis linguistic operations performing elimination of stopwords and stemming for Norwegian are central issues. Figure 3.6 shows use of a standard operations, Tokenization, and two language specific operations.

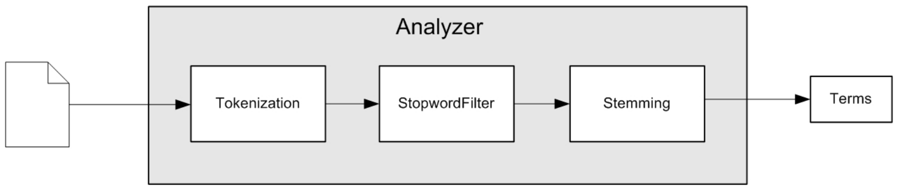


**Figure 3.6:** Analyzer is a collection of operations

In addition to those shown in figure 3.6 there is used a couple of language independent operations. These are StandardFilter and LowercaseFilter. Before the NorwegianAnalyzer was implemented in this thesis, InfoFinder did not have an analyzer specific for Norwegian language. Earlier InfoFinder used StandardAnalyzer from the Lucene library, which consists of only language independent standard operation. StandardAnalyzer can be used in cases of lack of analyzer supporting a desired language. One goal of this thesis is to compare efficiency of using StandardAnalyzer and NorwegianAnalyzer, hence the previous and current situation of InfoFinder. Hopefully, the effort of implementing NorwegianAnalyzer will result in better IR performance.

## 3.7 Conclusion

This chapter has connected the general aspects of information retrieval to Lucene. Index structure technique and retrieval model in Lucene are two important aspects which are based upon general methods in information retrieval. Lucene has also some added features, such as boosting, which enables tuning of document retrievals. StandardAnalyzer from the Lucene library uses basic linguistic operations also used by NorwegianAnalyzer. In addition NorwegianAnalyzer will use techniques specific for Norwegian in order to increase efficiency.

# Chapter 4

# Evaluation of IR systems

## 4.1 Introduction

The enormous increase of available information have resulted in many systems making it possible to reach the one document satisfying the needs of a user. A good example is the World Wide Web (or just the Web); a huge information resource which can be reached via many online search engines. Enterprises also use IR systems in order to keep track of documents stored in different types of information sources; databases, mailboxes and fileservers. Section 2.3 discusses relevancy and how different scenarios sees relevancy different. An enterprise would probably have a different view of relevancy than the Web, which can be dependent on data resources used. *Evaluation* is therefore not always trivial; for example different views of relevancy must be taken into account.

The purpose of this chapter is to describe common evaluation techniques and introduce an less known technique which is explored through experiments. The new method explored is later used for evaluation of DotLucene and Norwegian Analyzer.

## 4.2 Retrieval evaluation

When an information retrieval system receives an input query it returns a list of ranked documents. The ranking method or algorithm used by a specific IR system is usually evaluated by sending in an query and explore the results returned by the system.

The following two sections will (1) describe how we measure effectiveness for a given list of retrieved documents and (2) how reference collections can be used to perform large scale evaluation.

### 4.2.1 Recall and precision

Two common and widely used measurement units for retrieval performance are *recall* and *precision*. *Precision* is the proportion of retrieved material that is relevant. Usually these values are calculated from the $k$ top-ranked documents retrieved by the actual ranking method. Precision $P_k$ is given by:

$$P_k = \frac{number\ retrieved\ that\ are\ relevant\ (Rr)}{total\ number\ retrieved\ (Dr)}$$

*Recall* is the proportion of relevant material actually retrieved. $R_k$ for the $k$ top-ranked documents is given by:

$$R_k = \frac{number\ relevant\ documents\ that\ are\ retrieved\ (Rr)}{total\ number\ relevant\ (Rn)}$$

Figure 4.1 illustrates the variables used in previous definitions. If a ranking method retrieves one document which is relevant we get $P_1 = \frac{1\ relevant\ retrieved}{1\ totally\ retrieved} = 100\%$. If we say that there are totally 10 relevant documents in the collection we get $R_1 = \frac{1\ relevant\ retrieved}{10\ relevant\ documents} = 10\%$.

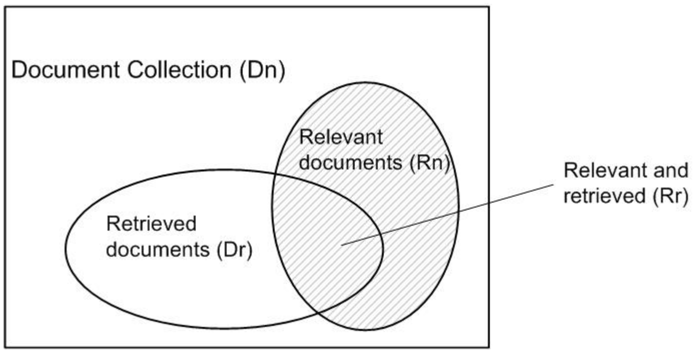


**Figure 4.1:** Illustration of document subsets used for precision and recall calculations [3].

Optimally, recall and precision would be 100%. In that case all relevant document are returned and no irrelevant. Unfortunately, this is rarely the case. Recall-precision curves shows the relation between these two measurement units. Usually high precision values gives low recall, and vice versa. Figure 4.2 shows a sample recall-precision curve from the TREC measures. It gives an idea of the practical relation between recall and precision and the fact that one normally have to take a tradeoff into account.

Section 2.3 discussed two different cases where one prefers precision and the other recall. As mentioned, it depends on user needs and environment. Therefore, when evaluating an IR system we use the measurement unit fitting the goals of the IR system. For example, if one desire to retrieve as many of the relevant documents in the collection as possible, using recall would tell if the system is efficient or not. On the other hand, if the systems goal is to retrieve a proper portion of relevant document, precision would be suitable for evaluation. Optimizing either of the two would maybe hurt the other

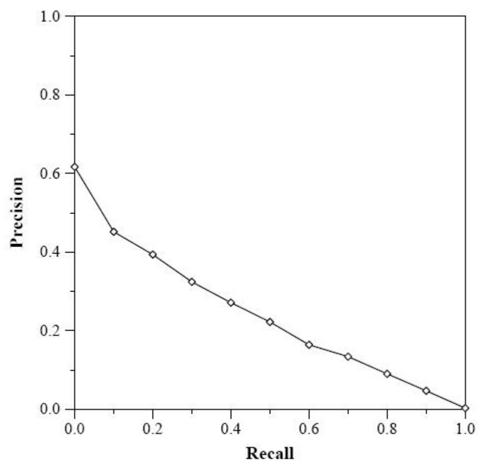


**Figure 4.2:** Sample recall-precision curve from TREC 2005 [19].

significantly. Therefore it could be a solution to optimize both precision and recall at the same time so the solution fits both desires pretty well.

Searching the Web is quite challenging in some ways. The number of relevant document for a given query can be incredibly high. A search engine for the Web which maximizes recall, would probably return many document hardly ever read by the user. Therefore, precision is often maximized by search engines on the Web.

### 4.2.2 TREC collections

The Text REtrieval Conference (TREC) is arranged once a year and has become a standard for comparing IR models and algorithms. People come here to evaluate their IR system. The participants (100 in 2004) receive a document collection and a set of queries or ”topics”.

```
<top>
<num> Number: 505
<title>  edmund hillary; sir?

<desc> Description:
Who is/was Edmund Hillary?

<narr> Narrative:
A relevant document will provide biographical information on
Edmund Hillary.
</top>
```

**Table 4.1:** Sample TREC topic [26]

After all participant have searched the document collection with all the queries, they all have lists of retrieved documents, which their IR system find the most relevant for each query. Each participant hands the lists of retrieval to a "committee" of human assessors. Each document is judged either relevant or not relevant to the query [16].

Queries are described in a topic which is formatted using a simple SGML-style tagging. Table 4.1 above shows a sample TREC topic [27]. The number identifies the topic and the description tag contains a query. The "narrative" field describes what a relevant document should contain. Relevance judgments called *Qrels* are lists telling which documents that are relevant to a given topic. Qrels contains four variables as listed below.

- TOPIC is the topic number.
- ITERATION is the feedback iteration (almost always zero and not used).
- DOCUMENT# is the official document number.
- RELEVANCY is a binary code of 0 for not relevant and 1 for relevant.

Table 4.2 contains five lines picked from a randomly picked Qrels file. The example shows that three of the documents are relevant and two are not relevant for the same topic.

```
301 0 FBIS3-10082 1
301 0 FBIS3-10169 0
301 0 FBIS3-10243 1
301 0 FBIS3-10319 0
301 0 FBIS3-10397 1
```

**Table 4.2:** Sample Qrels [18]

To judge retrievals TREC uses a pooling technique on the set of documents retrieved by each participant. Each document in the pool is judged by a human assessor. When enough results have been assembled and judged, the relevance judgment are considered "complete" for a particular set of documents. It is assumed that the most relevant documents have been found [16]. This means that there still may be relevant documents which are not found by any of the participants. When the judgment is done, the efficiency of each participant's IR system are measured and evaluated.

When all TREC runs are completed and judgments done, the collection of Qrels (judgments) can be used by non-participants and participants to evaluate IR systems. A document collection and belonging queries and Qrels, can be downloaded and used for evaluation. This is an great resource after a conference is finished. English is the language which have the greatest amount of test collections. French, German, and Italian are also languages with growing document collections and judgments.

TREC has done a great job gathering information retrieval experts every year to participate and share knowledge. The database has become very large and they have created a well working infrastructure for large scale testing of IR systems. Though, using TREC Collections for evaluation of IR systems has its drawbacks; evaluation is dependent of fixed document collections and few languages and domains are supported (roughly 8). The creating of Qrels includes manual assessment of documents which is time consuming and requires a great deal of human resources.

## 4.3 Evaluation using Term Relevance Sets

### 4.3.1 Introduction

There is clearly a need for an evaluation method which can be used on any document collection (probably a collection changing daily), requiring less human assessment, and easy to deploy for different languages and domains.

Evaluation with Term Relevance Sets (Trels) is described in *Scaling IR-System Evaluation using Term Relevance Sets* and was published in 2004 by researchers from the IBM Haifa Research Lab, Israel [1]. The key feature and advantage of Trels-based evaluation is, in contrast to Qrels, that no manual document judgment is needed and a evaluation setup can be done for any language. In addition Trels-based evaluation is not adversely affected by changes to the underlying collection, since its not based on document relevance judgments. Experiments show that Trels-based evaluation is highly correlative with Qrels-based evaluation (estimated correlation is 0.93) [1].

Because TREC and Qrels-based evaluation does not have any support for large scale evaluation of Norwegian text and queries, Trels are used for evaluation in this thesis. This chapter will describe the details of the method and explore it on sample collections. This is done as a preparation for the evaluation on a large document collection later in chapter 8.

### 4.3.2 Goals

The further exploration will focus on the following two aspects, which also appears in the "further work" proposals in the article [1]. The answers to these questions will also be important as a basis of the evaluation in chapter 8.

- Facilitating the creation of Trels: how important is it to pick the right on/off topic terms.
- Examining the robustness of Trels: how many terms are usually needed for on/off topic sets.

The second goal implies a subgoal. The decision of how many terms that is needed will be taken out from the resulting relevance score. It is expected after a given number of term additions that the score will start to converge. If this shows to be true this converging score threshold will be used to decide how many terms that is needed. It will also be discussed if the threshold could be used as a yardstick to tell how relevant document retrievals are.

### 4.3.3 Understanding Trels

Trels-based evaluation have a different approach than evaluation based on Qrels in respect of finding out whether documents are relevant or not relevant. Instead of comparing retrieval results to pre-specified relevant pages, Trels-based evaluation examines the content of the results.

The content of each result is evaluated by looking for occurrences of a pre-specified list of terms. These terms are believed to indicate that a document is either relevant or irrelevant to a specific query [1]. The method uses a set of queries as its input. Each query is associated with a Trels, which consist of two sets of terms:

- onTopic: terms related to the query that are **likely** to appear in relevant documents.
- offTopic: terms related to the query but **unlikely** to occur within relevant pages.

If there are found occurrences of onTopic terms in a document, the confidence that this document is relevant increases. If there are found offTopic terms as well, the confidence is decreased. A sample Trels is given below:

```
query: "recycle, automobile tires"

onTopic: "rubberized asphalt", "door mats", playground

offTopic: "traction, air-pressure, paper, plastic, glass
```

As you know, old tires are recycled into door mats or rubberized asphalt and they are often used at playgrounds. When creating the set of onTopic terms we have to either know what kind of terminology used in the desired documents, or we could use a third-party retrieval tool to browse through relevant documents found [1].

The offTopic terms can also be selected by using a third-party retrieval system. If we for example give Google [1] the query "recycle", we would probably get documents about paper, plastic and glass which are related to source segregation. The second query term "automobile tires" can also be associated with traction and air-pressure, which is also irrelevant.

The original query terms may appear in the phrases or lexical affinities in the onTopic set, but the individual query terms and linguistic derivates should be excluded. The

[1]http://www.google.com

motivation for this is that we want to grade the system on terms or compounds they have not been exposed to [1].

## Measures

The relevance of a list of retrieved documents for a specific query are evaluated using a measure unit called *tScore*. Basically the measurement of results for query $q$ is done in two steps; first the $tScore(d, q)$ for each retrieved document $d$ is calculated. Secondly $tScore(D_q)$ for the whole list of retrieved documents is calculated. Figure 4.3 illustrates this.

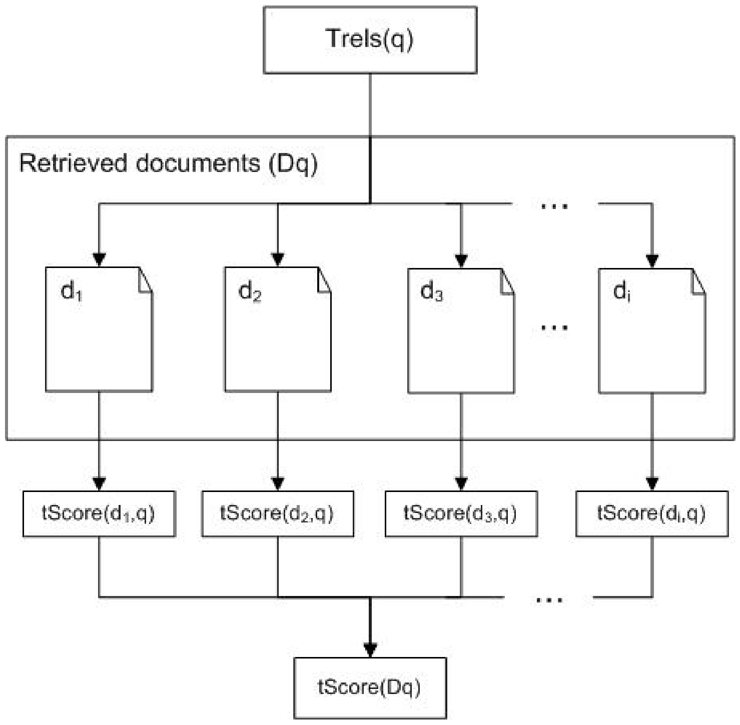


**Figure 4.3:** Document level scoring ($tScore(d_i, q)$) and collection scoring ($tScore(D_q)$)

There are two different ways of calculating *tScore* for each document; equation 4.1 or equation 4.2. The first scheme is for query $q$ the weighted difference between the number of onTopic and offTopic terms appearing in document $d$.

$$tScore_{basic}(d, q) = \mid t \in \mathit{onTopic} \cap d \mid \; - \beta \times \mid t \in \mathit{offTopic} \cap d \mid \qquad (4.1)$$

The second method for calculation of $tScore(d, q)$ is called the *Similarity scheme* and requires some IR resources which can measure the cosine similarity between the examined results and the term vectors induced from the Trels.

$$tScore_{sim}(d, q) = cos(onTopic, d) - \beta \, cos(offTopic, doffTopic, d)) \qquad (4.2)$$

The two schemes are highly correlated according to the developers of the Trels-based evaluation methodology. There were found through experiments a correlation of 0.991, and they emphasize that the choice of scheme depends on facilities supported by the IR environment [1].

The constant $\beta$ is set according to how heavily the offTopic terms should be weighted. $\beta = -1$ basically make them count as onTopic terms. I choose to set $\beta = 1$ which weights onTopic and offTopic terms equally, and yields the highest correlation with Qrels [1].

To calculate the final score for a result set $D_q$ for query $q$ either equation 4.3 or equation 4.4 can be used. These two differs in how they take the document ranking into account. Equation 4.3 will put less weight on the ones ranked low, and high weight on the first ones. Equation 4.4 simply calculates the mean values of all the $tScore(d, q)$. Both ways of calculating total score among document retrieval are explored later.

$$tScore(D_q) = \frac{\sum_{i=1}^{n} \frac{1}{i} tScore(d_i, q)}{\sum_{i=1}^{n} \frac{1}{i}} \tag{4.3}$$

$$tScore@k(D_q) = \frac{1}{k} \sum_{i=1}^{k} tScore(d_i, q) \tag{4.4}$$

**Own considerations**

Its been chosen to normalize $tScore(d, q)$ on the *size* (number of words in the body of the document) of the respective document. For example if a document consists of 100 words, 10 onTopic terms and 0 offTopic terms, we would get a $tScore_{basic}$ of 10. If we duplicate the body of the document the number of onTopic terms will be duplicated as well, and $tScore_{basic}$ will be 20. A normalization on document size would result in a equal $tScore_{basic}$ for both the original one and the duplicated. If $w$ is any word in document $d$ the general normalization follows equation 4.5 below.

$$tScore_{norm}(d_i, q) = \frac{1}{\mid w \in d_i \mid} tScore_{basic}(d_i, q) \tag{4.5}$$

### 4.3.4 Exploration of Trels

There are several issues according to use of Trels that have to be discovered through experiments since no such results have been published. The most important constants are related to the offTopic and onTopic-term sets. How many terms in the two sets are necessary? Experiments with different numbers of terms will be compared before a conclusion can be drawn. Another issue is the use of the measure unit tScore. What tScore can be expected from a good retrieval, a less good retrieval, or a really bad retrieval? And finally we will discuss whether its possible to decide a threshold which determines the retrieval as either relevant or not relevant.

The following tests are executed without any influence of DotLucene, neither any other retrieval system. More precisely this means that the top-ranked documents, which are used to calculate $tScore(D_q)$, are set up manually and will be fictive retrieval results. When a fictive top-ranked-list consists of only relevant documents we can get an idea of $tScore(D_q)$-values if the IR system only returned highly relevant documents. Likewise the fictive retrieved documents were set with only *irrelevant* documents and finally a set of fifty-fifty relevant and irrelevant documents.

For the experiments its used two different test cases. This is done both to have a wider basis for the conclusions and to test two different collections according to their nature of content.

### Case 1

The relevant and irrelevant documents for case 1 is collected through a third-party search engine on the Web. The retrieved documents collected from the approximately hundred top-ranked documents. Either they were found relevant or irrelevant. Totally ten relevant documents and ten irrelevant documents were collected.

For case 1 both the relevant and irrelevant documents is closely related to each other. For this reason they probably use same terminology and therefore harder to separate as irrelevant or relevant. The query

```
"hvordan oppdra barn" (in english: how to raise children)
```

results in irrelevant documents that's either about raising other things that children, animals mostly. Irrelevant documents were also about subjects related to children in general, diseases for example.

| | |
|---|---|
| Query | hvordan oppdra barn |
| Relevant documents | 10 |
| Irrelevant documents | 10 |
| Average document size | 5.9kB |
| Document subjects | Tutorials on how to raise kids |
| Test purpose | Separate a subject inside a domain |
| Kinds of onTopic and offtopic terms | The offTopic terms are that is inside the domain(raising and children) but not the wanted subject inside the domain. The onTopic terms are related to the wanted subject inside the domain |

**Case 2**

This case uses a shorter query and the collection of irrelevant documents differs totally from the relevant. In contrast to case 1, where the irrelevant documents are likely to be closely related to the relevant, the irrelevant documents of case 2 should be easier to separate from the relevant ones since they, with small probability, use the same terminology (terms). The query

```
"springer" (in english: bishop (in chess))
```

gave irrelevant documents mostly about dog breeds (springer spaniel). There were also collected irrelevant documents that were about workout (”springer” has a second meaning ”running” in Norwegian). Relevant documents were picked from web pages containing chess rules, match reports and tutorials.

| | |
|---|---|
| Query | springer |
| Relevant documents | 10 |
| Irrelevant documents | 10 |
| Average document size | 8.5kB |
| Document subjects | Chess rules, tips and tricks, match logs |
| Test purpose | Differentiate documents from different domains |
| Kinds of onTopic and offtopic terms | The offTopic terms are easy to confuse with the subjects closely related to the query. The onTopic terms are related to chess in general |

**Experiment procedure**

First all documents needed for the test runs where collected. Documents of different sizes retrieved by Google using the suffix ”site:no” in order to retrieve Norwegian documents[2], where categorized either relevant or not relevant.

For each test case there were created a Trels with on/off topic sets of 20 terms each. For each increment of number of terms tested, there were included another term from the Trels. For example when Case 1 is running the experiment with on/off topic set of 1 term, only the first onTopic term and the first offTopic term were used from the Trels with 20 terms that belongs to Case 1. The Trels-definitions for both cases can be found in appendix E.

Three different experiments where $tScore(D_q)$ were calculated for each case using the same Trels, both equation 4.3 and 4.4:

[2]Norway have two official languages; Bokmål and Nynorsk. The second official language ”Nynorsk” is a much smaller language and directly translated it means ”New Norwegian” (even if its older). All documents and Trels used in experiments of this chapter are written in Bokmål

1. Collection of relevant documents
2. Collection of irrelevant documents
3. Collection of irrelevant and relevant documents

In parallel to all executions there were done a experiment trying to eliminate two factors; *number of terms used* and *human assessment when selecting terms to be used.* These results are supposed to show the effect of eliminating human influence of term order selection. For each number of terms test $x = \{1, 2, ..., 20\}$ there were executed $N = 200$ iterations. For each iteration there were picked $x$ term pairs (both from onTopic and offTopic set) randomly without repetition. With $T = 20$ equation 4.6 describes the calculation for each term number.

$$\overline{tScore(D_q)_x} = \frac{1}{N}\sum_{i=1}^{N} tScore(D_q)_x \quad x = \{1, \ldots, T\} \tag{4.6}$$

Using the evaluation framework implemented and described in chapter 6 the results were plotted.

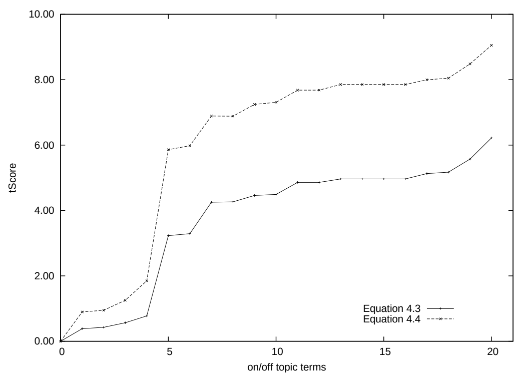
### 4.3.5 Results

This section investigates Trels-based evaluation measurement calculations in practice for case 1 and case 2. The raw result material is organized according to goals and areas to be explored.

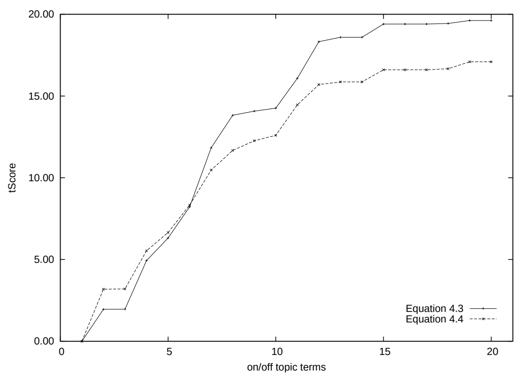
#### Evaluating query results

When a query is submitted into an IR system a set of documents is returned. Earlier figure 4.3 illustrated how each of the documents were given a score $tScore(d, q)$ and how all these score were used to calculate a total score $tScore(D_q)$ for the result of the query. This section shows the result of this process as numbers of terms varies in the on/off topic terms sets. Section 4.3.3 investigates two different equation used for scoring. Equation 4.1 is used to calculate scores for each document. In addition to vary number of on/off topic terms, two ways of calculating total score of all documents returned to a given query are plotted. As earlier mentioned, equation 4.3 aggregates the document scores with higher weighting of the top-ranked. Equation 4.4 is only the mean calculation.

Figure 4.4 shows the results of case 1 and case 2 for the respective **relevant** document collection. What we see is that increasing use of terms increases the total score. This means that if we increase the number of terms, we gain the confidence that the document collection of being relevant. Since we know that the documents are relevant, this is an expected outcome.



(a) Case 1



(b) Case 2

**Figure 4.4:** Calculation of result list score $tScore(D_q)$ on relevant documents

If we take a look at the difference between equation 4.3 and 4.4 for both cases in figure 4.4, we can observe something interesting. For case 1 we see that equation 4.4 gives higher score than equation 4.3, and for case 2 its the opposite (when using more than 6 terms). Equation 4.3 weights the top-ranked documents heavier than lower ranked documents. This means that if we use equation 4.3 we get an idea of how well the documents are ranked. Figure 4.4 tells us that the document ranking is less good in case 1, but seem to be slightly better in case 2.

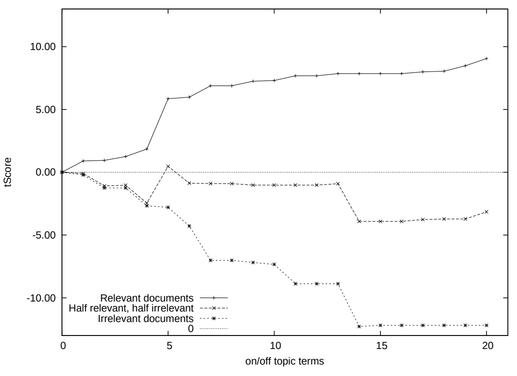
Lets say we have two documents $d_1$ and $d_2$. Each document is first given an individual score. $tScore(d_1, q) = 4$ and $tScore(d_2, q) = 2$ for query $q$. Calculating the mean score of those would not be affected by ranking; $(4 + 2)/2 = 3$ and $(2 + 4)/2 = 3$. Now we choose to weight the first ranked document by 1 and the second by $\frac{1}{2}$. If $d_1$ with the highest score 4, is ranked on top we get; $(1 \cdot 4 + \frac{1}{2} \cdot 2)/2 = 2.5$. On the other hand, if $d_1$ is ranked below $d_2$ we get; $(1 \cdot 2 + \frac{1}{2} \cdot 4)/2 = 2$.

In this way does equation 4.3 show how well ranked the documents are. Back to figure 4.4 we now see that the documents of case 1 are better ranked than the documents of case 2.

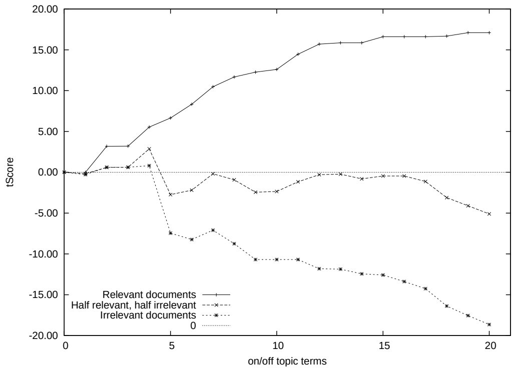
For later experiments its preferred to get a general impression of the top $k$ ranked documents. Therefore its chosen to use equation 4.4 which calculates the mean score of the top $k$ ranked documents. This choice will make it easier to compare results with common measurements. Precision and recall calculations count how many of the $k$ top ranked documents that are relevant, and not where in the result list they occur. If its desired to measure the ranking more in detail, its common to vary $k$. It could be 5, 10, or sometimes 100. If $k = 10$ for example gives high precision and $k = 5$ gives relatively low precision, this tells that the ranking should be improved.

## Number of terms needed

This section has the goal of determining how many terms that should be included in the on/off topic term sets. Again, experiments with different numbers of terms were executed, but this time also on documents of different levels of relevancy. An evaluation method should be able to detect both relevant and irrelevant document retrievals. The evaluation is executed on three different document collections; one consisting of only relevant documents, one only irrelevant documents, and the last with half relevant and half irrelevant. This should give an idea of how many on/off topic terms we need in order to determine relevancy or irrelevancy.



(a) Case 1



(b) Case 2

**Figure 4.5:** Experiments on relevant, irrelevant and fifty-fifty document collection

Figure 4.5 shows that less than four terms does almost give the same score for relevant, irrelevant, and the mixed collection of documents. Using four terms would probably result in "random" relevance judging of document retrievals. As we see, after 5-6 terms *tScore* for relevant documents increases while for irrelevant documents *tScore* decreases. And using 10 terms would make the distinction clear.

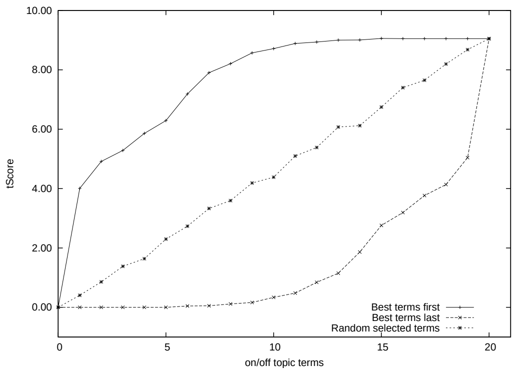


**Figure 4.6:** Changing priority of terms

When an "expert" uses domain knowledge to create Trels, she would probably write down the first terms that comes to her mind first. This may result in a set of terms with decreasing degree of importance. Previously, we found that 10 terms is enough to determine relevancy of a document collection. Now we are trying to change the order of choosing terms, so that the most important terms are applied first. Additionally, the inverse order were plotted, so the terms come in increasing degree of importance. Would the optimal choice of terms change the number of terms needed? This question is now to be answered.

Figure 4.6 shows that if the most significant terms are chosen first, we reach the maximum level of score at 10 terms. The original term set, where "own knowledge" were used to pick terms, also ended up with the same conclusion. This tells us that using an experts knowledge worked as good as forcing the best terms to come first. The second plot in figure 4.6, where the best terms are picked last, show that after 10 terms we can hardly tell if these documents are relevant or not.

### 4.3.6 Discussion

The past experimentation has been focusing on exploring constants related to Trels sets, methods used for evaluation and values to expect from the evaluation process.

The constant exploration focusing on finding how many terms needed in the on/off topic sets. We have seen that different terms affect the score in different degree. Some of the terms even do not affect the score in any degree at all. This means that the ones that does not affect the score, does not have to be present in the term sets. The relevance score seem to converge between 10-20 terms, but the score can also change by adding a 20th term. Despite this it seems like the relevance judgment would be the same whether 10 or 20 terms is used, hence 10 terms seem to be a natural choice since collecting terms is rather time consuming.

Two different ways of calculating the total score $tScore(D_q)$ for the whole retrieval list were also tried out. They have been referred to as equation 4.3 and 4.4. Its a fact that equation 4.3 takes the ranking into account. The highly ranked documents are weighted more that the lower ranked documents. The effect of weighting highly ranked documents is that irrelevant documents can be retrieved, but as long as they are ranked low it does not affect the score significantly.

If its demanded few relevant documents the weighted aggregation of $tScore(D_q)$ should be preferred. If we want to know if all the documents in the top $k$ ranked documents are relevant, the equation calculating the mean value should be used.

Experiments on both relevant and irrelevant documents results in positive and negative $tScore(D_q)$, respectively. It seems natural to set the threshold of relevancy to zero; negative score is irrelevant and positive is relevant. This also seems reasonable since positive score means more onTopic terms than offTopic terms.

Its also important to use the right method when calculation of the whole retrieval list according to the specific needs. If the needs are few relevant document and the mean *tScore* for the retrieval list is calculated. The result can misleading since irrelevant documents in the list that's lower ranked will lead to negative score.

In comparison to the TREC measures its been observed that documents may be judged differently due to the nature of the methods. TREC would judge a document relevant if only a small part of the document answers the query. Trels-based evaluation would also judge the document as relevant if not the other parts of the documents consist of major portion of offTopic terms.

### 4.3.7 Conclusion

Section 4.3.2 describes that the goals of this chapter were to find out how many terms in the onTopic and offTopic sets required in order to perform a reliable relevance judgment. There were executed several evaluation processes on different fictive retrievals for two different cases. The cases were selected in order to perform experiments on different types of document collections. During the experiments its been discovered several properties of Trels-based evaluation. The experiments show that the use of 10 terms, both onTopic and offTopic, is adequate for a reliable relevancy judgment.

The score threshold for relevancy is set to zero where negative score is irrelevant and positive is relevant. In every case the person responsible for the evaluation have to decide what criteria that's behind relevant or not relevant. The method used for calculation should be chosen out of the specific needs in every case.

# Part III

# Implementation

Chapter 5

# Implementation of Norwegian Analyzer

## 5.1 Introduction

This chapter describes the implementation of the NorwegianAnalyzer module. As described in section 3.6, analyzers in Lucene is used to perform linguistic operations on documents before indexing or searching. StandardAnalyzer which is included in the Lucene library performs language independent operations on documents as shown in figure 5.1. NorwegianAnalyzer is uses two additional operations for the Norwegian language compared to StandardAnalyzer. This chapter will describe these two processes and the additional standard operations included.

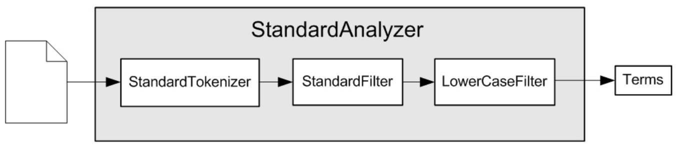


**Figure 5.1:** Components of default StandardAnalyzer

## 5.2 Components

NorwegianAnalyzer is basically a collection of five text processing operations. In Lucene, the processing operations are called *filters*. Text processed through NorwegianAnalyzer is manipulated by five filters. Three filters are used from the DotLucene library and two are implemented for the Norwegian language. Figure 5.2 shows the three standard filters and the two Norwegian specific filters which are shaded.

Table 5.1 contains detailed information about all filters used. Note that StandardTokenizer has the whole text as input argument. The remaining filters receives a token

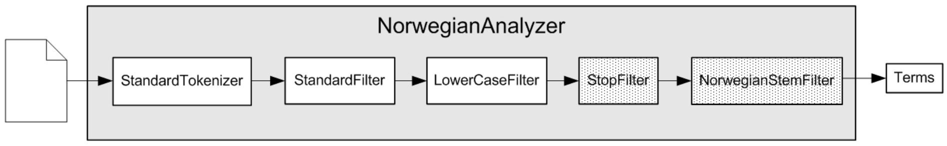


**Figure 5.2:** Components of NorwegianAnalyzer

stream and passes the stream over to the next filter. As mentioned, text are divided into pices of word during tokenization. As a result, the document is transferred into a token stream where each word in the text is a token in the stream.

| Class name | Arguments | Description |
|---|---|---|
| StandardTokenizer | Text | Converts the text to a stream of tokens |
| StandardFilter | Token stream | Removes dots from acronyms and 's from words with apostrophe. |
| LowerCaseFilter | Token stream | Case folding |
| StopFilter | Token stream, stopword list | Eliminates common words |
| NorwegianStemFilter | Token stream | Suffix stripping (stemming) |

**Table 5.1:** Components of Norwegian Analyzer in executed order

Figure 5.3 shows how text is treated in the analyzer. First text is divided into tokens, and after filtering they are converted to terms. A token is called a term when it is associated with a field. Fields are earlier discussed in section 3.3.

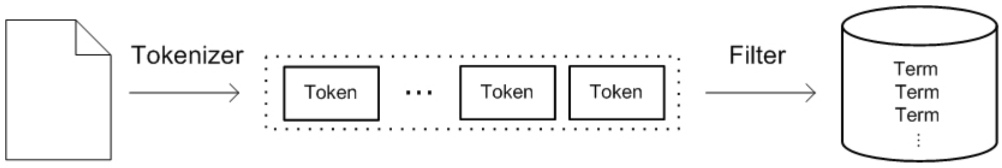


**Figure 5.3:** Documents are first converted to a stream of tokens, filtered and conflated into terms in index

The three first operations, StandardTokenizer, StandardFilter, and LowerCaseFilter simply perform the text processing described in table 5.1. StopFilter which eliminates stopwords, is used from the DotLucene library with information of stopwords for the Norwegian language. It is given a file containing the stopwords as an argument. As commented before, Norway have two official languages. The stopword list supports both languages.

NorwegianStemFilter is implemented for this thesis with use of design patterns from other stem filter for DotLucene. NorwegianStemFilter uses a class called NorwegianStemmer which contains the stemming algorithm for both Norwegian official languages. Figure 5.4 shows all relevant classes and relations. Classes which are adapted for Norwegian are shaded.

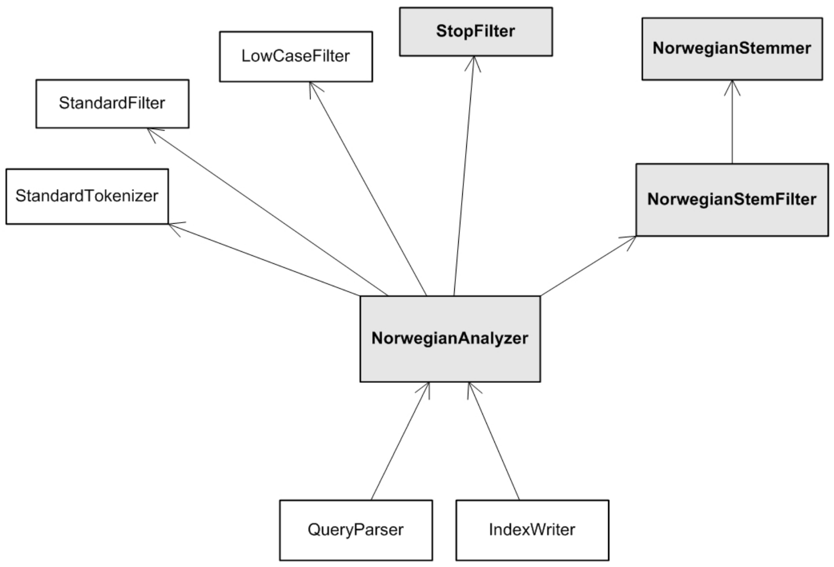


**Figure 5.4:** Relevant classes and their relations

## 5.3 NorwegianStemmer: Stemming using Snowball

Snowball is a resource maintained by M.F. Porter and contains stemming algorithms for several languages including Norwegian. The algorithms are written in a language called Snowball [10]. All snowball stemmers are based on the Porters stemming algorithm from 1980 [17].

The core of the Snowball's Norwegian stemming algorithm[1] is quite simple. First a well defined suffix region of a word is determined. Secondly the region is checked against a set of rules, and then the suffix region manipulated according to the matches found. The manipulation can either be deletion or substitution of one or several letters.

Norwegian has three additional letters which also are vowels. Note the vowels of the Norwegian alphabet below:

```
a e i o u y ø æ å
```

All stemmers described with Snowball use at least one of two regions called R1 and R2[2]. Norwegian stemmer use only R1 and is defined by

> "R1 is the region after the first non-vowel following a vowel, or is the null region at the end of the word if there is no such non-vowel" [17].

[1]http://snowball.tartarus.org/algorithms/norwegian/stemmer.html (18-05-2006)
[2]http://snowball.tartarus.org/texts/r1r2.html (18-05-2006)

For example the word "beautiful" would have R1 "iful";

```
 b   e   a   u   t   i   f   u   l
                   |<------------->|    R1
```

Before we can go on with the steps of the algorithm, one more definition is needed; only valid s-ending if "s" is proceeded by

```
 b c d f g h j l m n o p r t v y z
```

or **k** if its not preceded by a vowel.

Now we have all definitions needed to run the algorithm which consists of three steps. Each step is done for all words. Note that the description below is not written in the language Snowball, its rather

1. Search for the longest among the following suffixes in R1, and perform the action indicated.

   (a) **a e ede ande ende ane ene hetene en heten ar er heter as es edes endes enes hetenes ens hetens ers ets et het ast**
   → delete

   (b) **s**
   → delete if preceded by valid s-ending.

   (c) **erte ert**
   → replace with **er**

   *Note; the letter of the valid s-ending is not necessarily in R1*

2. If the word ends **dt** or **vt** in R1
   → delete the **t**

3. Search for the longest among the following suffixes in R1
   **leg eleg ig eig lig elig els lov elov slov hetslov**
   → delete

## 5.4 Verification of NorwegianStemmer

After implementing Porters stemming algorithm for Norwegian, it was desired to verify that NorwegianStemmer performs stemming according to the specification. To ensure correctness there were created an *NUnit test* for the stemmer. NUnit is a unit-testing framework for .NET. It simply make it possible to check if a unit, typically a method or function, gives the correct output to given arguments.

The unit test uses NorwegianStemmer to stem a given word from a list[3] and compares

[3]http://snowball.tartarus.org/algorithms/norwegian/diffs.txt (20-05-2006)

the result with the correct stem of the word. A list of totally 20635 words are stemmed and compared with the prespecified stemmed equivalent. Does all words pass the test, the stemmer is assumed to be implemented correctly.

Table 5.2 shows some words from the file containing the records used for verification. The word on the left side is stemmed by NorwegianStemmer and the result is compared with the correct equivalent on the right side.

| | |
|---|---|
| `adgang` | `adgang` |
| `adgangen` | `adgang` |
| `adkomst` | `adkomst` |
| `adkomstdokument` | `adkomstdokument` |
| `adkomstdokumenter` | `adkomstdokument` |
| `adkomsten` | `adkomst` |
| `adlyde` | `adlyd` |
| `adlyder` | `adlyd` |

**Table 5.2:** Sample words with their stemmed equivalent

NorwegianStemmer were developed until all words passed the test successfully. Approximately, a day or two were spent on getting all steps of the algorithm to perform correctly.

# Chapter 6

# Implementation of Trels-based evaluation

## 6.1 Introduction

This chapter will describe the implementation of Trels-based evaluation methodology described in section 4.3. This implementation can be used to evaluate any IR system as long as the retrieved documents are located in a filesystem and Term Relevance Sets are available.

## 6.2 Evaluation procedure

As mentioned, Trels-based evaluation judges relevancy based on the content of documents. Therefore there is functionality for loading the content of the retrieved documents. Since the content of documents is not stored in the index, there is functionality for loading and parsing the content of document. It possible to get a document's location in the filesystem. This path is used when loading and parsing the content of documents. There is support for parsing Microsoft Word documents, Adobe PDF, HTML, and plain text.

Figure 6.1 shows the evaluation procedure. First the documents are parsed (they can be of any of the previously given formats). After the document is parsed it is browsed through for topic terms defined in a Trels. When all topic terms are counted in the document content, the document is given a relevance score. Finally, when all document returned for a given query is given an individual score, a total score for all document is calculated. This is the final score for the query these documents are supposed to answer.

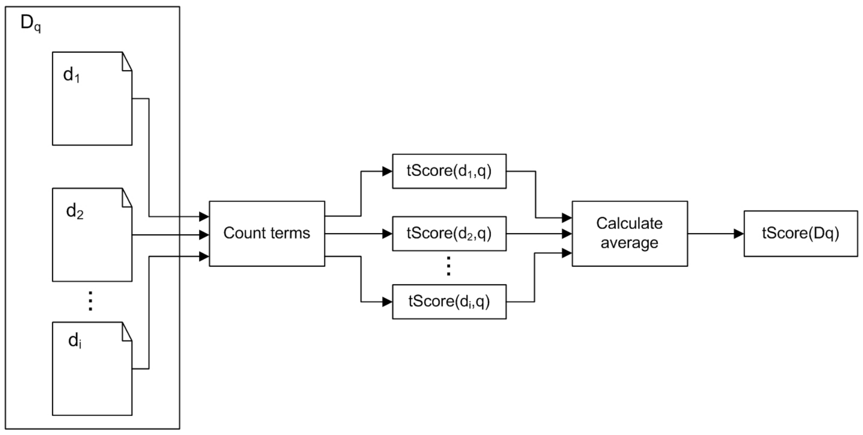


**Figure 6.1:** Calculating score for a collection of retrieved documents $D_q$

## 6.3 Open and parsing content of documents

In order to browse content of document, they need to be opened and parsed. It is added several parser so that different formats can be parsed and browsed. Figure 6.2 shows that after using a format specific parser, the content is give to the evaluation process as a string.

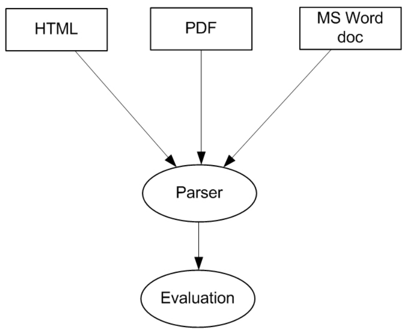


**Figure 6.2:** Parsing supporting different document formats.

## 6.4 Counting terms

For a given Trels all documents are treated one at a time. For each document the content is browsed for occurrences of on/off topic terms. It is chosen to count words that contains the term as well. This has advantages and disadvantages. It is particularly an advantage in relation to the Norwegian language since compound words are quite common. The disadvantage is that opposite meanings will be conflated; "visible" would match "invisible".

Below you see the code used to count terms. This is for a given Trels referenced to by the variable ”trels”. First the onTopic set is iterated and the total number of terms found are stored in the variable ”numOn”. The same is done for the offTopic set of terms. The variable ”text” is the content of a given document. At last the individual document score is calculated.

```
foreach (string term in trels.getOnTopics())
{
       MatchCollection matches = Regex.Matches(text, term);
       numOn += matches.Count;
}
foreach (string term in trels.getOffTopics())
{
       MatchCollection matches = Regex.Matches(text, term);
       numOff += matches.Count;
}

double tScore_basic = (numOn - ( BETA * numOff ));
```

The score of this document is after counting stored and another document is browsed. Finally, all scores are averaged and the total score is returned, as shown in figure 6.1 above.

Chapter 7

# Implementation of Trels-based evaluation framework for DotLucene

## 7.1 Introduction

The Trels-based evaluation framework uses Trels-based evaluation methodology implemented in chapter 6. This framework performs automatic evaluation of DotLucene. The evaluation process uses a set of pre-specified Trels-definitions, requests the IR system DotLucene to answer a query and evaluates the retrieved documents. Figure 7.1 shows an conceptual overview of the framework.

This chapter describes how Trels-based evaluation was realized through the framework created. The IR system was implemented using indexing and search functionality from the DotLucene toolkit.

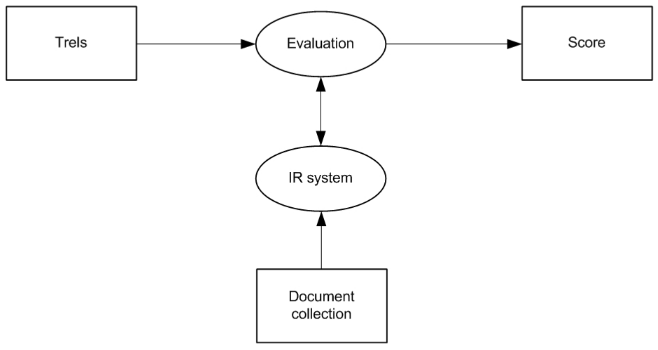


**Figure 7.1:** Trels-based evaluation framework overview

## 7.2 Components

The framework consists of two parts; an IR system and the evaluation part. Both parts needs one external resource each. The IR system needs access to a document collection which is indexed and made searchable. The evaluation part needs one or more Trels-definitions. For each Trels the query is first given to the IR system for searching, then the on/off topic terms are used to evaluate the ranked list of retrieved documents returned for the respective query. Figure 7.2 illustrates the important processes and their relations.

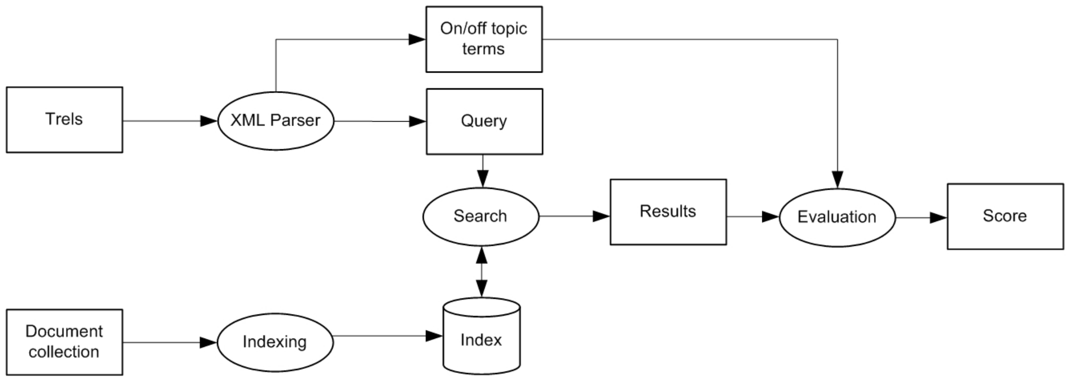


**Figure 7.2:** Trels-based evaluation framework processes

Its chosen to distinguish the IR system and the process of evaluation. This emphasizes that the evaluation does not depend on what IR system that is used. The evaluation process considers the IR system as a black-box which answers queries. This means that it practically only knows about queries and documents, and its purpose is to evaluate how well the documents are ranked. The two following sections describes the two parts and their respective components.

### 7.2.1 IR system based on DotLucene

This is the part actually evaluated. It is a minimal implementation using basic indexing and search functionality from the DotLucene library. An IR system has many opportunities for tuning the library so it performs at its maximum; such as intelligent decisions according to user context and the environment. Such functionality is not added in this turn. Only basic parameters are set appropriately so that lexical analysis and filters are suitable for Norwegian. Specifically, the implementation of NorwegianAnalyzer represents the largest change of parameters. Except this, all other parameters are kept default.

Table 7.1 contains the most important classes used, their arguments and a short description.

| Class name | Arguments | Description |
|---|---|---|
| DotLucene.IndexWriter | Analyzer | Indexing document collection |
| DotLucene.IndexSearcher | DotLucene.Query | Searches the index based on a query |
| DotLucene.QueryParser | Query string from a given Trels, Analyzer | Creates a query |
| DotLucene.Hits | | Collection of retrieved documents |
| NorwegianAnalyzer | Stopword list | Used by IndexWriter and QueryParser to analyze text |

**Table 7.1:** Associated classes

### 7.2.2 Evaluation of IR system based on DotLucene

The evaluation part encapsulates the IR system and performs necessary tasks. Initially it builds the index. For large document collections this may be time consuming. Unless any parameters of the filters needs to be changed, the index does not have to be rebuilt. If additionally documents needs to be added, these can be added to an existing index. After the index is built, searches are performed and the respective results are evaluated and a final score is returned. The process of evaluation for a given Trels follows these steps:

1. Load the Trels and extract the query and the two term sets (onTopic and offTopic).
2. Search the index using the query $q$.
3. For each $k$ top ranked document hit (according to a cut-off point $k$) do
   (a) Load content of document.
   (b) Count onTopic terms in document.
   (c) Count offTopic terms in document.
   (d) Calculate document score ($tScore_{basic}$).
4. Calculate score ($tScore(D_q)$) for all documents retrieved as a result for query $q$.

Most tasks as part of the evaluation is done by the class TrelsEval. In addition to the IR system, it uses tools for parsing documents in order to load the content and perform term counting (on/off topic terms). The IFilter[1] parser used supports several document formats, including PDF, DOC (Microsoft Word), HTML and plain text. TrelsXmlReader parses Trels-definitions in XML-format and creates object instances of RelevanceSet. They most relevant classes are listed in table 7.2.

[1]http://www.codeproject.com/csharp/IFilter.asp

| Class name | Arguments | Description |
|---|---|---|
| TrelsEval | Analyzer, Document collection, Trels collection | Performs Trels-based evaluation (indexing, searching and evaluation of ranked documents) |
| RelevanceSet | Query, OnTopic terms, OffTopic terms | Container for a query and its respective on/off topic terms |
| TrelsXmlReader | XML definition of a Trels | Parses and validates Trels from XML |
| IFilter.Parser | Parsers documents. Used to scan document contents | |

**Table 7.2:** Components of Trels-based evaluation framework

Overall the implementation is pretty straight-forward. Most of the choices made are related to the methodology itself, and therefore left to section 4.3 which discusses the Trels-based evaluation approach.

# Part IV

# Results

# Chapter 8

# Results

## 8.1 Introduction

The focus of this chapter is to evaluate DotLucene using NorwegianAnalyzer. Trels-based evaluation framework is executed using DotLucene with different analyzers; first with NorwegianAnalyzer and then two variants of StandardAnalyzer.

This comparison tells if the use of NorwegianAnalyzer makes DotLucene retrieve documents of higher relevancy to a given query. Comparison with StandardAnalyzer is done because it was previously used by InfoFinder, the initiator for this thesis.

## 8.2 Evaluation procedure

The evaluation procedure consist of three main steps; indexing, searching and evaluation. As mentioned earlier, analysis is used both during indexing and during searching (see section 3.6). Since we are evaluating several analyzers, both indexing and searching have to be done for each analyzer evaluated. These is the procedure steps done

1. Index document collection with each of the analyzers.
2. Create collection of Term Relevance Sets (Trels).
3. Search using all analyzers on respective indices.
4. Perform evaluation.

Figure 8.1 shows the evaluation steps for two analyzers. Note how "Document collection" and "Trels collection" are shared resources. For each Trels in the Trels collection a query is extracted. For each Trels-query $q$ a set of documents $D_q$ is retrieved by the IR system, which is given a $tScore(D_q)@k$ for a given cutoff point $k$. The total

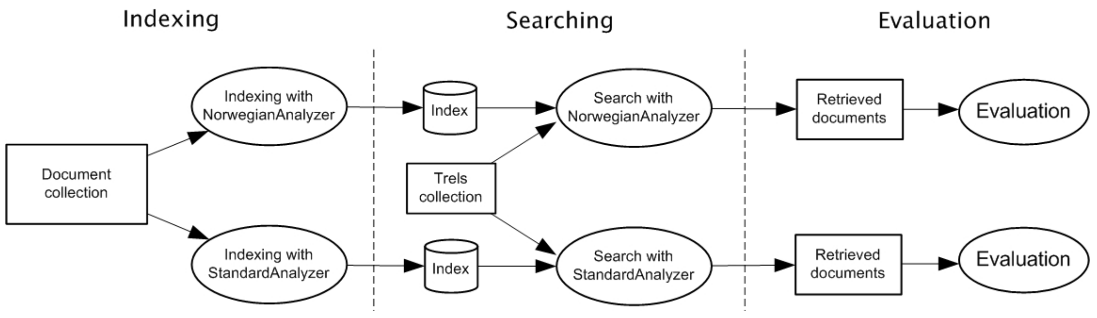


**Figure 8.1:** Abstract evaluation procedure for NorwegianAnalyzer and StandardAnalyzer.

evaluation score for the whole system S is called $tScore[@k](S)$, and is the average of all $tScore(D_q)$ from each query or Trels.

## 8.3 Deciding cutoff point

When a given query is submitted, the IR system returns varying number of documents. If many documents, lets say 100, are returned as a result for a query, the IR system must decide how many of those that should actually be presented to the user. Its very unlikely that the user bother to read the summaries of all results, unless they are few. Many search engines chose to present a given number of top ranked documents, which are most likely to capture the users attention.

When evaluating IR system its often taken into account that users, most likely, only see a given number of top ranked documents. Therefore the evaluation is on the basis of the $k$ top ranked documents, which is called the *cutoff point*. Sometimes evaluation is also done with several different cutoff points, 5, 10 or 100 is often found in evaluation results. Precision measure would then be marked with an '@' and the cut off point $k$ (P@5, P@10 or P@100).

In this evaluation its chosen to use cutoff point $k = 10$. This means that the 10 documents that DotLucene consider the most relevant ones are evaluated, both for precision calculations and Trels-evaluation.

## 8.4 Document collection

The document collection consists of all news articles from Adresseavisen[1] 2003 and 2004. Totally, the collection consist of 100601 articles (155 MB) with an average of 244 words per document. The indexing process took roughly 1 hour and 25 minutes. All benchmark data can be found in appendix D.

It was observed that some of the articles were typically "front page" notes. Such articles

[1]http://www.adressa.no

consists of a headline and a reference to the page where the article can be found in the paper, such as "Saddam Hussein is now captured. See page 13". Since these articles does not contain any significant topic information, there were set a boost factor lower than one to documents containing less than 15 words. Larger documents kept the default boost factor 1.0 (see table 8.1). The consequence is that these "front page" notes are given lower score and therefore not ranked so high. See section 3 for boosting details.

| Words in document | Boost |
|---|---|
| less than 5 | 0.3 |
| less than 10 | 0.6 |
| less than 15 | 0.9 |
| 15 or more | 1.0 |

**Table 8.1:** Boost factors for documents of different size

## 8.5 Term Relevance Sets

Totally 10 queries with appropriate term sets were created. As previously mentioned, the document collection consists of news articles from 2003 and 2004. Queries were picked based on the biggest news events during those years, both international news and news from Norway. Five from 2003 and five from 2004.

The process of creating Trels is quite time consuming (roughly 30-60 minutes each), but having the framework ready, creation of Trels is mainly all manual work needed for a large scale evaluation on a document collection. Based on the results of section 4.3 its chosen to use 10 onTopic and 10 offTopic terms in each Trels.

Every term in the sets have a potential to affect the total score. There is always a chance for including terms that is not a true on/off topic term for the respective query. This can result in Trels that does not perform the correct relevance judgment. Using several queries with related Trels and calculate the average would even out misjudgments and give a more correct picture of a systems performance. This is also emphasized by the authors of Trels-based evaluation:

> "Individual terms are almost never absolute indicators of relevance or lack thereof. The power of the evaluation lies in the aggregation of term appearances across many documents returned for many queries" [1].

The time dedicated to create Term Relevance Sets for this evaluation was quite limited, so in the end it were used totally 10 queries and for each query there were created a Trels. The Trels collection used for evaluation is added in appendix F. They are written in Norwegian, so for non-Norwegian readers, they can be browsed to notice the structure, query length or just observe the basis for the evaluation.

## 8.6 Results

This section will present the results from the evaluation process on DotLucene using three levels of analysis.

- StandardAnalyzer doing
  - Tokenization with cleverness for alphanumerics, acronyms, company names, e-mail addresses, numbers and words with apostrophe.
  - Case folding (lowercasing).
- StandardAnalyzer* doing the same as StandardAnalyzer + eliminating stopwords.
- NorwegianAnalyzer doing the same as StandardAnalyzer + eliminating stopwords + stemming

This approach will show how effectiveness is affected by first eliminating stopwords and secondly eliminating stopwords and stemming. Its expected that stopword elimination increases the effectiveness or score and applying stemming is expected to increase effectiveness even more. For later discussions there were also calculated *precision* in addition to $tScore$. For each of the three analyzers all documents retrievals, totally 300 documents, were judged manually in order to calculate $P@10$ for each query/Trels. Table 8.2 shows $tScore(D_q)@10$ and $P@10$ for each query/Trels as they appear in appendix F.

| | StandardAnalyzer | | StandardAnalyzer* | | NorwegianAnalyzer | |
|---|---|---|---|---|---|---|
| Query/Trels | $tScore(D_q)$ | $P@10$ | $tScore(D_q)$ | $P@10$ | $tScore(D_q)$ | $P@10$ |
| 1 | 2.4 | 1.0 | 2.1 | 1.0 | 1.6 | 1.0 |
| 2 | 10.5 | 0.9 | 6.2 | 0.9 | 10.1 | 1.0 |
| 3 | 3.2 | 0.9 | 7.2 | 0.9 | 11.6 | 0.9 |
| 4 | 4.4 | 0.9 | 6.1 | 1.0 | 4.5 | 1.0 |
| 5 | -0.1 | 0.2 | 0.7 | 0.4 | 2.6 | 0.6 |
| 6 | 0.4 | 0.2 | 0.5 | 0.1 | 5.6 | 1.0 |
| 7 | 3.1 | 1.0 | 2.1 | 1.0 | 4.4 | 1.0 |
| 8 | 3.7 | 1.0 | 4.2 | 1.0 | 4.5 | 1.0 |
| 9 | 9.6 | 1.0 | 10.9 | 1.0 | 10.6 | 1.0 |
| 10 | 5.4 | 1.0 | 8.4 | 1.0 | 6.1 | 0.9 |
| Average | 4.26 | 0.81 | 4.84 | 0.83 | 6.16 | 0.94 |

**Table 8.2:** Evaluation results of StandardAnalyzer with and without stopword elimination, and NorwegianAnalyzer

The average over all queries shows effectiveness performance for the whole IR system S, where S is varied using different analyzers. The system performance shows an increased effectiveness using stopword elimination, and a significant higher effectiveness using both stopword elimination and stemming. Table 8.3 shows increased score compared

| Increased effectiveness using | $tScore(S)$ | $P@10(S)$ |
|---|---|---|
| Stopword elimination | 13.6% | 2.5% |
| Stemming | 27.3% | 13.3% |
| Stopword elimination and stemming | 44.6% | 16% |

**Table 8.3:** Increased effectiveness using different analysis operations

to StandardAnalyzer. Remember that using stopword elimination *and* stemming equals NorwegianAnalyzer.

These results seem realistic compared to published results for other European languages. At CLEF 2003 it was reported results for 9 European languages where Porter's Snowball stemmers where compared with no stemming. For Swedish it was reported that P@10 increased with 18.1% with use of Snowball, and for German it was reported an increase of 12.6% [25]. These languages are quite related to Norwegian so they can give a hint about how well the results achieved in this thesis are. The evaluation of Norwegian Snowball stemmer done here shows an increased P@10 of 13.3%.

At the Royal Institute of Technology in Stockholm, Sweden, it has been published results for a self-created stemmer. They found that P@10 increased with 15% using their stemming algorithm compared to no stemming. This also shows that the achieved results here are reasonable [6]. It did not succeed to find published results for using stemming on Norwegian.

## 8.7 Discussion

The main goal of this chapter is to present results of using DotLucene with Norwegian analysis compared with a standard approach. The comparison is done using evaluation based on term relevance sets. In addition the traditional measure *precision* were calculated. This section will discuss some important issues related to information retrieval and evaluation done. Primarily, the focus of the results is limited to

- Using DotLucene as retrieval machinery.
- Advantage of using analysis for Norwegian.
- Evaluation based on Trels.

DotLucene have proved to be reliable and easy to use throughout the evaluation phase. The boosting property of documents came in handy when the problem of "small" documents were discovered. The highly flexible and modular architecture made it easy to run evaluation with different analyzers. The benchmark details in appendix D seems relatively similar to others found at The Apache Software Foundation[2] according to performance.

[2]Other benchmarks can be found at http://lucene.apache.org/java/docs/benchmarks.html

Using analysis for Norwegian shows in general a significant increase of retrieval effectiveness, with background on both Trels-based evaluation and precision measurement. Table 8.2 show for some queries the opposite; that analysis *decreases* effectiveness. For example for query/Trels number 2 we see that stopword elimination affects $tScore$ negatively, while stemming again increases $tScore$. Such behaviour occurs for several queries both for stemming and stopword elimination. Despite these few exceptions the overall effect of using analysis for Norwegian seem to be positive.

The Trels-based evaluation methodology was explored in section 4.3, and proved to be useful for relevance evaluation of information retrievals. In previous section we saw that deploying different analyzers resulted in different $tScore$s. Since $tScore$-scoring is quite unknown, precision calculations were applied so the measurement could be compared in some way. When looking at all results from all three analyzer variants, we see that low $tScore$ can give high precision; average precision corresponding to $tScore > 1.6$ is 0.956 and average precision corresponding to $tScore < 1.6$ is 0.225. This is an interesting observation if we look at the nature of the two approaches:

- For **precision** calculations an human assessor judges a document as relevant if it in some way is relevant to the query.
- **Trels-based evaluation** says that a document is relevant if it consists of pre-specified terms.

Since Trels-based evaluation estimates relevancy based on content, high $tScore$-results necessary does not mean higher "relevancy" in the way we measure precision. For example if document $d_1$ receives an estimated $tScore$ of 5.0, and document $d_2$ have a $tScore$ of 10.0 for a given query. Assuming that a document is relevant as long as its related to the query, we could say that $d_1$ and $d_2$ are equally relevant, but $d_2$ probably consists of more relevant content to read. The definition of relevancy is of course an important issue here. To summarize we can say that

- If a relevant document should contain a significant amount of text related to a topic
  - Trels-based evaluation returns high values of $tScore$.
  - For precision measurement, the human assessor judges documents relevant if they satisfies the minimum level of content.
- If a relevant document should just answer the query (necessarily just a sentence)
  - Trels-based evaluation can be used by setting a *low* threshold for $tScore$.
  - For precision measurement, the human assessor judges documents relevant if they just answer the query.

These two requirements scenarios emphasizes an advantage of Trels-based evaluation. Change of requirement to document relevancy would just introduce an change of $tScore$ threshold for Trels-based evaluation, while precision measurements requires re-judging of documents.

## 8.8 Conclusion

The results show a significant increase of effectiveness of the IR system using NorwegianAnalyzer compared to StandardAnalyzer. Totally precision increased with 16% and *tScore* had an increase of 44.6%. This increase comes both from elimination of stopwords and stemming.

The use of Trels-based evaluation showed high correlation with precision measurements, and there were discovered that *tScore* values are relatively low even for relevant documents. As an feature, *tScore* can be used to indicate greater amount of relevant content.

# Chapter 9

# Conclusion and further work

The work of this thesis has consisted of implementing an Norwegian Analyzer and a framework for evaluation of DotLucene. Evaluation of standard methods and the improved Norwegian Analyzer showed that use of stopword elimination and stemming increases the retrieval performance of DotLucene for information retrieval in Norwegian text documents. The improvements implemented in Norwegian Analyzer increased precision with 16%. Evaluation with Term Relevance Sets gave an increased relevancy of 44%. Both measures indicates that InfoFinder who initially desired the the improved analyzer should, for Norwegian documents, use Norwegian Analyzer together with their existing IR system based on DotLucene.

Evaluation based on Term Relevance Sets were explored and implemented. The evaluation method is easy to use for any IR system, but the current implementation only performs automatic evaluation for DotLucene based IR systems. The Term Relevance Sets were created with 10 offTopic terms and 10 onTopic terms, which during exploration showed to be enough to judge the relevancy of documents. Occurrences of these terms in a document content either increases or decreases the confidence of this document to be relevant for a given query. Totally, there were used 10 such queries with belonging Term Relevance Sets. The average of each query result gave the total score of the performance of DotLucene and Norwegian Analyzer.

The exploration of Trels-based evaluation done has given indication of that the methodology performs reliable relevance judging of retrieved documents. The two cases experimented with was supposed to challenge the evaluation methodology differently. Still, there should be done experiments on several cases to increase the confidence of that Trels-based evaluation performs reliable relevance judgments. Due to the time limitation of this thesis this is left for future work.

Experiments with selecting the 10 most important terms among a set of 20, showed that the 10 least important did not affect the relevance score in any significant degree. Future work could be focusing on ways of measuring the quality of terms included in on/off topic terms. It was also chosen to use the same number of onTopic terms and offTopic terms. It would be useful to find out if the number of terms required are different for the onTopic and offTopic set. The published work of Trels-based evaluation uses significantly fewer term in the offTopic set than in the onTopic set. Does this affect the identification of irrelevant documents, or should it be prioritized to gather onTopic terms to identify relevant documents? This could also be a subject for further experimentation of Trels-based evaluation.

# Part V

# Appendices

# Appendix A

# Fields in Lucene

| Field method/type | Analyzed | Indexed | Stored | Example usage |
|---|---|---|---|---|
| Field:Keyword(String, String) Field.Keyword(String, Date) | | x | x | Telephone and Social Security numbers, URLs, personal names, Dates |
| Field.UnIndexed(String, String) | | | x | Document type(PDF, HTML and so on), if not used as a search criteria. |
| Field.UnStored(String, String) | x | x | | Document titles and content |
| Field.Text(String, String) | x | x | x | Document titles and content |
| Field.Text(String, Reader) | x | x | | Document titles and content |

**Table A.1:** An overview of different field types, their characteristics, and their usage [12]

# Appendix B

# Query subclasses in Lucene

| | |
|---|---|
| TermQuery | The most elementary way to search an index for a specific term. TermQuery is case-sensitive. Useful for retrieving documents by key; ISBN-numbers for example. |
| RangeQuery | Facilitates searches from a starting term through an ending term. |
| PrefixQuery | Matches documents containing terms beginning with a specified string. |
| BooleanQuery | A container of boolean clauses. A clause is a subquery(any of the other query subclasses) that can be optional, required or prohibited. These attributes allow for logical AND, OR and NOT combinations. |
| PhraseQuery | Using the positional information about terms in the index, terms within a certain distance of each other can be located. The maximum allowable positional distance between terms to be considered a match is called *slop*. |
| WildcardQuery | Used to query for terms with missing pieces but matches in some degree. * is used for zero or more characters. ? for zero or one character. |
| FuzzyQuery | Matches terms *similar* to a specified term. Uses *Levenshtein distance* algorithm to determine how similar terms in the index are to a specified target term. |

**Table B.1:** Query subclasses in Lucene [12]

# Appendix C

# Common words in the Norwegian language[1]

| | | | |
|---|---|---|---|
| og | i | jeg | det |
| at | en | et | den |
| til | er | på | de |
| med | han | av | ikke |
| ikkje | der | så | var |
| meg | seg | men | ett |
| har | om | vi | min |
| mitt | ha | hadde | hun |
| nå | over | da | ved |
| fra | du | ut | sin |
| dem | oss | opp | man |
| kan | hans | hvor | eller |
| hva | skal | selv | sjøl |
| her | alle | vil | bli |
| ble | blei | blitt | kunne |
| inn | når | kom | noen |
| noe | ville | dere | som |
| deres | kun | ja | etter |
| ned | skulle | denne | for |
| deg | si | sine | sitt |
| mot | å | meget | hvorfor |
| dette | disse | uten | hvordan |
| ingen | din | ditt | blir |
| samme | hvilken | hvilke | sånn |
| inni | mellom | vår | hver |
| hvem | vors | hvis | både |
| bare | enn | fordi | før |

[1]http://snowball.tartarus.org/algorithms/norwegian/stop.txt

| | | | |
|---|---|---|---|
| mange | også | slik | vært |
| være | båe | begge | siden |
| dykk | dykkar | dei | deira |
| deires | deim | di | då |
| eg | ein | eit | eitt |
| elles | honom | hjå | ho |
| hoe | henne | hennar | hennes |
| hoss | hossen | ingi | inkje |
| korleis | korso | kva | kvar |
| kvarhelst | kven | kvi | kvifor |
| me | medan | mi | mine |
| mykje | no | nokon | noka |
| nokor | noko | nokre | sia |
| sidan | so | somt | somme |
| um | upp | vere | vore |
| verte | vort | varte | vart |

# Appendix D

# Indexing benchmark

**Hardware environment**

| | |
|---|---|
| Dedicated machine for indexing | Yes |
| CPU | Pentium4 @ 2.4 GHz |
| RAM | 512 MB |
| Drive configuration | IDE |

**Software environment**

| | |
|---|---|
| DotLucene Version | 1.4.3 |
| Compiler | Visual Studio C# 2005 Express |
| OS Version | Microsoft Windows XP Service Pack 2 |
| Location of index | Filesystem |

**DotLucene indexing variables**

| | |
|---|---|
| Number of source documents | 100601 |
| Total filesize of source documents | 155 Megabyte |
| Average filesize of source documents | 1.7 KB |
| Source documents storage location | Filesystem |
| File type of source documents | Plain text |
| Analyzer used | NorwegianAnalyzer (Snowball) |
| Number of fields per document | 3 |
| Type of fields | 1 text, 1 keyword, 1 unstored |
| Index persistence | FSDirectory |
| Index size | 60 MB |

**Figures**

| | |
|---|---|
| Time taken (in ms/s as an average of 3 indexing runs) | 1 hour 25 minutes |
| Time taken / 1000 docs indexed | 50 seconds |
| Memory consumption | 65 000 KB |

# Appendix E

# Term Relevance Sets used for exploration of Trels-based evaluation

## Case 1

```
<?xml version="1.0" encoding="UTF-8"?>
<Trels>
    <query>hvordan oppdra barn</query>
    <offTopic>
        <term>dyr</term>
        <term>hund</term>
        <term>hest</term>
        <term>valp</term>
        <term>føll</term>
        <term>sykdom</term>
        <term>vannkopper</term>
        <term>underholdning</term>
        <term>bøker</term>
        <term>svangerskap</term>
        <term>adhd</term>
        <term>røde hunder</term>
        <term>meslinger</term>
        <term>feber</term>
        <term>bleier</term>
        <term>kosthold</term>
        <term>gravid</term>
        <term>skillsmisse</term>
        <term>barneklær</term>
        <term>separasjon</term>
    </offTopic>
    <onTopic>
        <term>barneoppdragelse</term>
        <term>mamma</term>
        <term>pappa</term>
        <term>autoritet</term>
        <term>kjærlighet</term>
        <term>grenser</term>
        <term>straff</term>
        <term>respekt</term>
        <term>familie</term>
        <term>vilje</term>
        <term>sinne</term>
        <term>gi etter</term>
        <term>konflikt</term>
        <term>misunnelse</term>
        <term>forklare</term>
        <term>irettesette</term>
        <term>streng</term>
        <term>snill</term>
        <term>utvikling</term>
        <term>ungdom</term>
    </onTopic>
</Trels>
```

## Case 2

```
<?xml version="1.0" encoding="UTF-8"?>
<Trels>
    <query>springer</query>
    <offTopic>
        <term>trene</term>
        <term>jogge</term>
        <term>kondisjon</term>
        <term>arkiv</term>
        <term>hund</term>
        <term>sykle</term>
        <term>verlag</term>
        <term>spaniel</term>
        <term>engelsk</term>
        <term>kennel</term>
        <term>oppdrett</term>
        <term>trening</term>
        <term>mat</term>
        <term>axel</term>
        <term>løype</term>
        <term>skogen</term>
        <term>rase</term>
        <term>mosjon</term>
        <term>sykle</term>
        <term>fuglehund</term>
    </offTopic>
    <onTopic>
        <term>brettspill</term>
        <term>spill</term>
        <term>oppstilling</term>
        <term>angrep</term>
        <term>forsvar</term>
        <term>svart</term>
        <term>hvit</term>
        <term>løper</term>
        <term>konge</term>
        <term>dronning</term>
        <term>tårn</term>
        <term>bonde</term>
        <term>hest</term>
        <term>sjakkmatt</term>
        <term>matt</term>
        <term>rokkering</term>
        <term>taktikk</term>
        <term>motstander</term>
        <term>brett</term>
        <term>koordinat</term>
    </onTopic>
</Trels>
```

# Appendix F

# Term Relevance Sets used for evaluation

```
<Trels id="1">
    <query>irak krig</query>
    <offTopic>
        <term>kuwait</term>
        <term>valg</term>
        <term>vietnam</term>
        <term>gulfkrigen</term>
        <term>gresshopper</term>
        <term>ugler</term>
        <term>verdenskrig</term>
        <term>korea</term>
        <term>japan</term>
        <term>afrika</term>
    </offTopic>
    <onTopic>
        <term>amerika</term>
        <term>george bush</term>
        <term>colin powell</term>
        <term>saddam hussein</term>
        <term>usa</term>
        <term>storbritannia</term>
        <term>baghdad</term>
        <term>angrep</term>
        <term>krig</term>
        <term>styrker</term>
    </onTopic>
</Trels>

<Trels id="2">
    <query>utbrudd av sars virus</query>
    <offTopic>
        <term>fugleinfluensa</term>
        <term>south african revenue service</term>
        <term>michael sars</term>
        <term>georg ossian sars</term>
        <term>peter christen asbjørnsen</term>
        <term>pollen</term>
        <term>samples of anonymised records</term>
        <term>allergi</term>
        <term>sola</term>
        <term>legionella</term>
    </offTopic>
    <onTopic>
        <term>hong kong</term>
        <term>kina</term>
        <term>virus</term>
        <term>influensa</term>
        <term>smitte</term>
        <term>severe acute respiratory syndrome</term>
        <term>spredning</term>
        <term>epidemi</term>
        <term>sykdom</term>
        <term>who</term>
    </onTopic>
</Trels>

<Trels id="3">
    <query>angrep på irak i 2003</query>
    <offTopic>
        <term>kuwait</term>
        <term>valg</term>
        <term>vietnam</term>
        <term>gulfkrigen</term>
        <term>gresshopper</term>
        <term>ugler</term>
        <term>verdenskrig</term>
        <term>øks</term>
        <term>korea</term>
        <term>afrika</term>
    </offTopic>
    <onTopic>
        <term>usa</term>
        <term>george bush</term>
        <term>colin powell</term>
        <term>saddam hussein</term>
        <term>invasjon</term>
        <term>storbritannia</term>
        <term>baghdad</term>
        <term>amerika</term>
        <term>krig</term>
        <term>styrker</term>
    </onTopic>
</Trels>

<Trels id="4">
```

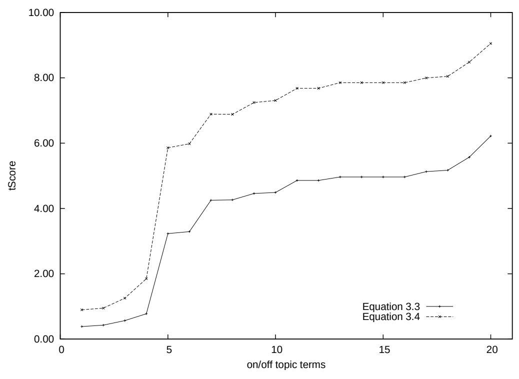
```
    <query>programmet idol på tv2</query>
    <offTopic>
        <term>billy</term>
        <term>sport</term>
        <term>fotball</term>
        <term>idrett</term>
        <term>golf</term>
        <term>tippeligaen</term>
        <term>eliteserien</term>
        <term>valgkamp</term>
        <term>kommunestyrevalg</term>
        <term>miljøvernpris</term>
    </offTopic>
    <onTopic>
        <term>kjartan salvesen</term>
        <term>popstjerne</term>
        <term>platekontrakt</term>
        <term>finale</term>
        <term>kurt nilsen</term>
        <term>gaute ormåsen</term>
        <term>rørlegger</term>
        <term>spektrum</term>
        <term>jorunn stiansen</term>
        <term>idol</term>
    </onTopic>
</Trels>

<Trels id="5">
    <query>drept på kjøpesenter i stockholm</query>
    <offTopic>
        <term>olof palme</term>
        <term>voldtekt</term>
        <term>gøteborg</term>
        <term>oslo</term>
        <term>skutt</term>
        <term>ulykke</term>
        <term>selvmord</term>
        <term>skjøt</term>
        <term>terror</term>
        <term>pistol</term>
    </offTopic>
    <onTopic>
        <term>anna lindh</term>
        <term> nk </term>
        <term>knivstukket</term>
        <term>mord</term>
        <term>utenriksminister</term>
        <term>statsråd</term>
        <term>sverige</term>
        <term>1957</term>
        <term>politiker</term>
        <term>mijailovic</term>
    </onTopic>
</Trels>

<Trels id="6">
    <query>flodbølgekatastrofe i asia</query>
    <offTopic>
        <term>fedafjorden</term>
        <term>tafjord</term>
        <term>ramnefjellet</term>
        <term>bødal</term>
        <term>ytre nesdal</term>
        <term>flybilletter</term>
        <term>næringsliv</term>
        <term>børs</term>
        <term>kina</term>
        <term>korea</term>
    </offTopic>
    <onTopic>
        <term>sør-asia</term>
        <term>bølge</term>
        <term>jordskjelv</term>
        <term>tsunami</term>
        <term>sørøst-asia</term>
        <term>katastrofe</term>
        <term>omkomne</term>
        <term>phuket</term>
        <term>døde</term>
        <term>savnet</term>
    </onTopic>
</Trels>

<Trels id="7">
    <query>rocknes ulykke</query>
    <offTopic>
        <term>exxon-valdez-ulykken</term>
        <term>mehamn-ulykken</term>
        <term>sleipner-ulykken</term>
        <term>bilulykke</term>
        <term>tsjernobyl-ulykken</term>
        <term>kursk</term>
        <term>nintendo</term>
        <term>emulator</term>
        <term>software</term>
        <term>freeware</term>
    </offTopic>
    <onTopic>
        <term>forliste</term>
        <term>druknet</term>
        <term>kantret</term>
        <term>vatlestraumen</term>
        <term>19. januar</term>
        <term>sjøforklaringen</term>
        <term>bergen</term>
        <term>spesialskipet</term>
        <term>havarerte</term>
        <term>omkom</term>
    </onTopic>
</Trels>

<Trels id="8">
    <query>munch ranet</query>
    <offTopic>
        <term>kiosk</term>
        <term>hotel continental</term>
        <term>leie ut</term>
        <term>munch-utleie</term>
        <term>refsnes gods</term>
        <term>nasjonalgalleriet</term>
        <term>bensinstasjon</term>
        <term>bankran</term>
        <term>thon hotel munch</term>
        <term>restaurant munch</term>
    </offTopic>
    <onTopic>
        <term>skrik</term>
        <term>madonna</term>
        <term>edvard munch</term>
        <term>munch-museet</term>
        <term>tøyen</term>
        <term>munch-ranet</term>
        <term>maleriene</term>
        <term>stavanger-ranet</term>
        <term>munch-maleriene</term>
        <term>munch-saken</term>
    </onTopic>
</Trels>

<Trels id="9">
    <query>knutby drapet</query>
    <offTopic>
        <term>lindh</term>
        <term>fadime</term>
        <term>ronald ramm</term>
        <term>fadime sahindal</term>
        <term>oluf palme</term>
        <term>hariri senior</term>
        <term>nordisk råds filmpris</term>
        <term>film</term>
        <term>roman</term>
        <term>meta-drapet</term>
    </offTopic>
    <onTopic>
        <term>helge fossmo</term>
        <term>kristi brud</term>
        <term>åsa waldau</term>
        <term>menighet</term>
        <term>sara svensson</term>
        <term>pastor</term>
        <term>barnepiken</term>
        <term>fossmo</term>
        <term>gränsta</term>
        <term>alexandra fossmo</term>
    </onTopic>
</Trels>

<Trels id="10">
    <query>siktede i nokas ranet</query>
    <offTopic>
        <term>noka milkshake-diett</term>
        <term>isys</term>
        <term>noka dietten</term>
        <term>landvetter</term>
        <term>verditransport</term>
        <term>tveita</term>
        <term>meta</term>
        <term>kløfta</term>
        <term>haugesund</term>
        <term>eldre kvinne</term>
    </offTopic>
    <onTopic>
        <term>stavanger</term>
        <term>norsk kontantservice</term>
        <term>gissel</term>
        <term>politidrap</term>
        <term>nokas</term>
        <term>skuddveksling</term>
        <term>rettsak</term>
        <term>toska</term>
        <term>schumann</term>
        <term>havnå</term>
    </onTopic>
</Trels>
```

Appendix G

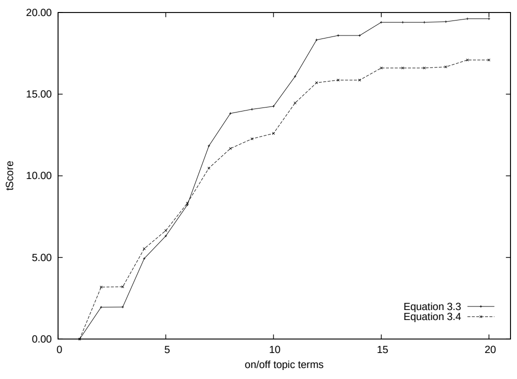
# Results of Trels-based evaluation experiments

As an supplement to section 4.3 some experiment results can be found in this appendix. The graphs shows results from the same cases described.

## Collection of relevant documents

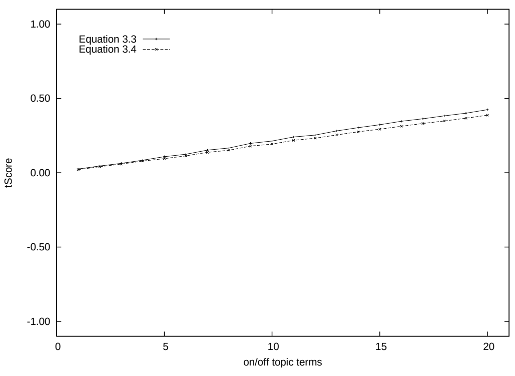


(a) Case 1

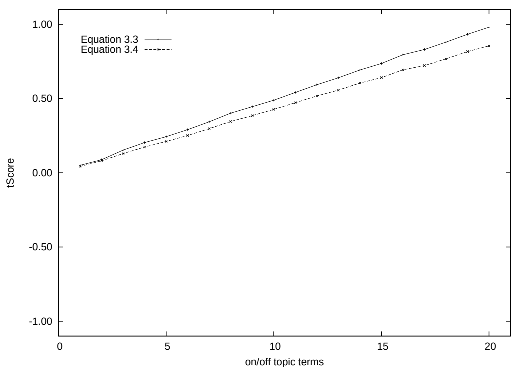


(b) Case 2

**Figure G.1:** Normalized for document size in relevant document



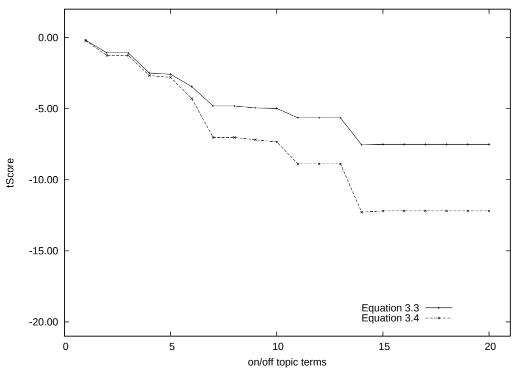
(a) Case 1



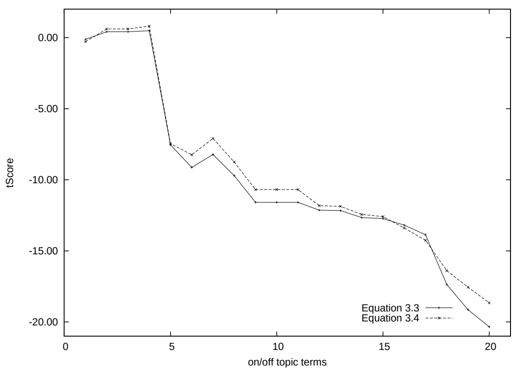
(b) Case 2

**Figure G.2:** Mean of random selected term pairs

## Collection of irrelevant documents

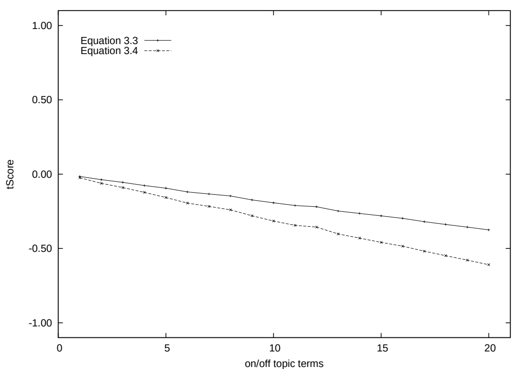


(a) Case 1

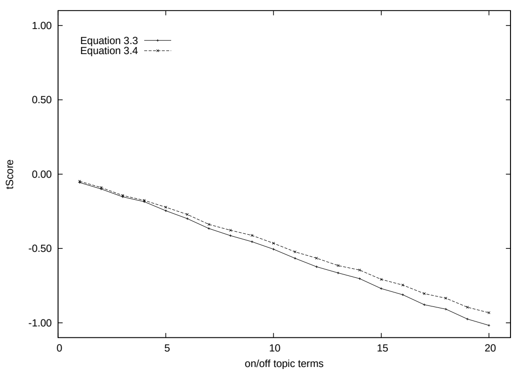


(b) Case 2

**Figure G.3:** Normalized for document size in irrelevant documents



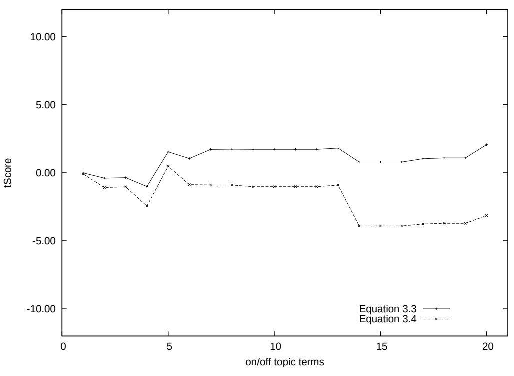
(a) Case 1



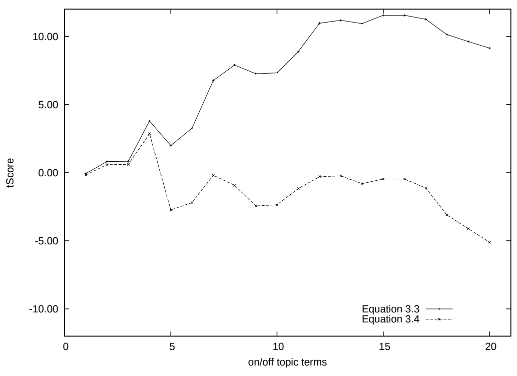
(b) Case 2

**Figure G.4:** Mean of random selected terms in irrelevant documents

## Collection of relevant and irrelevant documents

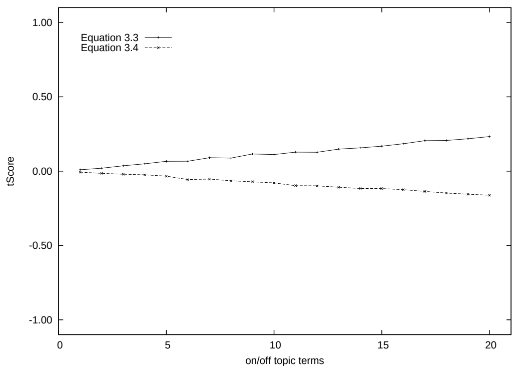


(a) Case 1

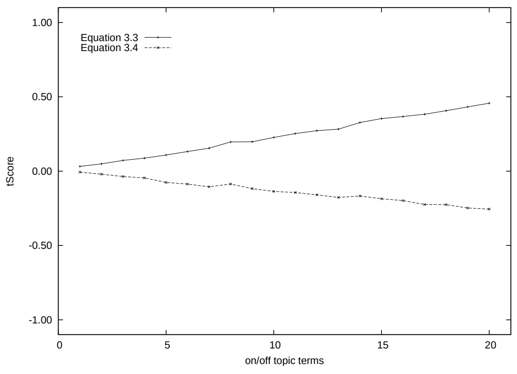


(b) Case 2

**Figure G.5:** Fifty-fifty relevant and irrelevant documents



(a) Case 1



(b) Case 2

**Figure G.6:** Mean of random selected terms in fifty-fifty relevant and irrelevant documents

# Appendix H

# Norwegian stemmer in Snowball[1]

```
routines (
           mark_regions
           main_suffix
           consonant_pair
           other_suffix
)

externals ( stem )

integers ( p1 x )

groupings ( v s_ending )

stringescapes {}

/* special characters (in ISO Latin I) */

stringdef ae   hex 'E6'
stringdef ao   hex 'E5'
stringdef o/   hex 'F8'

define v 'aeiouy{ae}{ao}{o/}'

define s_ending  'bcdfghjlmnoprtvyz'

define mark_regions as (

    $p1 = limit
```

[1]Porters stemming algorithm for Norwegian written in Snowball can be fount at http://snowball.tartarus.org/algorithms/norwegian/stemmer.html

```
    test ( hop 3 setmark x )
    goto v  gopast non-v  setmark p1
    try ( $p1 < x  $p1 = x )
)

backwardmode (

    define main_suffix as (
        setlimit tomark p1 for ([substring])
        among(

            'a' 'e' 'ede' 'ande' 'ende' 'ane' 'ene' 'hetene' 'en' 'heten' 'ar'
            'er' 'heter' 'as' 'es' 'edes' 'endes' 'enes' 'hetenes' 'ens'
            'hetens' 'ers' 'ets' 'et' 'het' 'ast'
                (delete)
            's'
                (s_ending or ('k' non-v) delete)
            'erte' 'ert'
                (<-'er')
        )
    )

    define consonant_pair as (
        test (
            setlimit tomark p1 for ([substring])
            among(
                'dt' 'vt'
            )
        )
        next] delete
    )

    define other_suffix as (
        setlimit tomark p1 for ([substring])
        among(
            'leg' 'eleg' 'ig' 'eig' 'lig' 'elig' 'els' 'lov' 'elov' 'slov'
            'hetslov'
                (delete)
        )
    )
)

define stem as (

    do mark_regions
    backwards (
        do main_suffix
        do consonant_pair
        do other_suffix
    )
)
```

# Bibliography

[1] Einat Amitay, David Carmel, Ronny Lempel, and Aya Soffer. Scaling ir-system evaluation using term relevance sets. Technical report, IBM Haifa Research Lab, Haifa, Israel, July 2004.

[2] Fast Search & Transfer ASA. *The Book of Search*, volume 1. First edition, 2006.

[3] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval.* ACM press, 1999.

[4] A. Bookstein and D.R. Swanson. Probabilistic models for automatic indexing. *Journal of the American Society for Information Science*, 26(1):45–50, September 1974.

[5] Vannevar Bush. As we may think. *Atlantic Monthly*, 176:101–108, July 1945.

[6] J. Carlberger, H. Dalianis, M. Hassel, and O. Knutsson. Improving precision in information retrieval for swedish using stemming. Technical report, Numerical Analysis and Computing Science, Royal Institute of Technology, Stockholm, Sweden, May 2001.

[7] N. Craswell, S. Robertson, H. Zaragoza, and M. Taylor. Relevance weighting for query independent evidence. In *SIGIR'05: Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 416–423. ACM Press, 2005.

[8] Doug Cutting, Julian Kupiec, Jan Pedersen, and Penelope Sibun. A practical part-of-speech tagger. In *Proceedings of the Third Conference on Applied Natural Language Processing*, pages 133–140, 1992.

[9] Douglass R. Cutting and Jan O. Pedersen. Space optimizations for total ranking. Technical report, Excite Inc. and Verity Inc., 555 Broadway, Redwood City, CA 94063.

[10] Snowball: A Language for stemming algorithms. *http://snowball.tartarus.org/*. (18-05-2006).

[11] Lucene Similiarity Formula. http://csunx4.bsc.edu/lucene/api/org/apache/lucene/search/Similarity.html. (06-06-2006).

[12] Otis Gospodnetic and Erik Hatcher. *Lucene in action.* Manning Publications, 2005.

[13] R. Krovetz. Viewing morphology as an inference process. In *Proceedings of the 16th ACM SIGUR Conference*, pages 191–202, 1993.

[14] M.E. Maron and J.L. Kuhns. On relevance, probabilistic indexing and information retrieval. *Journal of the ACM*, 7:216–244, 1960.

[15] The Essentials of Google Search. *http://www.google.com/help/basics.html.* (12-05-2006).

[16] TREC Overview. http://trec.nist.gov/overview.html. (01-05-2006).

[17] Martin F. Porter. An algorithm for suffix stripping. 14(3):130–130, 1980. See also http://www.tartarus.org/martin/PorterStemmer/def.txt.

[18] TREC Sample Relevance Judgments (Qrels). http://trec.nist.gov/data/robust/qrels.robust2004.txt. (5-06-2006).

[19] Evaluation Report. http://trec.nist.gov/pubs/trec14/appendices/ce.measures05.pdf. In *Text REtrieval Conference 2005.* (06-06-2006).

[20] S.E. Robertson. The probabilistic ranking principle in ir. *Journal of Documentation*, 33:294–304, 1977.

[21] S.E. Robertson and K. Sparck-Jones. Relevance weighting of search terms. *Journal of the Americal Society of Information Science*, pages 129–146, 1976.

[22] S.E. Robertson, C.J. van Rijsbergen, and M.F. Porter. Probabilistic models of indexing and searching. In *Proceedings of the 3rd annual ACM conference on Research and development in information retrieval*, pages 35–56. Butterworth & Co, 1980.

[23] G. Salton and M.J. McGill. *Introduction to Modern Information Retrieval.* McGraw-Hill Book Company, New-York, 1983.

[24] Gerard Salton, A.Wong, and C.S.Yang. A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620, November 1975.

[25] Stephen Tomlinson. Lexical and algorithmic stemming compared for 9 european languages with hummingbird searchserver at clef 2003. In *Cross-Language Evaluation Forum (CLEF) 2003*, July 2003.

[26] TREC Sample Topics. http://trec.nist.gov/data/topics_eng/topics.501-550.txt. (14-06-2006).

[27] TREC Test Questions (Topics). http://trec.nist.gov/data/testq_eng.html. (01-05-2006).

[28] T. Upstill. *Document ranking using web evidence.* PhD thesis, Australian National University, 2004.

[29] T. Upstill, N. Craswell, and D.Hawking. Query-independent evidence in home page finding. *ACM Trans. Inf. Syst.*, 21(3):286–313, 2003.

[30] Apache License version 2.0. *http://www.apache.org/licenses/LICENSE-2.0.txt*, 2004. (13-05-2006).

[31] Ian H. Witten, Alistair Moffat, and Timothy C. Bell. *Managing Gigabytes: compressing and indexing documents and images.* Morgan Kaufmann, second edition, 1999.