FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Tuning Retrieval System Effectiveness via the Retrievability Bias

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Medieninformatik

eingereicht von

Michael Alexander Kascha
Matrikelnummer 0548800

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Andreas Rauber

Wien, 04.05.2012

(Unterschrift Verfasser) (Unterschrift Betreuung)

FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Tuning Retrieval System Effectiveness via the Retrievability Bias

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Master of Science

in

Media Informatics

by

Michael Alexander Kascha
Registration Number 0548800

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Andreas Rauber

Vienna, 04.05.2012

(Signature of Author) (Signature of Advisor)

Erklärung zur Verfassung der Arbeit

Michael Alexander Kascha
Schlesienstraße 1, 89518 Heidenheim an der Brenz, Deutschland

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Ort, Datum)	(Unterschrift Verfasser)

Acknowledgments

First of all, I would like to thank Andreas Rauber for introducing me to the academic field of information retrieval research and his wonderful guidance during the creation of this work.

In the same way thanks to Shariq Bashir for the software tools, data and advice he provided me with. Without them, it would not have been possible to create this thesis.

Furthermore, I would like to thank my parents for their unquestioning financial and moral support, enabling me to pursue this study.

Special thanks go to Evelyn Koller, who was always someone I could absolutely rely on during my last years of study, for many lectures that were based on group work and beyond. It was a real pleasure to study with you, thanks for the good time!

Abstract

The performance of an information retrieval system depends on the document collection it has to work on and can be influenced to a certain degree by tuning the systems parameter values. There are several different measures to evaluate the effectiveness of retrieval systems. All of them have in common that they require human relevance judgments to determine if search results are satisfying the information need of the queries. However, their creation is both very expensive and time consuming.

There have been three fundamentally different approaches to create performance rankings without the need of these relevance judgments. This thesis first introduces relevant background information and then takes a closer look at the approach that shows the most potential in determining the top performers. It employs an unsupervised measure called retrievability which expresses how accessible documents are for the system. Experiments are conducted to compare various effectiveness measures to the retrievability bias at different system parameter values. This is done with three different retrieval methods on four different collections.

Results show that while there is usually always a general correlation of high effectiveness and low retrievability bias observable, it largely depends on the length and/or quality of the user queries issued. In some cases a minimal retrievability bias does not indicate a good parameter setting for maximizing the effectiveness values. There might be several reasons for this behavior which still require further research.

Kurzfassung

Die Leistung eines Information Retrieval Systems hängt von der zu bearbeitenden Dokumentensammlung ab und kann bis zu einem gewissen Ausmaß durch das Abstimmen seiner Parameterwerte beeinflusst werden. Es existieren unterschiedliche Maße, um die Effektivität von Retrieval Systemen zu beurteilen. Diese haben jedoch allesamt gemeinsam, dass von Menschen durchgeführte Beurteilungen der Relevanz nötig sind, um festzustellen, ob die Suchergebnisse die Informationsanforderungen der Suchanfragen befriedigen. Diese Beurteilungen sind jedoch mit einem hohem Kosten- und Zeitaufwand verbunden.

Es existieren drei fundamental unterschiedliche Ansätze, um Leistungsrankings zu erstellen, ohne dass dafür Beurteilungen der Ergebnisrelevanz nötig sind. In dieser Arbeit werde zunächst wichtige Grundlagen präsentiert und dann jener Ansatz weiter verfolgt, der das größte Potential zum Ermitteln der Top-Performer zu besitzen scheint. Dabei kommt ein unüberwachtes Maß zum Einsatz, welches sich Retrievability nennt und angibt, wie zugänglich Dokumente für ein bestimmtes System sind. Es werden mit unterschiedlichen Parametereinstellungen Experimente durchgeführt, um einige Effektivitätsmaße mit dem Bias der Retrievability bei denselben Parametern zu vergleichen. Dies geschieht für drei verschiedene Retrievalmethoden und vier Dokumentensammlungen.

Die Ergebnisse zeigen, dass fast immer eine generelle Korrelation zwischen hohen Effektivitätswerten und niedrigem Retrievability Bias zu beobachten ist. Allerdings hängt dies stark von der Länge und/oder der Qualität der vom Benutzer verwendeten Suchanfragen ab. In manchen Fällen weist ein minimaler Retrievability Bias nicht auf einen guten Parameterwert hin, um die Effektivität des Systems zu maximieren. Dieses Verhalten könnte mehrere Gründe haben, welche weiterführende Untersuchungen nahelegen.

Contents

CHAPTER 1

Introduction

Nowadays millions of people produce new media content every day. No matter if short micro blog entries, long articles, pictures, music or video content, they all face the same challenge: once published, can they also be found again by other users? Retrieval systems and their optimization for the specific content they are employed to find, play a vital role in this. Even the best content is virtually worthless if it can't be easily and correctly retrieved again.

1.1 Motivation

Evaluation is an important cornerstone of *Information Retrieval* (IR) research. Test document collections play a vital role in this process, as they offer a controlled setting for the evaluation and enable researchers to compare retrieval systems under similar conditions. However, due to the fact that the relevance of documents in respect to search topics need to be determined by humans, the construction of these collections is a very time consuming and expensive procedure. The *Text REtrieval Conference* (TREC) tries to cope with this problem by pooling the top 100 documents retrieved by each participant and only judging the relevance of them. While this leads to good results, it is still very costly. This raises the need to find alternative ways to determine and optimize the effectiveness of retrieval systems.

In the last years several different methods have been proposed that allow to create a ranking of retrieval systems without the need of these relevance judgments. There are three basic approaches. The first is to replace the human judgments by automatically created pseudo-relevance judgments. The second is to create a ranking based on the retrieval systems similarity. The third is to use the retrieval bias as a measure for ranking, as there is supposed to be some correlation between this and the systems effectiveness measures. Since the results of the first and second approach positively correlated with official TREC rankings but usually failed to properly identify the top performing systems, this thesis takes a closer look at the latter one.

1.2 Goal

The aim of this thesis is to investigate the following three questions:

- To what degree is there a relationship between effectiveness measures and the retrievability bias?
- Is this relationship consistent for different document collections?
- Is it possible to employ retrievability (as an unsupervised measure) to tune the parameters of a retrieval system in order to optimize its effectiveness measures?

1.3 Outline

This thesis is organized as follows:

Chapter 1 contains the motivation for this work, the goals it wants to achieve and this outline explaining the structure of the document.

Chapter 2 tries to give a definition what information retrieval is and what it is not. Then the basic theoretical background of information retrieval systems and related work get introduced, on which the practical part builds upon. This includes index creation, term weighting, the retrieval models employing these weights and ways to improve query generation. Then it is explained how the performance of a system can be evaluated by means of different measures, toolkits and test collections. Finally different approaches from other scientific publications are presented that offer possible starting

points for the experiments. The most promising one is selected, concluding the theoretical part.

Chapter 3 contains information about hardware and software used for the practical part, followed by detailed explanations of the experiment setups.

In the next chapter 4, all experimental results are analyzed in detail, interesting findings presented and possible relations discussed.

Finally, chapter 5 concludes the thesis by providing a comprehensive summary of achieved insights and an outlook on further possible work to be conducted in this field of study.

CHAPTER 2

Fundamentals

This section first tries to give a definition of what IR is and what it is not. Then generic concepts and problems of IR in computer science are presented upon which the following sections are based on. Starting with common IR tasks, where the ad hoc retrieval task is further elaborated. The steps needed in order to create an index are stated, term weighting schemes and their use in retrieval models explained, and possible ways to improve queries presented. Finally, the important area of evaluation gets examined, including different evaluation measures followed by important toolkits and standard test collections, which are often used to compute these.

2.1 Information Retrieval Basics

Information retrieval (IR), as an academic field of study, is a rather old and well established discipline and goes back to the late 1940s. It was originally an activity which only a few and mostly professional people concerned themselves with. The main focus used to be the retrieval of unstructured text, in fact in many cases the term 'information' could simply be substituted with 'document' when describing the process [58, pp. 1, 3].

In the last two decades, especially since the Word Wide Web gained popularity, this began to change. Nowadays, hundreds of millions engage themselves with IR every day in one way or another. IR has also overtaken the traditional form of information access, which used to be performing searches in some kind of structured database. The already

vast amount of information is growing fast one a daily basis and IR is now often also used to retrieve other data that goes beyond plain text, like content from multimedia archives.

This development and the new challenges surfacing with it, greatly amplified the interest in IR as a field of study. Already established methods get revisited and enhanced, new ones proposed, representation and structure of the information changed in order to improve automatic retrieval. Generally speaking, research in this field focuses on finding better methods to do the same. Evaluation always plays a key role in all of this [27, p. 1], [4], [31, pp. 1-2].

2.1.1 Definition

A general definition of IR is that from Baeza-Yates and Ribeiro-Neto [6, p. 1]:

> *"Information retrieval (IR) deals with the representation, storage, organization of, and access to information items. The representation and organization of the information items should provide the user with easy access to the information in which he is interested."*

The following is a more narrow definition of IR as an academic field of study by Manning et al. [31, p. 1]:

> *"Information retrieval (IR) is finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers)."*

2.1.2 Data versus Information Retrieval

The computer science disciplines data retrieval and information retrieval are closely related, therefore, their distinguishing characteristics and differences are presented here in order to further refine the IR definition.

A data retrieval system uses an artificial language with a rather restrictive vocabulary and syntax in order to retrieve data that has a well defined structure and semantics. The query is a complete specification of the data that should be retrieved where even a small error can lead to total failure of the process, as it aims for total matching where

the document has to satisfy every part of it. It provides a suitable solution for a database system where you enter clearly defined key terms like an identification code. Nonetheless, it does not solve the problem of retrieving information in a desirable way where the query only partly matches.

IR on the other hand normally uses natural language text for the query, which also can be incomplete. The goal of an IR system is not just to extract the information out of unstructured data, but in most cases also to decide how relevant regarding the query it is. This needs to be done, because unlike database searches, there is no clear hit and miss for the results in IR. Ideally, all relevant documents should be retrieved while retrieving as few as possible non-relevant ones, so the user can sift through the list of partly matching ranked results in order to easily pick the ones satisfying his information need [6, p. 2], [12, pp. 29-30].

2.1.3 IR Tasks

The most standard IR task for a system is to provide relevant documents from a collection that satisfy a user's indiscriminate information need. This is called *ad hoc retrieval*. The *information need* is the subject about which the user wants to gain additional knowledge. The user tries to communicate this to the system by issuing a one-time *query* of which he thinks it conveys his need in a suitable way. A document returned by the system is considered a *relevant* one if the user perceives that it contains information relevant to his initial information need [31, p. 5].

Other tasks are for example the retrieval of multimedia content like speech, music, images and videos. In *cross-language IR* the system tries to retrieve results in several languages for a query in one. Also routing, filtering and further processing of already retrieved sets of information are important tasks, too [31, pp. 2, 478].

However, the major part of this thesis focuses solely on the ad hoc retrieval task for documents.

2.1.4 Ad Hoc Retrieval Process

Figure 3.1 illustrates the basic and generic interaction steps and components of an ad hoc IR system.

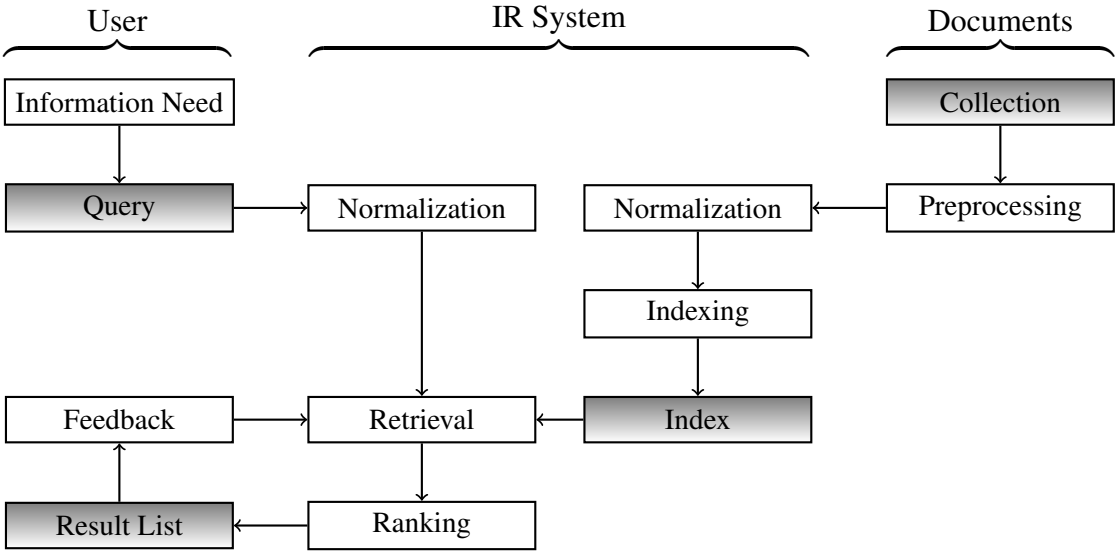


Figure 2.1: Generic information retrieval process adapted from Baeza-Yates and Ribeiro-Neto [6, p. 10].

First of all, a document collection is needed and has to be indexed. To do that, it is normally necessary to preprocess the collection first by converting it into a format which the indexer can parse. The indexer then chops the documents into tokens and optionally applies certain normalization operations like case-folding, stop word removal, equivalence classing, stemming and lemmatization to them. These tokens are then added to the index.

Once this is done, it is possible for a user to search documents. He has some information need which he wants to satisfy. The IR system cannot understand that imprecise need directly, so the user has to try to express it with a query and submit that to the system. Usually the same normalization methods used for the documents also get applied to the query, to allow correct matching. The IR system then retrieves documents that partially match the query. Most retrieval models have the ability to rank these documents now, by comparing the query with each of them and computing a similarity measure. Finally, a list of these ranked documents gets returned to the user. Optional some IR systems allow to improve the result list by giving a relevance feedback [6, pp. 9-10].

2.2 Index Creation

The most basic way to perform document retrieval is by simply performing linear scans on the documents. This method is also called *grepping*, named after the Unix command grep. For simple querying on small collections this is basically all you need, but there are several limitations to it [31, p. 19]:

- The performance of this method depends on the document collection, it diminishes when the collection grows in size.
- It does not work well with complex query matching operations.
- It does not allow ranked retrieval, which is needed to get the best answer to an information need.

In order to avoid grepping through all the documents for every query, a popular solution is to *index* them first. One possible way to do this would be to construct a term-document matrix in which every document gets a unique documentID and the mapping gets saved in a lookup table. The rows and columns of the matrix contain both vectors for each document, showing the terms it contains and vectors for each term, listing the documents it appears in. But it is obvious that for large collections with a usually also huge vocabulary, the dimensions of this matrix would be rather bulky. This is due to the fact that it would contain a lot of redundant information, like all these document-term combinations that do not occur and would have to be labeled with 0's.

This problem is usually solved via an *inverted index*, which is also the most efficient way for text applications, according to Witten et al. [61, p. 109]. The inverted index only stores term-document combinations which actually occur. It consists of two major parts: a dictionary and postings. The dictionary, which sometimes is also referred to as a lexicon, consists of all terms in the collection. For each of these terms a list is used, storing in which documents the term occurs in. Optional also the position in the document can be stored. This automatically stores how often the term in contained, as the amount of positions can simply be accumulated to obtain that information. Each item in such a list is called a posting, and the combination of all lists is referred to as postings. This basic concept is illustrated in the following figure 2.2.

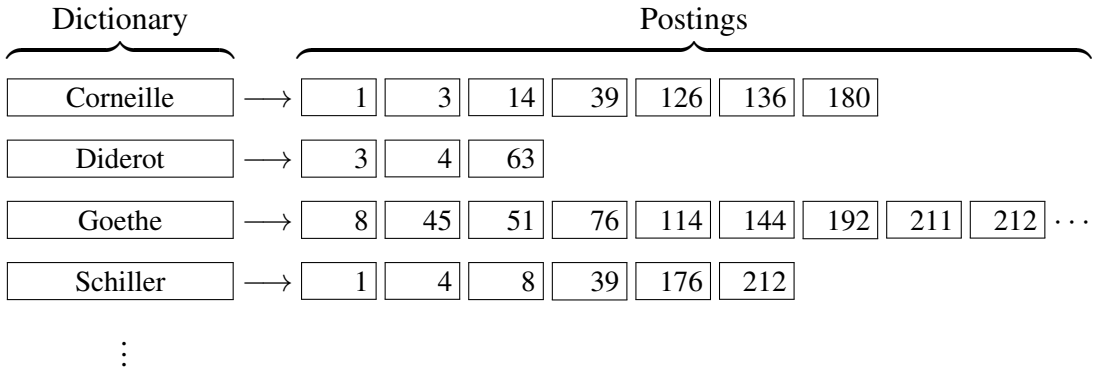


Figure 2.2: Components of an inverted index adapted from Manning et al. [31, p. 7].

To create such an inverted index, Manning at al. [31, p. 19] describe four major steps that are required:

- All documents that should be indexed need to be collected.
- Tokens from each documents text are extracted, creating a token list out of it.
- The indexing terms are created by normalizing these tokens with linguistic pre-processing.
- The inverted index is created by indexing the documents each term occurs in.

To further optimize this, the terms in the dictionary can be replaced by termIDs. This is done by assigning each term a unique number and creating a term-termID mapping, very similar to the way it is done for documents, too.

2.2.1 Document Collection

Basically any compilation of documents can be used here. To simplify the process and provide even testbeds, usually standard test collections are used in IR research. For further information regarding test collections, see chapter 2.7.

The main task in this step is to determine the *document unit* for indexing. One option might be to simply handle each file as a document, but that is not always the ideal solution. For example in Unix, email messages usually get stored in a single file, but it might make sense to handle each email as a separate document. Also for large

books, in most cases it will not be the best solution to handle the entire book as a single document. Separating it into chapters first could be the preferred way to index it. This issue is referred to as *indexing granularity*. Sometimes text that might commonly be regarded as a single document might be split into smaller files, too. For example when information is processed for the web and stored in various HTML files. In this case it is sensible to merge them into a single document. A good understanding of the document collection, the information need and the usage patterns is helpful to come to a decision regarding the document unit [31, p. 22].

2.2.2 Tokenization

Tokenization is the process of chopping the document units into smaller pieces. These are consequently called tokens. A *token* consists of a sequence of characters from the document, which form a semantic unit that is useful for further processing. The class of all tokens which contain the same sequence of characters is called a *type*.

In this step, certain characters like punctuation marks can be discarded. Some character combinations also provide a real challenge for the tokenization process. For example in the English language, apostrophes are used for different semantic purposes like the contraction of words or indicating possession. There are also several uses of hyphenation: grouping words, bonding nouns together as names or splitting up vowels in words. This shows that some tokenization issues are related to the language of the document, which consequently should be known to the tokenizer. For this purpose classifiers exist that recognize languages via character subsequences. Various other special character sequences exist that should be recognized as a single token. Popular examples are city names which sometimes consist of several words separated by white space, certain programming languages like C++ containing special characters, URLs for web pages, IP or email addresses which partially consist of punctuation marks and some other unusual characters.

Other languages add new issues that need to be addressed. In German compound nouns are written without spaces in between, so the tokenizer needs to try to split them into single tokens which appear in a vocabulary. This is usually done with a so called *compound-splitter*. This problem gets even more complicated in some Asian languages like Chinese, Korean and Japanese. In these, text is written without any additional white

spaces to separate the words, so *word segmentation* has to be performed first. This is a source for errors, as it is not always clear where their word boundaries are [31, pp. 22-28].

2.2.3 Normalization

Before the tokens get added to the IR system's dictionary, various normalization processes can be applied to remove superficial differences and merge several tokens into a single one. In most cases this helps improving match finding with user queries. For this purpose, the same normalization steps that are applied for index generation are usually also applied to the queries.

Case-folding

A very common normalization approach is to reduce all letters to lower case, also known as *case-folding*. This way a token that begins with a capital letter, because its position is at the beginning of a sentence, gets folded to the same spelling as it would have in the middle of a sentence. This change also brings some downsides with it, as some information gets lost this way. It will not be possible any more to distinguish between common and proper nouns and acronyms might get merged with a similar written word, although they got a completely different meaning. For English texts, an alternative to prevent this information loss would be to use *true-casing*. This only reduces these tokens to lower case that are at the beginning of a sentence or either completely or mostly uppercase because they are part of a headline. Since most users do not pay attention to proper capitalization and submit their queries all in lower case, simply case-folding everything is usually the best solution [31, p. 30].

Stop Word Removal

Extremely common terms are usually of little help to distinguish and select these documents, which satisfy the user need. These terms are referred to as *stop words*, because text processing stops when one is encountered and they are not added to the dictionary. The usual approach to building a *stop list*, is by sorting the terms by their collection frequency and add the most common ones to the list. *Collection frequency* is the accumulated number of times, each term appears in the entire document collection. Often

it also makes sense to manually edit the stop list with the purpose to adjust it to the semantic domain of the documents. One advantage of using a stop list is the drastically reduced number of postings that needs to be stored in the index [p. 64] [20]. An example stop list of common words in English texts is shown in figure 2.3.

a	an	and	are	as	at	be	by	for	from	has	he	in
is	it	its	of	on	that	the	to	was	were	will	with	

Figure 2.3: Stop list of 25 common words from the Reuters-RCV1 corpus [31, p. 26].

For most queries the fact that these stop words are not indexed has little to no impact, as long as the stop list is created with caution and not too many words are removed. But there are also exceptions, like phrase queries or when searching for song titles or verses. These often consist of very common words and important parts might be stopped out. Using the example stop list from figure 2.3 on the famous soliloquy quote from William Shakespeare's play The Tragical History of Hamlet, Prince of Denmark "To be or not to be", would result in a search on only the two terms 'or' and 'not'.

If you look at the history of stop list usage in IR systems, they generally used to be rather large in the range of 200 to 300 terms. Nowadays it is more common to use small lists consisting of about 10 stop words or no stop list at all, as there are rarely any problems with processing performance or storage space in modern systems [31, p. 27].

Stemming and Lemmatization

The goal of both stemming and lemmatization is to reduce grammatically altered forms of a term into a common base form. *Stemming* is a rather crude but also fast approach, as it simply chops off the ends and derivational suffixes of the terms in an attempt to achieve this. The most widely used one for English texts is the Porter Stemmer, which applies various rules in five sequential phases to shorten the term [35]. *Lemmatization* on the other hand tries to fully morphological analyze the term and return the *lemma* of it, which is the form usually found in a dictionary. This method is more elaborate, but in most cases does not improve the results for IR purposes in a meaningful way compared to the much simpler stemming. Generally speaking, both methods do not necessary help with all queries, rather they improve the performance of some and hurt that of others.

During retrieval, recall gets boosted at the expense of precision [pp. 65-67] [20] For further information about these two values see chapter 2.5.

Equivalence Classing

After stemming is done, several tokens are merged into one as they are equivalent. Nonetheless, there are also plenty of cases when two tokens are not quite the same, but it is still desired that a match occurs. An example is a search query for the term 'USA', where the user also might want to have documents returned that include 'U.S.A.', which just contains some superficial character differences. The standard way to normalize this is by creating *equivalence classes*, which are usually named after one member of the set of tokens they consist of. This way searches for one of the terms will retrieve documents that include either of them. This can be easily done by creating mapping rules that remove certain characters. Then again, it is not obvious when characters could be added, as these mapping rules used are implicit.

Instead of creating equivalence classes, one can also use *synonym relations* between tokens which are stored in a list and can be manually expanded. These relations can be employed during index creation, when a document for example contains the term 'lift', it also gets indexed under 'elevator'. Alternatively they can be indexed the way they are and the synonym list gets considered for query terms. This is the standard way and further explained in section 2.4.2 which handles query expansion [31, p. 28].

2.2.4 Indexing

Indexing is usually done by whatever indexer the IR system has available. As we have seen in the previous section, there are several issues with phrasal or multiword queries that are needed for many technical concepts, organization or street names. To counter that problem, there are different indexing approaches. A *biword index* sees pairs of consecutive terms as a phrase, while a *phrase index* does the same for several terms. However, the most often used method is the use of a *positional index*, storing the position of each term within the document. Occasionally, both approaches are combined [31, pp. 39-43].

Once the indexing is done, the IR system is ready to accept queries and respond to user information needs.

2.3 Retrieval Models

The performance of the information retrieval relies heavily on the retrieval model used. This section gives an introduction to some of the major models and explains the most important term weighting schemes they employ.

2.3.1 Boolean Retrieval

The first developed and most widely used retrieval model is Boolean Retrieval. Until the early 1990s, it was the only model implemented by huge commercial information providers, despite several decades of academic research that proclaimed the advantages of ranked retrieval systems. The *Boolean model* only knows binary weights. That means an index term is either present in a document and therefore has a binary value of 1, or absent and has a value of 0. It does not pay attention to any other factors, no grading scale is possible. The queries for this model follow simple semantics and consist of index terms connected with 'AND', 'OR' and 'NOT' as operators.

One of the major drawbacks of this relatively simple approach is the fact that it uses exact matching, so document retrieval is based on binary decisions. As this allows no ranking, the amount of retrieved documents is directly dependent on the query and can lead to very few or also too many results, which can exceed the number a user can look at more comprehensive. The three operators mentioned above are also too limited for some information needs and make it hard to express them properly, so *extended Boolean models* were developed that also implemented operators for the proximity of terms [6, pp. 25-27], [31, pp. 14-15, 109].

2.3.2 Vector Space Model

This model recognizes the limiting factors of the Boolean Retrieval and offers a framework that also allows partial matching and the computation of a grade of relevance towards the query by finding the documents which have the highest similarity to the query. Unlike boolean retrieval, it can create a list of ranked results.

The basic idea of the *vector space model* is to represent each document as a vector in a multidimensional vector space, where each dimension is corresponding to one term from the dictionary. Coefficients are used to represent the presence or importance of the

terms. This can be done by using binary values, yet most of the time some scaled non-binary term weights are employed. Queries can be represented in the same vector space and are normally entered as a list of terms without using a query language to connect them like Boolean retrieval models do. This concept is illustrated in figure 3.1, which shows a two-dimensional vector space spanned by two terms. In it the document 1 has the highest similarity to the query.

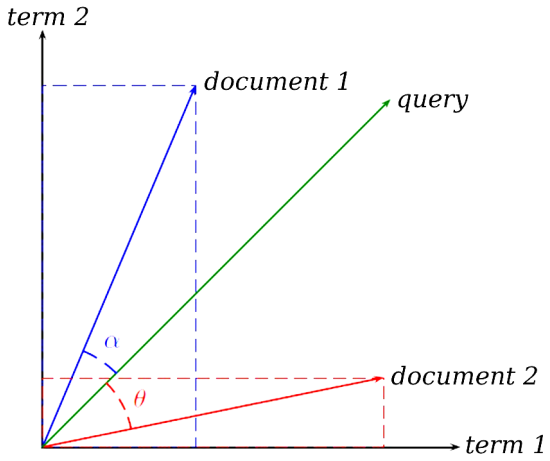


Figure 2.4: Vector space model adapted from [45].

There are two possible methods to quantify the similarity of two documents in a vector space. One of them is to measure the distance between the endpoints of the document vectors. This has the drawback that documents with very similar content might still have a huge distance between each other, simply because one of them is considerably longer than the other. The second possibility to measure the similarity is the most commonly used one, called *cosine similarity*, which is represented by the cosine of the angle between the vectors $\vec{q}$ and $\vec{d}$ in the following equation:

$$similarity(q, d) = \frac{\vec{q} \cdot \vec{d}}{|\vec{q}||\vec{d}|} \tag{2.1}$$

In equation 2.1 the numerator represents the inner product of the two vectors. The denominator is the product of their Euclidean lengths, which normalizes their lengths and thus counters the drawback the first method had [57], [6, pp. 27-30], [31, pp. 120-121].

TF

The most simple way to assign some kind of scaled weight to each term in a document is by setting the weight of the terms equal to the amount of times they occur in the document. This is referred to as *term frequency* (TF) and denoted as $tf_{t,d}$ with t being the term and d the document. In this weighting scheme the ordering of the terms is not important, which is referred to as a *bag of words*. All terms are considered equally important towards the relevancy of a query, although there are often some which have little to no discriminating power at all. Thus this equal importance is considered a problem.

TF-IDF

Instead of the raw term frequency, most of the time some variation of the *term frequency - inverse document frequency* (tf-idf) weight is used. As some terms are generally really common this should be also taken into consideration, which is done by the inverse document frequency, diminishing the weight of terms that are frequent within the entire collection and on the other hand increasing that of terms with a rare occurrence. The tf-idf weight for a term t in a document d is calculated as depicted in equation 2.2. With N being the total number of documents in the collection and df_t the total amount of documents containing t.

$$tfidf_{t,d} = tf_{t,d} \times log\frac{N}{df_t} \tag{2.2}$$

Standard tf-idf is biased towards large absolute term frequencies, as it does not normalize them relative to the document length. *Normalized tf-idf* addresses this issue by utilizing the document length $|d|$ [31, pp. 117-119].

2.3.3 Probabilistic Models

When using the boolean or vector space model for IR, the system only has a semantically imprecise calculus of index terms. With just one query, it can only produce an uncertain guess whether a document has content that is relevant to the information need, as the IR systems understanding for the information need is unsure. *Probability theory* can be used for reasoning under uncertainty and can be used to estimate the likelihood of a document being relevant. There are several retrieval models which have a prob-

abilistic basis, representatively the Binary Independence Model and the Okapi BM25 Model are presented here [31, p. 219].

Binary Independence

As one of the first probabilistic models, the *Binary Independence* uses binary term weights, which in this case is equivalent to Boolean. Both the queries and the documents are represented as binary vectors of the form $\vec{x} = (x_1, ..., x_n)$. If a term t is present in the document, then x_t is set to 1, which reduces many potential documents to the same representation. It assumes that terms occur independently within the documents, meaning that no association between them are recognized, which is actually not correct. Nonetheless, it still performs quite well on most collections, but was not adapted for modern full-text retrieval [31, pp. 222-223].

Okapi BM25

The *Okapi BM25* was the first non-binary model and employs as the name already hints BM25 as weighting scheme. It was developed under the premise that a probabilistic model for full-text retrieval should pay attention to the term frequency and document length, while not introducing too many additional parameters [25]. It was first made public in the 1990s and gained popularity fast, as it demonstrated good performance in full-text retrieval, especially in many of the TREC test collections. Section 2.7.1 provides further information about TREC. Nowadays, Okapi BM25 is one of the most widely used models and there are many different variations of the formula. A commonly used version is displayed in 2.3.

$$BM25(q,d) = \sum_{t \in q} log \left[\frac{N}{df_t} \right] \frac{(k+1)tf_{td}}{k((1-b) + b \times (\frac{L_d}{L_{ave}})) + tf_{td}} \tag{2.3}$$

Here L_d is the length of the document d and L_{ave} the average document length within the collection. The two variables b with $0 \geq b \geq 1$ and k with $0 \leq k$ are tuning parameters. The document length can be scaled by adjusting the value of b, with $b = 0$ corresponding to no length normalization and $b = 1$ fully scaling the term weight by the document length. k calibrates the document term frequency scaling, with $k = 0$ representing no scaling and thus corresponds to a binary model [31, pp. 232-234].

2.3.4 Language Models

The usual approach for a user to formulate a good query, is to try express his information need in terms that are likely to appear in the relevant documents. This basic idea is directly implemented by *language models*. Instead of modeling the likelihood of a document to be of relevance to the query, the basic language modeling approach builds for each document d a probabilistic language model M_d. Based on the probability $P(query|M_d)$ to generate that query, the documents get ranked. To be able to do this, it is necessary to assign probability values to the terms of the language. Different models exist to achieve this by constructing probabilities over sequences of terms, like for example the unigram, bigram and multinomial unigram ones. A language model over an alphabet A can be denoted as the following function, putting a probability measure over the strings s, whereas the accumulated probability is always equal 1 [31, pp. 237-238]:

$$\sum_{s \in A} P(s) = 1 \tag{2.4}$$

Unigram

The *unigram language model* shown in equation 2.5 simply ignores all conditioning context and the ordering of the terms within the document, making it a bag of words model. The fact that it estimates each term t independently, makes it the simplest form of a language model [31, pp. 117, 240].

$$P_{uni}(t_1 t_2 t_3 t_4) = P(t_1)P(t_2)P(t_3)P(t_4) \tag{2.5}$$

Bigram

An example for a more complex model which also takes the preceding context into evaluation, is the *bigram language model* depicted in equation 2.6. Here there probability for each term is conditioned on the previous one [31, p. 240].

$$P_{bi}(t_1 t_2 t_3 t_4) = P(t_1)P(t_2|t_1)P(t_3|t_2)P(t_4|t_3) \tag{2.6}$$

Multinomial Unigram

The unigram and bigram models do not take the multinomial probability of a bag of words into account, as they do not sum over all the possible orderings of the tokens.

This can be done by using the multinomial coefficient C_q shown in 2.7, resulting in the standard *multinomial unigram language model* for the query q in equation 2.8, with M_d being the model for a specific document d.

$$C_q = \frac{L_d!}{tf_{t_1,d}!tf_{t_2,d}!...tf_{t_M,d}!} \tag{2.7}$$

$$P(q|M_d) = C_q \prod_{V \in q} P(t|M_d)^{tf_{t,d}} \tag{2.8}$$

For a particular bag of words the coefficient C_b will be simply a constant. Also, if some of the terms from the query do not appear in the document at all, it might result in a similarity score equal zero. To compensate this, these language models usually employ some form of *term smoothing*. Not only does the zero problem get prevented this way, but at the same time also major parts of the term weighting component are implemented [13], [31, pp. 241-244].

2.4 Query Improvements

> *"The last time that I stood here was seven years ago. [...] You accused me of being the representative of a barbarous species."*
> *"I believe my exact words were: a dangerous, savage, child-race."*
> Captain Picard and Q, Star Trek: The Next Generation, "All Good Things..."

As already briefly mentioned in the equivalence classing subsection 2.2.3, *synonyms* often pose a problem for the IR system when trying to retrieve relevant documents. The degree of overlap between the query terms and terms in the relevant documents influences the effectiveness of the IR system.

Users often try to address this problem by manually resubmitting a query with different synonyms and refining it that way, until they get a desired result list. However, the ability of the user to successfully formulate alternative queries depends on his general experience of searching and his expert knowledge in the domain of the collection he is searching in. There are ways for the IR system to take on that tedious process or at least to assist the user with it [21], [31, p. 177].

2.4.1 Relevance Feedback

The idea behind *relevance feedback* is that it is often hard for the user to formulate a good query if he does not know the collection well. So he is given the option to interact with the retrieval process, with the purpose to improve the final result list. The basic procedure consists of the following steps:

- The user submits a query, which should be simple and short, to the system.
- The system returns an initial result list.
- The user gives feedback by flagging some of them as relevant or non-relevant.
- Based on that feedback, the system computes an improved representation of the information need.
- The systems returns a revised result list.

The relevance feedback can go through multiple iterations, repeating the last three steps for further refinement. Relevance feedback is especially potent when performing image search, where it can be hard to express the need in a few terms, but easy to quickly judge the results. Nonetheless, it is often not popular with users as they do not want to invest further time into refining the query or are reluctant to provide additional feedback. It is sometimes also difficult to comprehend why a certain feedback choice influenced the next result set the way it did.

To cope with the issue of the user potentially refusing the explicit interaction, some modified versions of relevance feedback have been developed. *Blind relevance feedback* automatically presumes that the top ranked documents in the result list are relevant and uses this assumption to generate feedback data, so it can feed the feedback-loop itself. *Implicit relevance feedback* analyses the user actions, like the clicking on a returned document in order to read it, and rearranges the ranking based on that indirect feedback [6, pp. 117-118], [31, pp. 178, 185-188].

2.4.2 Query Expansion

With *query expansion* users have the option to give additional input on the query terms themselves instead on the result. Some IR systems, especially those used on the web,

suggest alternate but related queries to the one submitted by the user. The most common way to generate these alternate suggestions is global analysis by the use of a *thesaurus*, which contains synonyms or semantically related subjects. It is also possible to automatically expand the query by adding for each query term its synonyms from the thesaurus to the query. This generally increases the amount of relevant documents retrieved while not requiring a user input.

There are several ways to create such a thesaurus, either manual by human editors or automatically. For example the editors can maintain a list with canonical terms for each concept, much like in traditional libraries where you have subject indices and under each of these a vocabulary of possible synonyms or related terms. This is an especially common procedure for well resourced domains. A thesaurus can also be automatically derived by the use of statistics about term co-occurrences in a collection of documents from a specific domain. Another automatic option to suggest query alternatives to a new user is by exploiting the manual query reformulation attempts of users who already used the IR system before him. To perform this query log mining, a huge amount of generated queries and thus users is needed, which makes it an eligible method for web search systems [31, pp. 189-192].

2.5 Evaluation of Information Retrieval Systems

The objective evaluation of search performance is an important cornerstone of IR research.

2.5.1 Reasons for Evaluation

Given the experimental nature of IR research, progress critically depends upon experimenting with new ideas. However, experience has shown that new ideas and potential improvements for the search ability of a system that look good in theory often have very little or no impact at all when observed under excessive testing. To make noticeable progress, it is necessary to keep evaluating the performance and experimenting with alternatives, so useful changes can be distinguished from superfluous ones [48].

Two different ways to evaluate the performance of an IR system are effectiveness and efficiency. In a general sense, *effectiveness* measures the ability to find the right

information. More specific, if a definition of relevance is available, it is possible to determine the effectiveness by comparing the ranking an IR system produced in response to a query to the ranking created by user relevance judgments. *Efficiency*, on the other hand, measures how much time or space the system needs to create that ranking.

Most of the time, IR research first focuses on improving the effectiveness and once a new technique is found to do that, resources are used to create an efficient implementation. It is usually done this way, because the main intention is to actually find relevant information. A retrieval system that is only fast without producing good results would be useless in most use cases [58, p. 112], [20, pp. 269-271].

2.5.2 Basic Requirements

To measure the performance of an IR system in the standard way, the following three components are required:

- A document collection.
- A set of queries, expressing the information need.
- A set of relevance judgments, labeling the relevance of each query-document pair in a binary way.

2.5.3 Evaluation Measures

The most common measures to evaluate if a retrieval system is effective and satisfies the information need are Precision and Recall. These two are usually inversely related, so it is not possible to get the best value for each of them. A typical relation is shown in figure 2.5. While the slope of the curve may vary for different collections and retrieval systems used, the general relationship remains. The optimum value would be in the upper right corner of the graph, maximizing both Precision and Recall.

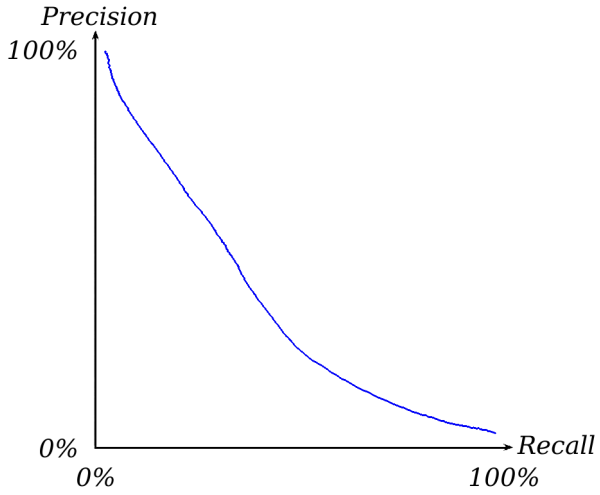


Figure 2.5: Precision and Recall inverse relation.

In order to express the Precision and Recall combination as a single value, F-measure can be used. There three values are in principle measures between 0 and 1, but it is common practice to express them as percentages. A single measure for Precision and Recall in ranked retrieval sets is the Mean Average Precision. Something all these measures have in common is the fact that they require relevance judgments. If they are not complete, Binary Preference can be used. Another measure, which does not need relevance judgments at all, is retrievability. This fact will play a vital role for the experiments conducted in the practical part of this thesis.

In *unranked retrieval* the IR system returns a set of documents for a given query. There exist four possible combinations of retrieval state and relevance of a document. This is depicted in table 2.1.

Table 2.1: Relevance and retrieved combinations [31, p. 155].

	relevant	not relevant
retrieved	true positives (tp)	false positives (fp)
not retrieved	false negatives (fn)	true negatives (tn)

Accuracy

Accuracy measures the fraction of classifications that an IR system did correct and is defined as shown in equation 2.9.

$$Accuracy = \frac{tp + tn}{tp + fp + fn + tn} \tag{2.9}$$

While this is an often used method to evaluate machine learning classification problems, it usually is not a useful one for IR systems. This is due to the fact that in information retrieval the data is in most cases massively skewed and less than 0.1 percent of the documents are actually relevant to a given query. The Accuracy of an IR system can simply be maxed by labeling all documents as non-relevant. However, this wouldn't be in the interest of the user at all, as he expects to get some relevant documents returned (tp). As long as his information need gets satisfied he usually does not mind if some of the returned documents he has to look through are not relevant (fp) [31, pp. 155-156].

Precision

Precision measures the share of documents returned by the system that are also relevant for the query relative to the total amount of documents retrieved. Knowledge about relevance and documents retrieved is needed to calculate it, as shown in equation 2.10. However, if for example the collection contains a lot of relevant documents and all the IR system returns is a single relevant document, the Precision is still 100 percent, without this being a satisfying result set in most cases. That is why Precision is often couped with Recall [31, pp. 154-155].

$$Precision = \frac{tp}{tp + fp} \tag{2.10}$$

Recall

Normally it is not worthwhile to just maximize Recall either, as for example 100 percent Recall can be achieved if the system simply returns all documents from the collection as result set. This automatically includes all relevant documents, but basically completely ignores the query.

Recall measures the ability of a retrieval system to successfully retrieve these documents from a collection that are relevant to the query. To measure this, knowledge

about relevance, documents retrieved and documents not retrieved is needed. Recall has a value between 0 and 1 and is defined as shown in 2.11.

$$Recall = \frac{tp}{tp + fn} \tag{2.11}$$

An ideal IR system should have both a high Recall and Precision value. Meaning it should retrieve as many relevant documents as possible while only retrieving a small amount of non-relevant ones. While one approach can be to simply maximize the combination of both, it not always represents the ideal combination for the needs of the user or the purpose of the system, where another balance might be considered more favorable to the desired result.

To make comparisons between different IR systems easier, a single value that can be used is the *breakeven point*. This is the point at which Precision and Recall are equal, which usually can be achieved by tuning the IR systems parameters [31, pp. 154-156, 161].

F-Measure

A single measure used to tune Precision versus Recall is called *F-measure*, which is the weighted harmonic mean of these two and calculated as depicted in equation 2.12. With $\beta < 1$ Precision (P) is emphasized and with $\beta > 1$ Recall (R). The default value for β is 1, equally weighting both. This is referred to as the *balanced F-measure*, also called F_1 which is an abbrevation for $F_{\beta=1}$ [31, p. 156].

$$F = \frac{(\beta^2 + 1)PR}{\beta^2 P + R} \tag{2.12}$$

Mean Average Precision

Nowadays it is standard for search engines to use *ranked retrieval*, which means the system usually returns the top k documents considered relevant for the query as a result list and the user is usually more interested in the ones at the beginning. This can be displayed in *Precision-Recall Curves*. The most standard measure to express that information as a single measure of quality is *Mean Average Precision* (MAP). For a single information need expressed as a query, the *Average Precision* is calculated by averaging the Precision values after each relevant document that is found. This results in a high

Average Precision value if many relevant documents are among the top k retrieved ones. To get the MAP, this is done for many queries and the thus obtained Average Precision values get averaged themselves. The equation is shown in 2.13, with $\{d_1, ..., d_{m_j}\}$ being the set of relevant documents for an information need $q_j \in Q$. R_{jk} is a set of ranked results, from the top one until document d_k is reached [31, pp. 159-160].

$$MAP(Q) = \frac{1}{|Q|} \sum_{j=1}^{|Q|} \frac{1}{m_j} \sum_{k=1}^{m_j} P(R_{jk}) \tag{2.13}$$

As the MAP values for different information needs within the same system often vary extremely, they are more often used to compare a system to others by the use of a single information need [31, p. 161].

Binary Preference

For incomplete relevance judgments, the *Binary Preference* (bpref) measure can be used. It computes a preference relation of judged documents: whether relevant ones are retrieved ahead of irrelevant ones. Bpref is calculated as shown in 2.14, with $r \in R$ being a relevant document and n being a member of the first R judged non-relevant documents retrieved by the IR system.

$$bpref = \frac{1}{R} \sum_{r} (1 - \frac{|n \text{ ranked higher than } r|}{min(R, N)}) \tag{2.14}$$

Bpref works well in most practical cases, unless the amount of relevant documents is extremely small [17].

Retrievability

To compute the *retrievability*, which is often also called *findability*, no relevance judgments are needed at all. For this a qualitative measure was introduced for the first time by Azzopardi and Vinayin in [5] and adopted from the over 50 year old concept of *Accessibility*, used in the field of land use and transportation planning. There it was defined as a measure of interaction with certain resources in a physical space like dining, education, employment, shopping etc. via the use of certain transportation systems, for example, streets for cars, public transportation like bus and railroad or by the use

of bicycles [24]. In the context of IR the analogy of physical space is the information space, usually represented by the document collection. Instead of various transportation systems, the user has an IR system that allows him to access that information by issuing queries. Unlike in physical space the current location of the user, meaning which document he retrieved last, does not matter. Everything is potentially accessible from everywhere at any time, as the user can issue any query he likes. However, the choice of the route and the distance the user is willing to travel have a direct impact on how accessible a document is within that information space [4].

Based on that analogy, retrievability indicates how easy the information within a document collection can be accessed with a given retrieval model, or if it can be accessed at all. A high retrievability value means the probability for a user to find it by querying is high, too. Intuition tells us that the retrievability is high if:

- There are many different queries $q \in Q$ which can be expressed to retrieve a document d or the probability o_q to formulate a relevant query is very high.
- The user is willing to examine a certain amount of documents in the result list, which is called rank cutoff c. The rank k_{dq} of the document in question for a certain query is as low as possible, but at least as low as the rank cutoff.

The willingness of the user to go down the list of ranked results is a very important factor and varies wildly from application to application. When using a web search engine the user usually only looks at the first page and is nearly never willing to go past page three, so c is rather low in this case. Based on this reasoning, the following measure 2.15 for the retrievability of a document $r(d)$ can be formulated.

$$r(d) = \sum_{q \in Q} o_q \cdot f(k_{dq}, c) \tag{2.15}$$

The function $f(k_{dq}, c)$ is a generalized utility/cost function and returns 1 if $k_{dq} \leq c$, in all other cases 0. As the likeliness that a user issues a query o_q is really hard to determine, unless the queries are based on a set of historical query samples, it is in most cases straightforward set to 1, giving all queries equal probability values. By this definition, the retrievability of a document is simply the accumulated amount of times it can be retrieved below the cutoff level over the set of all queries Q. In most cases it is impractical to calculate the absolute $r(d)$ values because of the large size of Q, so

some estimation of the retrievability has to be used. An approach to that problem is to use subsets of the possible queries, containing a sufficiently large amount of probable queries. This can for example be done by the use of a historical set of queries from a query log or by using an algorithm for query based sampling [18].

A visual way to analyze the inequality of retrievability in document collections is by the use of *Lorenz Curves*, which are especially popular in economics where they are for example used to visualize population income distributions, as shown in 2.6.

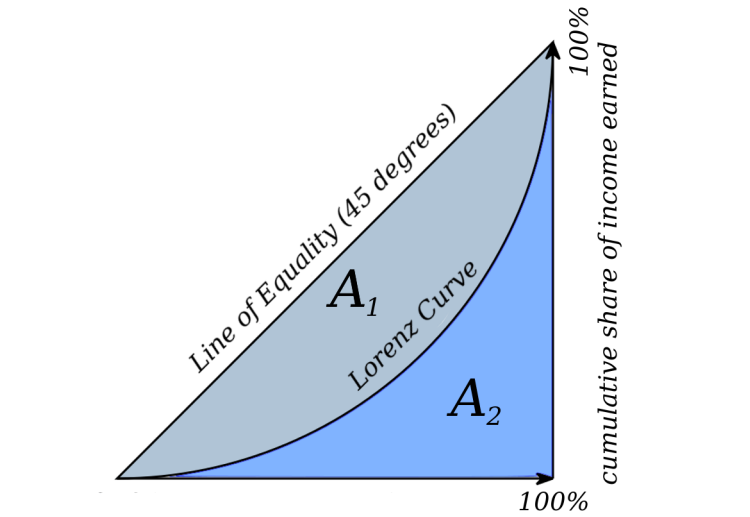


Figure 2.6: Example Lorenz Curve adapted from [44].

This is done by first sorting the population in ascending order of their wealth and then creating a plot of this cumulative distribution. In an equal distribution the line would be like the Line of Equality. The extent to which the distribution deviates from the Line of Equality is reflected by its skewness. This concept can be applied for retrievability analysis in the same way, by simply using r(d) instead of wealth and thus visualizing the bias of an IR system.

The *Gini-Coefficient* G can be used to summarize the Lorenz Curve into a single value. Looking at figure 2.6 from a bird's eye view, it is given by the ratio of the areas bounded by the curve as shown in equation 2.16. It can be computed as shown in

equation 2.17, with D being the amount of documents in the collection [22].

$$G = \frac{A_1}{A_1 + A_2} \tag{2.16}$$

$$G = \frac{\sum_{i=1}^{|D|}(2 \cdot i - |D| - 1) \cdot r(d_i)}{(|D| - 1)\sum_{j=1}^{|D|} r(d_j)} \tag{2.17}$$

Consequently G can have a value between 0 and 1. If it equals 0, then there is no bias at all and all documents are equally retrievable. The higher G, the more bias is present in the retrieval system towards the document collection used [5].

2.6 Search Engines and IR Toolkits

Several different search engines exist that allow comfortable indexing, retrieval and optional ranking of documents. This section will give a brief overview of some established open source search engines and IR toolkits which still received updates in the last four years, are not stagnating and therefore do not have to be considered outdated. A more elaborate description of the *Lemur Toolkit* is given, as this is the one employed for the experiments conducted in this thesis.

2.6.1 Comparison

Table 2.2 gives a basic comparison of the toolkits by means of nine important characteristics. Filetype describes the type of file the toolkit is able to parse. Storage describes the way the index is stored, which can either be in a database or a more simple file system, like an inverted index. The possibility to add new files to that index without the need of recreating it, is indicated by the increment column. Stop words indicates if the indexer is capable of using a list of words that should be discarded, and stemming if there is a possibility to apply stemming operations over words. Fuzzy search is the ability to solve queries without exact matching. Ranking describes if the results of a search are based on a ranking function. Three possible search types are Boolean (b), phrase (p) and wild card (w). Finally, language lists the programming language that was used to implement the toolkit.

Table 2.2: Comparison of search engine characteristics adapted from [32].

Search Engine	Filetype	Storage	Increment	Stop Words	Stemming	Fuzzy Search	Ranking	Search Type	Language
DataparkSearch	html txt xml	database	✓	✓	✓	✓	✓	b	C
IXE Toolkit	html txt xml	file	✓	✓		✓	✓	p b w	C++
Lemur Toolkit	html txt xml pdf	file	✓	✓	✓	✓	✓	p b w	C++
Lucene	html txt pdf	file	✓	✓	✓	✓	✓	p b w	Java
MG4J	html txt	file	✓	✓	✓		✓	p b w	Java
mnoGoSearch	html txt	database	✓	✓	✓	✓	✓	b	C
Namazu	html txt	file	✓				✓	p b w	C
Omega	html txt pdf ps	file	✓	✓	✓		✓	p b w	C++
Omnifind	html txt xml pdf ps	file	✓	✓	✓	✓	✓	p b w	Java
OpenFTS	html txt	database	✓	✓	✓	✓	✓	b p	Perl
SWISH-E	html txt xml	file	✓	✓	✓	✓	✓	p b w	C
SWISH++	html txt	file	✓	✓	✓		✓	p b w	C++
Terrier	html txt xml pdf ps	file		✓	✓	✓	✓	p b w	Java
XMLSearch	xml	file	✓	✓		✓		p b w	C++
Zettair	html txt	file	✓	✓	✓		✓	p b w	C

Supplementing the comparision table, a short additional description of the main characteristics is given in the following subsections.

DataparkSearch

The *DataparkSearch Engine* is a search engine with a web CGI front-end designed for searching websites, the intranet or a local system [63].

IXE Toolkit

The *Ideare indeXing Engine* (IXE) is a set of modular object-oriented frameworks consisting of C++ classes and utilities, providing indexing, querying and analyzing fuctionalities for collections of documents. While there exists a commercial version from Tiscali, there is also a non-commercial version available for academic use [32].

Lucene

As full-featured text search engine library, part of the Apache Software Foundation, *Lucene* is often used in various applications that make use of it [1].

MG4J

Managing Gigabytes for Java (MG4J) was developed at the the University of Milano initially as a text indexer consisting of a loosely coupled set of classes. It has evolved into a complex system implementing a large class of scalable algorithms that are of interest to the text-retrieval community [16].

mnoGoSearch

While the Windows version of *mnoGoSearch* has a graphical user interface and is sold commercially, the unix command line version is open source. Besides the usual indexing and retrieval of website content, this toolkit is also capable of indexing multilingual websites and fetching versions of the same page in different languages via content negotiation technology [29].

Namazu

Namazu, which is the japanese word for catfish, does not only allow easy search in document collections, but also provides a personal search system for emails or other files [41].

Omega

Omega is a retrieval application based on the *Xapian Project*, which is a probabalistic IR library and can be binded to different programming languages like Perl, Python, PHP, Java, Tcl, C#, Ruby and Lua [43].

Omnifind Yahoo! Edition (IBM)

The *Omnifind Yahoo! Edition (IBM)* combines an internal search based on the Lucene search engine with the possibility to search the Internet using the Yahoo! search engine [32].

OpenFTS

The *Open Source Full Text Search engine* (OpenFTS) provides online indexing of data and relevance ranking for database searching. The close database integration allows the use of metadata to restrict search results [7].

SWISH-E and SWISH++

Simple Web Indexing System for Humans-Enhanced (SWISH-E) is an improved version of the original SWISH which was written by Kevin Huges in 1994. SWISH-E was completely rewritten in C++ and released as SWISH++ containing most of the features, but not all of them [32].

Terrier

TERabyte RetrIEveR (Terrier) was developed at the University of Glasgow as a modular platform allowing the creation of large scale search applications. It is also often used for the evaluation of TREC collections [38].

XMLSearch

XMLSearch is a set of classes developed in C++, for indexing document collections and searching with text operators like equality, prefix, suffix, phrase, etc. There is a commercial version available from Barcino and a non-commercial version available for academic use [32].

Zettair

Zettair was originally known as *Lucy* and developed at the Royal Melbourne Institute of Technology University [32].

2.6.2 Lemur Toolkit

The *Lemur Toolkit* is an open source framework for language modeling and information retrieval, offering sophisticated structured query languages. It was developed as a

cooperative work between the Center for Intelligent Information Retrieval at the University of Massachusetts and the Language Technologies Institute at the Carnegie Mellon University as part of the *Lemur Project*. It offers Porter and Krovetz word stemming, acronym and stop word recognition for English, Chinese and Arabic text. The Lemur Toolkit includes the *Indri Search Engine*, which was originally developed as a standalone project, but later integrated into the toolkit. Indri offers both command line tools and a Java GUI and was designed to address the following goals:

- The query language supports complex queries which involve evidence combination and has the ability to specify a wide variety of constraints.
- Superior effectiveness across a range of query and document types provided by the retrieval model.
- Both query language and retrieval model allow retrieval at different granularity levels.
- Support very large and multiple databases, fast and concurrent indexing, optimized query execution, and portability.

As these four aims are partially in conflict with one another, decisions were made to support one at the expense of another. Nonetheless, the creators of Indri believe that they managed to achieve a functional balance between its goals while keeping a clear and easily modifiable code that is usable in an academic setting [53].

For index creation, compressed inverted lists for each term and field in memory are build and periodically written to disk. The data created this way is self-contained, meaning all necessary information to process a query is saved in it. It is also stored in an accessible manner to support the development of new retrieval strategies. Several retrieval models are supported, including tfidf, Okapi BM25, a method based on the KL-divergence language model, InQuery, cosine similarity model or the Indri structured query language. Different parameter settings can be used for these models.

The Indri system architecture offers concurrency. Queries can be run and evaluated against several indexes simultaneously, which do not necessarily have to be on the same machine. In addition to queries, also document deletions and insertions can be processed simultaneously, allowing retrieval operations on dynamic data collections like

constantly updating news feeds. It can handle terabyte-sized document collections and can be used on a cluster of machines to speed up the indexing and retrieval process [39].

2.7 Standard Test Collections

The principal tool the evaluate IR systems are *test collections*. They usually consist of documents, a set of queries and *relevance judgments*, which are a vital part of a test collection. It does not take much effort to gather the documents or create queries for them, but the manual creation of relevance judgments requires a lot of resources and is usually very costly, too. Therefore it is often an economic necessity to re-use the same collections for which these judgments already exist. There are other advantages of using standard collections, like the ability to allow researchers to better understand, reproduce and compare results [47].

This section presents test collections with the main evaluation focus on ad hoc IR systems and further explains the creation of relevance judgments.

2.7.1 Popular Collection

20 Newsgroups

A single but widely used collection which is called *20 Newsgroups* consists of 18941 accumulated documents, which were nearly evenly distributed across 20 different newsgroups, each corresponding to a different subject [28].

CLEF

The *Conference and Labs of the Evaluation Forum* (CLEF), which was formerly known as Cross-Language Evaluation Forum, focuses mainly on documents containing European language text and cross-language IR. For this purpose it provides plenty of collections, including an annually ad hoc retrieval one, between the years 2000 and 2009 [19].

Cranfield

The *Cranfield collection* was the first one to allow precise quantitative IR evaluation and was created in the late 1950s. It has an extremely small size of only 1398 abstracts

from aerodynamic journal articles, 225 queries and the corresponding list of relevance judgments. Because of that, nowadays it is not really used for anything any more except very elemental experiments [31, p. 153].

NTCIR

The *NII Test Collection for IR Systems* (NTCIR) project focuses on documents containing East Asian language text and cross-language IR. The size and amount of the collections is comparable to the TREC ones [37].

Reuters

Reuters is the largest international television and text news agency. The Reuters-21578 corpus is one of the most widely used collections for text classification and information retrieval and was used for hundreds of published studies. It originally contained 22173 newswire articles from 1987 and was made available for research purposes for the first time in 1990. During a cleanup and relabeling process in 1996, some of the documents that were exact duplicates of each other were removed, reducing the original corpus to 21578 documents, which gave the collection its name. Meanwhile, there also exists a follow up corpus called the Reuters Corpus Volume 1 (RCV1). The purpose of this was to eliminate some weaknesses of its predecessor, like the limited overall size of the collection or the fact its articles did not cover a whole year, arguably causing a somewhat biased content [46].

TREC

The *Text REtrieval Conference* (TREC) is a yearly recurring conference for the evaluation of IR systems, which was hosted for the first time in 1992. It originally developed out of TIPSTER, a project to encourage the advancement of text handling technologies. TREC provides many large English language collections, consisting of millions of documents. They are used very frequently by both scientific and commercial members as an equal testbed to evaluate and compare their results. For the practical part of this thesis, several TREC collections served as a testbed. An example document is depicted in listing 2.1.

Listing 2.1: Example Federal Broadcast Information Service document.

```
<DOC>
<DOCNO> FBIS3-3070 </DOCNO>
<HT> "drchi051_s_94001" </HT>
<HEADER>
<AU> FBIS-CHI-94-051 </AU>
Document Type:Daily Report
<DATE1>  10 Mar 1994 </DATE1>
</HEADER>
<F P=100> Northeast Region </F>
<H3> <TI> Heilongjiang Facilitates Border Trade With CIS,
    Japan </TI></H3>
<F P=102> OW1003134394 Beijing XINHUA in English 1313 GMT 10
    Mar 94 </F>
<F P=103> OW1003134394 </F>
<F P=104> Beijing XINHUA </F>
<TEXT>
Language: <F P=105> English </F>
Article Type:BFN
[Text] Harbin, March 10 (XINHUA) -- Among 21 state-approved
    border ports in northeast China's Heilongjiang Province, 17
    have gone into operation. According to one official, more
    and more ports with improved facilities have boosted the
    province's border trade with the Commonwealth of
    Independent States (CWIS) [abbreviation as received]. Up to
    now, the ports have handled over 273,000 tons of cargo
    annually and 1.76 million businessmen entering and leaving
    China. A transportation network has been established in the
    province, including airlines from Harbin, capital of the
    province, to Japan and CWIS, railways to Russia and
    shipping routes to Russia and Japan. In 1993, the
    province's import and export volume rose to over 2.6
    billion Swiss francs.
</TEXT>
</DOC>
```

Because of the enormous size of some of the collections, no exhaustive relevance judgments exist for all of them [31, pp. 153-154], [56].

2.7.2 Relevance Judgments

To create a complete set of *relevance judgments*, for each query every single document has to be judged if it contains relevant material. These relevance judgments, which are also known as ground truth, have to be manually created and usually contain binary relations between the queries and documents. For such a relation to be labeled as relevant, it is not important that the document contains all terms of the query, it simply has to satisfy the information need that stands behind the query, which is often also called the *topic*. A difficulty that arises during the labeling process is the fact that humans usually disagree about the relevance of some documents. However, studies have shown that looking at the big picture this has little influence [51], [59].

An example TREC relevance description for a query is shown in listing 2.2. While the `<title>` field contains a very short query, consisting of a maximum of three words, the `<desc>` field holds a description of the topic area in a single sentence. Finally, `<narr>` gives an exact definition of what makes a document relevant.

Listing 2.2: Example TREC topic relevance description.

```
1 <top>
2 <num> Number: 396
3 <title> sick building syndrome
4 <desc> Description: Identify documents that discuss sick
    building syndrome or building-related illnesses.
5 <narr> Narrative: A relevant document would contain any data
    that refers to the sick building or building-related
    illnesses, including illnesses caused by asbestos, air
    conditioning, pollution controls. Work-related illnesses
    not caused by the building, such as carpal tunnel syndrome,
    are not relevant.
6 </top>
```

To avoid having to manually label every query-document pair with a relevance judgment, which is nearly impossible for very large collections and would take thousands of hours for a single query, the *National Institute of Standards and Technology* (NIST) uses

a method called *pooling* to create a subset of potential relevant documents of the TREC collections. Only the documents in this pool are then judged for relevance regarding a specific topic and documents that are not in that pool are automatically assumed to be not relevant. This technique is considered valid when there are enough relevant documents found that the judgments set resulting from it is unbiased and approximately complete. The following steps are taken for the creation of such a pool [60]:

- NIST creates 50 new topic statements and releases them to the TREC participants.
- The participants are free to use any method they wish to create queries from these topics to search the document set. It is distinguished for comparison reasons if results were acquired with automatic or manual created queries.
- Each participant submits some number of runs, which consist of a maximum of 1000 top retrieved documents. A subset of these runs are labeled official runes.
- From each official run, NIST takes the top 100 documents per topic to form the pool for this topic. Duplicate documents are removed from the pool.

2.8 Information Retrieval Problems

This section introduces some important problems IR has to deal with, which are also relevant for the practical work conducted in this thesis.

2.8.1 Bias

In the IR domain, bias is the representativeness of a set of retrieved documents in response to a set of queries. Undue inclusion or exclusion of certain documents implies such a bias. An IR system is therefore highly biased if the results created by it are noticeable different from the norm created by a group of IR systems. There are plenty of sources that might create such a bias, for example the decision of what documents are included into a collection, in which way they are indexed or what algorithm is employed to retrieve them [33].

2.8.2 Vagueness and Uncertainty

In 1985 an often cited study by Blair and Maron [15] on the effectiveness of IBM's Storage And Information Retrieval System (STAIRS) was published. This IR system was state-of-the-art at that time and the study came to a surprising conclusion. Users were only able to retrieve about 20 percent of the relevant documents, while they thought themselves to be retrieving more than 75 percent. According to that study the main reason for these low recall values was the following:

> *"...full-text retrieval is difficult to use to retrieve documents by subject because its design is based on the assumption that it is a simple matter for users to foresee the exact words and phrases that will be used in the documents they will find useful, and only in those documents [...] (however) it is impossibly difficult for users to predict the exact words, word combinations, and phrases that are used..."*

While the study itself is quite old, especially in the fast changing field of computerized information technology, its conclusion still holds true today. There exist two basic problems when trying to access and retrieve information [14], [26]:

- *Vagueness*, which means that the user does not have a precise idea of what his actual information need is. Therefore the conditions for query generation are vague.
- *Uncertainty* on the other hand describes the lack of knowledge of the system about the content of one or several documents. One reason for this can for example be homographs, words that share the same spelling but have different meanings, like the word 'bow', which can be the front of a ship, a ranged weapon, the action of bending forward at the waist and a lot of other things [36]. The true meaning is usually only conveyed by the context, which is a common cause for incorrect retrieval.

2.8.3 Lack of Retrievability

With evaluation by means of the retrievability value, another major problem gets obvious: some retrieval systems are not able to retrieve some documents at all via means

of reasonable queries. So no matter what query the user issues, he will not be able to access them although they are present in the document collection. This is especially problematic in recall oriented fields like the patent law domain, where it is imperative to find all documents dealing with prior art and a failure to do so can cause very expensive lawsuits [5].

2.9 Related Work to substitute Relevance Judgments

The creation of human relevance judgments for large document collections is a very time and resource consuming process, as already elaborated in section 2.7. Nevertheless, this ground truth is needed for most evaluation processes. For this reason the interest in the IR research community has sparked to propose new methods to evaluate retrieval systems without the need of these judgments. This chapter provides an overview of other work related topic of this thesis, divided into three different approaches. First the replacement of human relevance judgments by automatically created ones is explained, followed by various experiments for retrieval system ranking by means of their similarity. Finally, a method to create the ranking based on the retrieval bias is presented.

2.9.1 Pseudo-Relevance Judgments

Based on previous observations that different human relevance judgments do not affect the relative measured effectiveness of the retrieval systems, Soboroff et al. propose in [49] to replace the human relevance judgments altogether. They use automatically created *pseudo-relevance judgments* instead. This is done by taking the top retrieved documents from a wide spectrum of retrieval systems like it is done for the TREC pooling method explained in subsection 2.7.2. Instead of having humans assign relevance to this selected subset, relevance judgments are created by randomly mapping topics to the documents. They conclude their experiments with the findings that the retrieval rankings created by their method positively and significantly correlates with the actual TREC rankings. They noticeable separate the best and worst systems from the middle ones. Still, the method is not good at predicting the best system.

In [34] Nuray and Can also employ pseudo-relevance judgments for automatic retrieval model evaluation. Much like Soboroff et al., they use a subset of top retrieved

documents for the creation of these judgments. Furthermore, they try several methods to find a good performing subset of retrieval systems for the creation of that document pool, instead of using all of them. The best performing method they found is to use retrieval systems with a large bias that deviate a lot from the average model. In addition to the selection of a retrieval model subset, they also try three different methods for merging the results created by them. These methods are rank position, Borda count and Condorcet election. The latter one performed the best. Their results are easy comparable to those of Soboroff et al., as they work with a similar set of TREC collections. They come to the conclusion that their method generally creates better results on these datasets used.

2.9.2 Retrieval System Similarity

Aslam proposes in [2] a measure to quantify the *similarity* of retrieval systems. This is done by assessing the similarity of their retrieval results as depicted in equation 2.18, where Ret_i indicates the set of documents returned by system i.

$$similarity(System_1, System_2) = \frac{|Ret_1 \cap Ret_2|}{|Ret_1 \cup Ret_2|} \tag{2.18}$$

With the purpose to create an evaluation for the systems, they are ranked according to their average similarity with the other systems. Aslam concludes that this similarity measure does not properly rank the top retrieval system and declares the similarity scores as a measure of the aggregate bias which must be overcome to achieve valid evaluation of a retrieval systems performance.

Wu and Crestani also employ the similarity between retrieval systems in order to rank them. They propose in [62] a measure called *reference count* to do that. This measure is computed by examining the result sets of different IR systems created by a query in the following way. A certain amount of top documents is selected in the result set of one system. The occurrences of these documents in the results of all other systems get accumulated. This creates a reference count score for the system under this particular query. This method can be further refined by assigning weights to the documents, for example weighting those superior that appear more frequent in the top positions of the result sets. They conclude that the reference count method is effective to predict the performance of retrieval systems and provides a good alternative measure

for their ranking. Like most methods without relevance judgments, it performs better at the bottom end of the ranking and the best systems cannot be predicted well.

Spoerri assumes in [52] that the use of all available retrieval models, as it was usually done in other similarity experiments, can create a biased boost for certain retrieval systems. This is due to the fact that some retrieval systems are nearly the same. Based on this assumption he experimented with multiple sets of five randomly selected retrieval systems and the result set similarities among them. To create a system ranking, the results of all random system subsets are averaged. Spoerri concludes that the correlations with the TREC rankings are stronger than the ones in previous works, especially due to the fact that the best systems usually reside in the top half of his performance ranking.

2.9.3 Retrievability and Effectiveness

Azzopardi and Bache conduct a preliminary study in [3] on the relation between effectiveness and retrievability. They compare the retrieval bias represented by the Gini-Coefficient with the effectiveness represented by the precision value at different retrieval model parameter settings. They come to the conclusion, that the goals of both improving retrievability and precision are indeed compatible, as parameters that lead to maximum retrievability still lead to good precision values. Thus, they propose the hypothesis that the performance of retrieval systems can be tuned using access based measures which do not require relevance information.

Bashir and Rauber conduct an experiment in [11] to investigate in which way improving the retrievability effects the accuracy on a retrieval system. For this they partition the documents into two categories which consist of documents with high and low retrievability and can be accessed separately. To avoid extensive retrievability analysis in order to create that split, they do that classification via a set of surface-level features as they describe in [10]. After the partitions are processed separately, their result sets are merged again. They come to the conclusion that this does not only improve the overall retrievability, but also has a positive influence on the accuracy of the result set. They remark that the accuracy improvement is dependent on the query generation process and the retrieval model used.

In [8] Bashir does an experiment with different parameter settings in which a strong relationship between the effectiveness measures and the Gini-Coefficient is indicated.

Nevertheless, on the collection used some parameter settings that lead to a low retrieval bias actually hurt the effectiveness by a small fraction.

2.9.4 Conclusion for the Experiments

The methods employing Pseudo-Relevance Judgments and Retrieval System Similarity usually only perform well in identifying the worst systems, but mostly fail to correctly distinguish the ones in the top performing half. However, for the tuning of retrieval systems especially the top performers are the interesting ones. Therefore, the experiments in this thesis will pursue the idea of a potential relationship between retrievability and effectiveness measures to investigate if this offers a suitable way for top performer identification.

CHAPTER 3

Methods

The main objective of this thesis is to investigate if and to what degree it is possible to optimize different search engine effectiveness measures by tuning them via the retrievability bias and thus avoiding the need for relevance judgments. In order to examine if good parameter values can be deduced this way, comparisons of these measures are created across a broad spectrum of parameter values. This is done for three different retrieval models, which were selected based on results shown in the related work, namely those of Bashir [8].

To allow the investigation of consistency across different collections, all experiments are conducted on four different document collections. These were chosen based on possessing different surface level features.

The basic steps taken and their dependencies on one another are illustrated in figure 3.1 and explained in detail in the rest of this chapter.

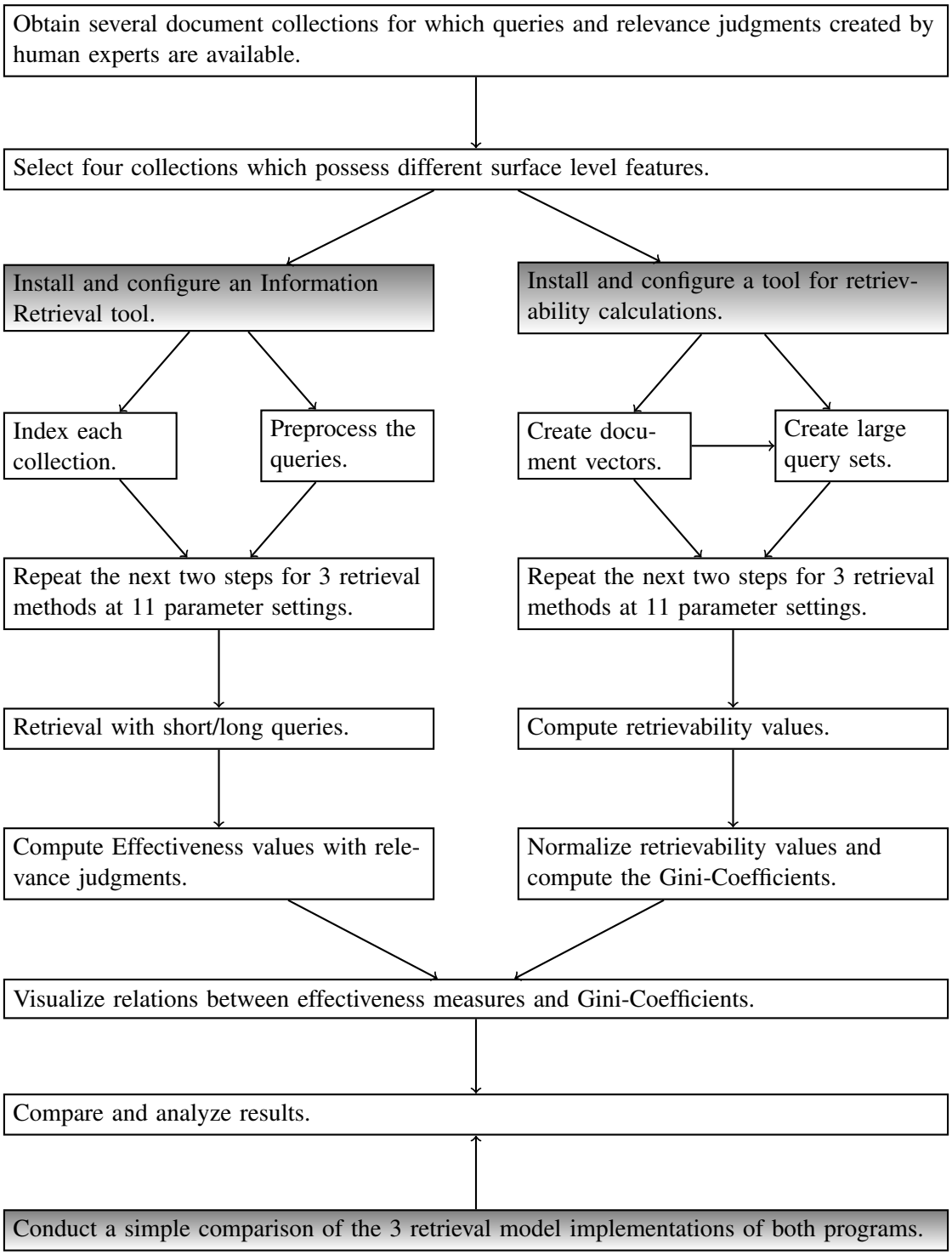


Figure 3.1: Basic steps of the experiment setup.

3.1 Hardware and Operation System

For the experiments including document preprocessing, indexing, and evaluation two different systems with the following main characteristics were used:

Laptop PC

- Intel Core i5 CPU M 460 @ 2.53GHz
- 8 Gigabyte main memory
- Ubuntu 11.10 (Oneiric Ocelot) - 3.0.0-13-generic Kernel
- Java 1.6.0_25-b06

TU Vienna Supercluster - Auxiliary Server

- 32 clusters with Intel Xeon X7560 CPU @ 2.27GHz
- 256 Gigabyte main memory
- Ubuntu 11.10 (Oneiric Ocelot) - 3.0.0-14-server Kernel
- Java 1.6.0_26-b03

All additional software used was identical:

- Lemur 4.12 with Indri 5.2
- trec eval 9.0
- Tool for Calculating Document Retrievability with Standard Retrieval Models (December 2011)
- R 2.14.1

The setup, use of these programs and parameter settings employed for the experiments will be further explained in the following subsections.

3.2 Document Collections

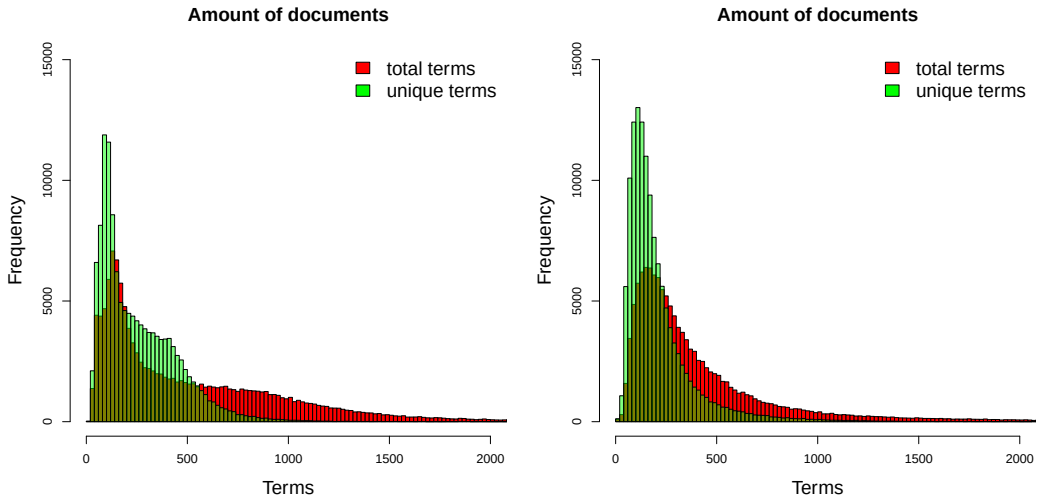
A general overview of widely used text test collections was already given in section 2.7. For this thesis four different document collections served as testbed which were selected based on the fact that they possess distinguishable surface level features and thus might require different parameter settings for the system to have good effectiveness and low retrievability bias values. The collections used are from TIPSTER Volume 1 and TREC Volume 5:

- *Los Angeles Times (LAT)* consisting of randomly selected articles from the years 1989 and 1990.
- *Foreign Broadcast Information Service (FBIS)* containing articles for 1996.
- *Federal Register (FR)* contains issues from 1989 that serve as a reporting source for actions taken by government agencies.
- *Department of Energy (DOE)* which is created out of short abstracts from the U.S. Department of Energy.

The surface level features of the collections are displayed in table 3.1 and figure 3.2. Following fields were indexed: graphic, headline, text, ti.

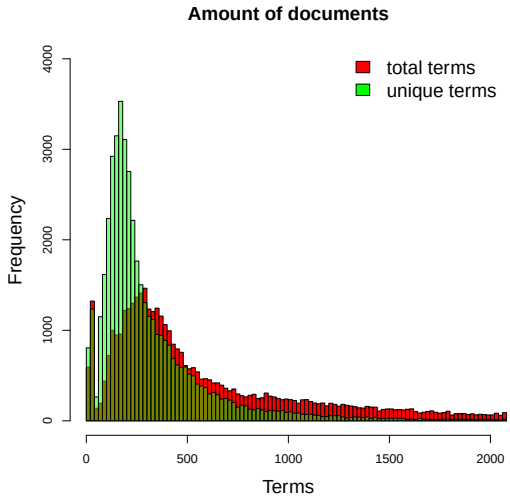
Table 3.1: Collection surface level features.

	Documents	Total Terms	Unique Terms	Average Document Length	Min Document Length	Max Document Length
Los Angeles Times	131,896	67,453,174	272,184	511.4	16	26,144
FBIS	130,471	66,675,831	290,968	511.0	9	144,218
Federal Register	45,820	68,512,915	282,563	1,495.3	0	425,711
Department of Energy	226,087	28,448,406	202,972	125.8	0	445

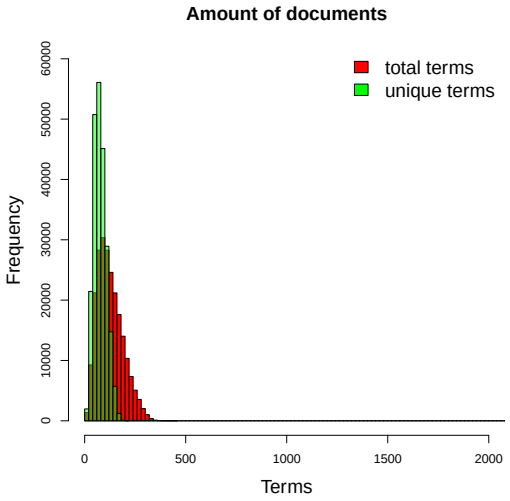


(a) LA Times

(b) Foreign Broadcast Information Service



(c) Federal Register



(d) Department of Energy

Figure 3.2: Document length distribution for collections used in the experiments.

3.3 Used Retrieval Models

As the Okapi BM25 Model and Language Modeling with Dirichet and TwoStage smoothing showed the strongest correlations in previous works by Shariq Bashir [8], these models were selected for further investigation in this thesis.

3.4 Computing the Effectiveness Measures

For both the indexing and retrieval process Lemur with indri 5.2 was used.

3.4.1 Indexing

In order to create an index, Lemur requires a parameter file containing some basic information, as depicted in listing 3.1. `index` is used to set the path for index creation. `memory` provides a rough limit to the bytes of memory the indexer will consume. `indexType` offers the choice between 'key' for KeyfileIncIndex (.key) or 'indri' for IndriIndex (.ind). `corpus/path` sets the path to the file or directory that should be indexed and `corpus/class` the type of the file(s) which can for example be 'html' for web page data, 'pdf' for Adobe PDF or 'trectext' for standard TREC collections. `field/name` specifies which field should be indexed as data and can be used multiple times. Additional parameters, for example the stemmer that should be used can be added, although for the experiments conducted here these described were sufficient. No stop word removal, case folding, stemming or lemmatization was applied [40].

Listing 3.1: Lemur parameter file - 'parameters.txt'.

```
<parameters>
  <index>/path/to/output/index</index>
  <memory>1G</memory>
  <indexType>indri</indexType>
  <corpus>
    <path>/path/to/collection/</path>
    <class>trectext</class>
  </corpus>
  <field><name>title</name></field>
  <field><name>body</name></field>
</parameters>
```

The indexing process is started by issuing the following command:

```
/path/to/indri-5.2/buildindex/IndriBuildIndex parameters.txt
```

3.4.2 Retrieval

The retrieval can be started without any parameter file. Nevertheless, due to the large amount of queries issued, these were stored in a query parameter file. This was created out of *TREC topics* and uses the same query numbering. During file creation, all punctuation marks, superfluous white spaces, and separators were removed. Listing 3.2 shows by means of an example snippet how such a file is formatted, with `number` containing the number of the query and `text` the query terms.

Listing 3.2: Lemur query parameter file - 'query.txt'.

```
<parameters>
  <query>
    <number>307</number>
    <text>New Hydroelectric Projects</text>
  </query>
  <query>
    <number>308</number>
    <text>Implant Dentistry</text>
  </query>
</parameters>
```

For the experiments sets of short and long queries are employed, which use the same relevance judgments. The short ones are created out of the `<title>` fields and the long ones from the `<desc>` fields TREC topic files contain, as shown in the fundamentals section in listing 2.2.

For the Los Angeles Times and Federal Broadcast Information Service collections, topics 301-450 (TREC-6, TREC-7 and TREC-8 ad hoc) and topics 601-700 (TREC 2003 and 2004 Robust Track) are used, resulting in a total of 250 queries for which complete relevance judgments exist. This includes topic 672 for which no relevant document exists in any of the collections used.

The same relevance judgments are not available for the Federal Register and Department of Energy collections. So for experiments with these collections, the topics 51-200 (TREC-1, TREC-2 and TREC-3 ad hoc) are employed, resulting in a set of 150 queries [30], [55].

The retrieval process itself is started by issuing variations of the following command:

```
/path/to/indri-5.2/runquery/IndriRunQuery query.txt
   -count=1000 -index=/path/to/index -trecFormat=true
   -baseline=okapi,k1:1.2,k2:0.7,b:0.7 > results.txt
```

Where `count` restricts the amount of results returned for each query, and `trecFormat` set to true ensures the output can be further processed by evaluation software like *ireval* or *trec eval* by formatting it like shown in listing 3.3. The `baseline` argument was used for okapi retrieval.

Listing 3.3: TREC formatting.

```
<queryID> Q0 <DocID> <rank> <score> <runID>
```

For retrieval experiments with language models, Dirichlet and TwoStage Smoothing were employed. This is done by issuing the following variation of the command shown above [50]:

```
/path/to/indri-5.2/runquery/IndriRunQuery query.txt
   -count=1000 -index=/path/to/index -trecFormat=true
   -rule=method:dirichlet,mu:5000 > results.txt
```

To thoroughly investigate the effects of different parameter values, the b parameter for Okapi BM25 and the λ parameter for TwoStage Smoothing were varied in steps of 0.1, from 0.0 to 1.0. For Dirichlet Smoothing the μ parameter was incremented in steps of 1000, from 0 to 10000.

3.4.3 Evaluation

The retrieved documents are evaluated against the relevance judgments provided in the *TREC query relevance* (qrel) files. The content of such a file is shown in listing 3.4 via a short clipping of three exemplary lines.

Listing 3.4: TREC query relevance file - ‘qrel.txt’.

```
161 0 DOE1-20-0168 0
161 0 DOE1-20-0902 0
161 0 DOE1-20-0913 1
```

One entry consist of four fields. The one in the first column contains the number of the corresponding topic, as used in the query parameter file. The second contains the feedback iteration which is nearly always 0 and not used. In the third field, the official document number is stored. It is followed by the relevance judgment itself in the last column which can be either 1 for relevant or 0 for not relevant [54].

The evaluation itself is done with *trec eval* which is a freely available program that allows the evaluation of TREC results using standard evaluation procedures from the *National Institute of Standards and Technology* (NIST). It can also be used to evaluate other collections as long as they get converted into TREC format first. An exemplary result set is depicted in listing 3.5.

Listing 3.5: Example trec eval result.

```
runid                   all indri
num_q                   all 249
num_ret                 all 222829
num_rel                 all 4887
num_rel_ret             all 3090
map                     all 0.2316
gm_map                  all 0.0537
Rprec                   all 0.2439
bpref                   all 0.2268
recip_rank              all 0.5608
iprec_at_recall_0.00    all 0.5811
iprec_at_recall_0.10    all 0.4708
iprec_at_recall_0.20    all 0.3855
iprec_at_recall_0.30    all 0.3057
iprec_at_recall_0.40    all 0.2548
iprec_at_recall_0.50    all 0.2142
iprec_at_recall_0.60    all 0.1597
```

```
18 iprec_at_recall_0.70    all 0.1316
19 iprec_at_recall_0.80    all 0.0996
20 iprec_at_recall_0.90    all 0.0729
21 iprec_at_recall_1.00    all 0.0632
22 P_5                     all 0.3229
23 P_10                    all 0.2522
24 P_15                    all 0.2137
25 P_20                    all 0.1878
26 P_30                    all 0.1525
27 P_100                   all 0.0731
28 P_200                   all 0.0453
29 P_500                   all 0.0222
30 P_1000                  all 0.0124
```

3.5 Computing the Retrievability Bias

In order to compute the retrievability bias for a specific model and parameter setting, it is first necessary to calculate the retrievability of every single document for that model and parameters. This can be done at different cutoff levels, which represent the willingness of the user to go down the result list.

3.5.1 Retrievability Tool

To obtain the retrievability values of the documents, a *Tool for Calculating Document Retrievability with Standard Retrieval Models* written by Shariq Bashir was employed. It is capable of calculating retrievability scores for documents with different retrieval models and requires various input files which had to be created.

First of all, the documents need to be represented as vectors containing number pairs of term ID and their frequency. Each vector has to finish with a -17 -17 end header tag. This is saved in fullText01.txt, where 01 is the vector files unique ID and can be changed, helping the user to remember or distinguish the collection he stored in it. The basic structure of such a file is shown in listing 3.6, containing three example document

vectors. To improve visibility and help understand the format, here the term frequencies are highlighted bold, although the real file itself only contains plain text.

Listing 3.6: Retrievability tool file - ‘fullText01.txt’.

```
1 1 14 2 4 3 7 4 7 -17 -17
2 1 4 8 3 27 4 36 11 56 4 57 8 62 5 67 3 -17 -17
3 2 2 3 11 8 2 9 5 11 1 12 1 13 3 56 1 -17 -17
```

Additionally a file called ItemsetProcessing01.txt, using the same unique ID as the vector file it belongs to, stores that vector files total amount of vectors and the highest term ID it contains. An example with content matching the fullText01.txt shown above is depicted in listing 3.7, where 3 is the total amount of vectors and 67 the ID of the highest term.

Listing 3.7: Retrievability tool file - ‘ItemsetProcessing01.txt’.

```
1 3
2 67
```

Also, a set of queries is required. The name of this file containing them can be chosen freely, so let’s call it querySet01.txt. The terms for the queries are represented by the same term IDs used in the fullText01.txt file and it uses a single -17 end tag for each query. An example content is given in listing 3.8.

Listing 3.8: Retrievability tool file - ‘querySet01.txt’.

```
1 1 2 4 -17
2 1 27 36 62 -17
3 3 11 12 -17
```

Finally, a file called Settings.txt is needed. It contains information about the total number of threads the program should use, the name of the query set file employed for the computations and entries for five cutoff levels which should be used for the retrievability calculation. Four additional settings that allow the use of extra term-posting and cluster files as well as their split detection were not required for the experiments conducted

here and marked as such in the Settings.txt. The content of listing 3.9 shows the settings employed.

Listing 3.9: Retrievability tool file - 'Settings.txt'.

```
1 25
2 query_set_long.txt
3 10
4 30
5 50
6 100
7 150
8 not required
9 not required
10 not required
11 not required
```

The retrievability tool is started by issuing this command:

```
/path/to/tool/RetrievabilityTool /path/to/myfiles
   <vectorFileID> <NN> <vectorAmount> <NN> <ModelID>
   <ModelParameter>
```

The <vectorFileID> represents the unique ID of the vector file and the amount of vectors it contains is submitted via the <vectorAmount> argument. <ModelID> is for example 2 for Okapi BM25 with <ModelParameter> having a value between 0 and 1 to set the b variable. Arguments denoted with <NN> are not needed and initialized with a value of 0. Once finished, the program creates an output file containing six different retrievability values for each document vector. These are the values at the five rank cutoff levels defined in the Settings.txt file and the total number of queries that retrieved the document without any cutoff being considered [9].

3.5.2 Document Preprocessing

A document vector tool was written to parse the same document fields used for the effectiveness calculations with Indri into a vector file which satisfies the input format need of the retrievability tool described in the previous section.

It is started with the following command, containing two program arguments:

```
/path/to/tool/VectorTool /path/to/collection <fieldsToUse>
```

The first argument is the main directory of the collection. The `<fieldsToUse>` argument contains all fields within each document that should be included in the conversion process. The different fields are separated by comma, with 'title,text,graphic' being a possible example for that argument.

3.5.3 Query Generation

To calculate the retrievability, a large set of queries is needed. These were created out of the vector files obtained in the document preprocessing. For this purpose a query generation tool was written which uses for each collection all terms contained as one-term queries. For the creation of two-term queries the following procedure is employed to obtain a reasonable amount. All terms with a collection frequency equal or greater than 20 percent of the amount of documents in that collection are considered too generic and excluded. For each document out of all remaining terms that occur more than once within it, two-term combinations are generated in a way that does not produce any permutations or duplicate queries. Finally, the set of one-term and two-term queries are merged. The amount of resulting queries per document collection is shown in table 3.2 for the different collections used.

Table 3.2: Queries generated for the retrievability computation.

	Indexed Fields	One Term	Two Term	Total
Los Angeles Times	headline, text, graphic	272,184	56,197,490	56,469,674
FBIS	ti, text	290,968	87,770,416	88,061,384
Federal Register	text	282,563	249,253,111	249,535,674
Department of Energy	text	202,972	7,636,442	7,839,414

3.5.4 Retrievability Normalization and Gini-Coefficient

The retrievability was calculated for the total set of combined queries at five different rank cutoff levels. As proposed by Bashir in [8, p. 27], the retrievability values were normalized to compensate the fact that long and vocabulary rich documents generate more queries than short documents and therefore retrievability scores based on these queries would favor long documents. The normalization is done by dividing the retrievability values of each document at the desired rank cutoff level through the retrievability value produced without any rank cutoff. This represents the total amount of queries that are potentially able to retrieve that document.

For the calculation of the Gini-Coefficient, *R* was used which is an environment and language for statistical computing. By employing the *reldist* package, the Gini-Coefficient can be computed out of the previously created vector files by issuing the following commands in R [23], [42]:

```
library(reldist)
gini(read.table("BM25_full_0.100000.txt", skip=6)[[1]])
```

Where `BM25_full_0.100000.txt` is the name of a result file created by the retrievability calculation tool described in subsection 3.5.1, `skip=6` makes sure the parsing starts where the actual data starts and `[[1]]` selects the first column, which was set to store the results at a cutoff level of 10.

As R also allows the component wise division of vectors, the normalized Gini-Coefficient $\hat{r}$ can be calculated by the use of the sixth column which contains the retrievability without any cutoff level considered. These are the retrievability bias values used for the comparison with the effectiveness measures and are computed as follows:

```
library(reldist)
gCutoff10 <- read.table("BM25_full_0.100000.txt", skip=6)[[1]]
gNoCutoff <- read.table("BM25_full_0.100000.txt", skip=6)[[6]]
gini(gCutoff50/gNoCutoff)
```

3.6 Retrieval Model Implementation Similarity

A lot of possible variations exist for the Okapi BM25, Dirichlet and TwoStage Smoothing formulas. Before starting to draw conclusions from the experimental results, a simple comparison of their implementation in *Lemur with Indri* and the *Tool for Calculating Document Retrievability with Standard Retrieval Models* is conducted. For this, the ranking created by a single query issued on the Los Angeles Times collection is compared for both programs. The collection consists of 131,896 documents. For this purpose, the short query from TREC topic 301 is employed, 'International Organized Crime'. Parameter settings used are b = 0.5 for Okapi BM25, μ = 5000 for Dirichlet Smoothing and λ = 0.5 for TwoStage Smoothing.

To create a ranking with the retrievability tool, it has to be executed several times to calculate all cutoff levels from 1 to 10. The ranking can then be created manually by adding each document that is new at a higher cutoff level to the bottom of a ranked list. The original TREC document number can be acquired by looking up the line number from the retrievability result file in a document number mapping file that was created as a byproduct of the fullText01.txt. Finally, the Lemur ranking of that document number can be directly read from a Lemur query result file.

A comparison of both rankings is depicted in table 3.3, where the rank position returned by Lemur is displayed for the top 10 retrievable documents for three different retrieval methods.

Table 3.3: Lemur rank position of the top 10 retrievable documents for three different retrieval models.

	Okapi BM25	Dirichlet	TwoStage
1	1	8	2
2	2	2	7
3	4	3	3
4	3	7	8
5	20	10	15
6	18	13	12
7	21	31	30
8	8	28	28
9	6	35	14
10	34	15	31

While the representative power of this single query sample is not strong, considering these numbers it at least suggests that the implementations of all three retrieval models are mostly similar in both programs. Implementation differences will therefore only minimally distort the direct comparison of effectiveness and retrievability bias values acquired in the experiment.

CHAPTER 4

Experimental results

This chapter will thoroughly examine and analyze the experimental results for the three retrieval models employed and explain observations by means of examples. The complete result set as visualizations and tables is attached in the appendices A and B.

4.1 Basic Information

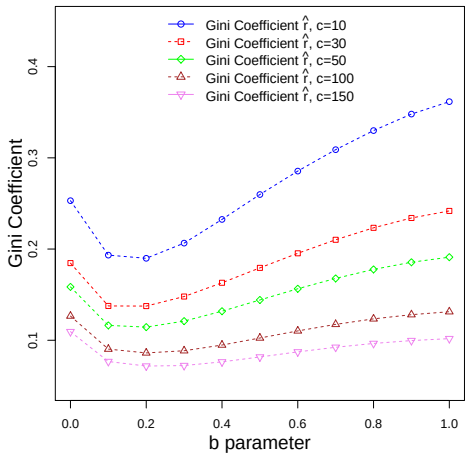
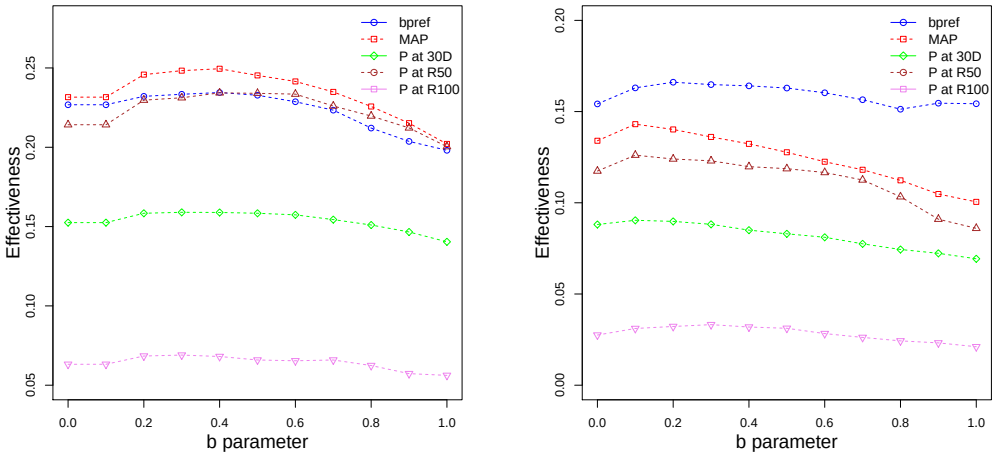


Figure 4.1: Gini-Coefficient at different cutoff levels, Okapi BM25, Los Angeles Times.

The Gini-Coefficients are calculated at several cutoff levels c, namely 10, 30, 50, 100 and 150. As shown in figure 4.1 on an example with Okapi BM25 and the Los Angeles Times collection, the general shapes of the curves barely change at the different cutoff levels. The setting resulting in the lowest Gini-Coefficient slightly fluctuates. As the relative difference between the Gini-Coefficient values is in most cases more prominent with a low cutoff level, c =10 is chosen for all further comparisons. For similar visualizations of Gini Coefficients at different cutoff levels for all other document collections and retrieval models used in the experiments, see figures A.1, A.7, A.13 in the appendix.

When comparing the results of Binary Preference, Mean Average Precision, Precision at 30 documents, Precision at Recall 50 and Precision at Recall 100 with each other it becomes obvious that the general shape of their visualization curves usually have a strong similarity. This is true across all collections and for all three retrieval models and shown with an exemplary visualization in figure 4.2. Only a few minor fluctuations can be noticed, like the rise in performance of Binary Preference for long queries on high settings of b in this example. This observation suggests that a parameter setting which is good for one of the effectiveness measures in most cases also delivers good results for the others.



(a) Short queries

(b) Long queries

Figure 4.2: Comparison of different effectiveness measures, Okapi BM25, Los Angeles Times.

The following sections report noticeable results of the three retrieval models employed. Each of them contains a table, namely 4.1, 4.2, 4.3, which shows the performance rank of different effectiveness measures at the same parameter settings that minimize the retrieval bias represented by the Gini-Coefficient, where the best possible rank is 1 and the worst performing one is 11.

Azzopardi and Bache notice in [3], that in their experiment the parameter setting leading to the lowest retrieval bias does not directly correspond with the one maximizing effectiveness. Nonetheless, the difference in performance is usually quite small. Therefore, in the three comparison tables mentioned above, the achieved effectiveness at the suggested parameter setting is also expressed as a percent value in relation to the best possible effectiveness that can be reached with one of the 11 different settings.

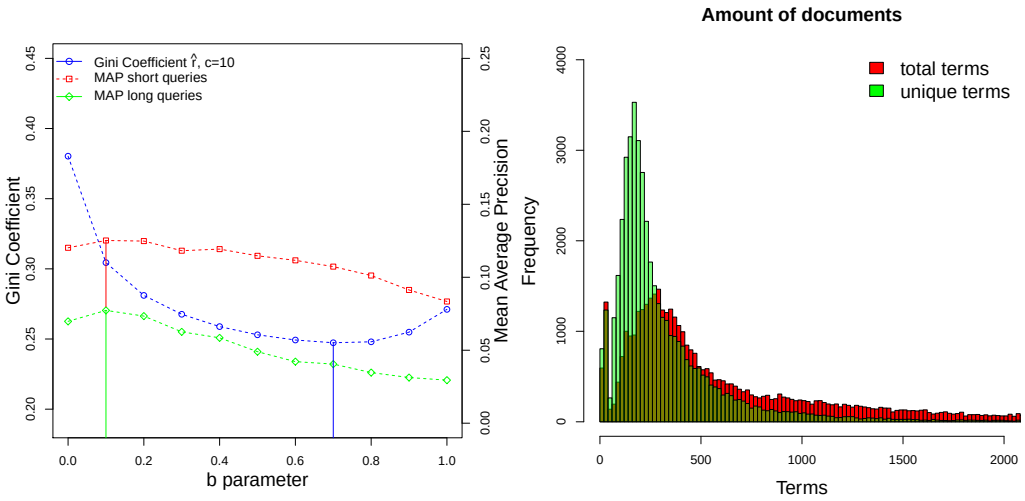
4.2 Okapi BM25

Table 4.1: Effectiveness measures ranking at minimum Gini-Coefficient, Okapi BM25.

	Short Queries					Long Queries				
	Binary Preference	Mean Average Precision	Precision at 30 documents	Precision at Recall 50	Precision at Recall 100	Binary Preference	Mean Average Precision	Precision at 30 documents	Precision at Recall 50	Precision at Recall 100
LA Times	4 99.0%	3 98.5%	5 99.6%	5 98.0%	2 99.1%	1 100%	2 98.0%	2 99.3%	2 98.3%	2 97.0%
FBIS	3 95.8%	3 94.3%	2 98.9%	3 95.3%	3 87.0%	2 98.2%	1 100%	3 98.6%	1 100%	1 100%
Federal Register	7 92.0%	8 85.7%	7 86.5%	8 82.6%	8 82.8%	9 84.3%	8 52.4%	8 75.6%	8 49.4%	8 17.2%
DOE	4 91.7%	4 94.0%	3 99.6%	4 90.9%	4 89.7%	4 98.2%	4 98.2%	4 99.3%	3 96.7%	3 66.7%
Average	4.5 94.6%	4.5 93.1%	4.25 96.2%	5 91.7%	4.25 89.7%	4 95.2%	3.75 87.2%	4.25 93.2%	3.5 86.1%	3.5 70.2%

As table 4.1 shows, the Federal Register collection has low effectiveness values at the parameter setting that leads to the lowest and thus considered optimum Gini-Coefficient. This is especially true for long queries, where the suggested parameter setting only leads to a Mean Average Precision value with 52.4 percent performance of the optimum Mean Average Precision that can be obtained. This setting is ranked 8th out of 11 possible ranks. What also stands out for this collection is the fact that with the parameter tuning employed in the experiment here, the retrievability bias cannot be reduced as much as for other collections. This is not only true for the Okapi BM25 retrieval where it is most prominent, but also for the Dirichlet and TwoStage Smoothing methods.

The visualization for this collection shows the same characteristic as the *Associated Press* and *Wall Street Journal* TREC collections investigated with Okapi BM25 retrieval by Azzopardi and Bache in [3]. There the minimum Gini-Coefficient does not directly correspond with the setting that maximizes the Mean Average Precision either and suggests a b value that is shifted 0.3 or 0.4 higher than the optimum. Here this phenomenon is actually even stronger and the shift has a magnitude of 0.6 on the b parameter scale. To further investigate a possible cause for this behavior a comparison with the document length distribution is conducted, which is depicted in figure 4.3.

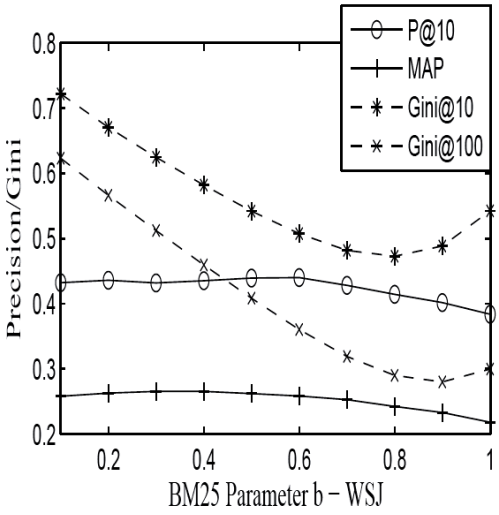


(a) Suggested b parameter higher than optimum

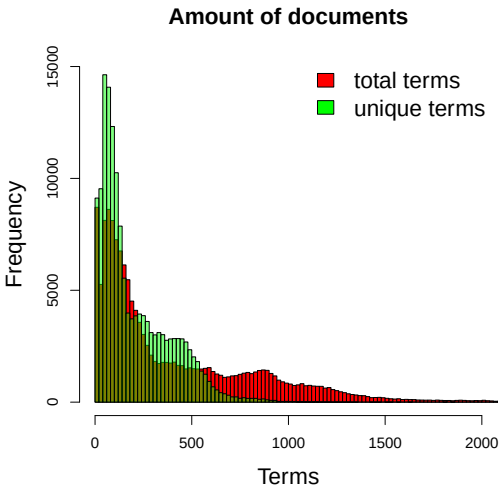
(b) Document length distribution, FR

Figure 4.3: Possible relation between parameter shift and document length distribution, Okapi BM25, Federal Register.

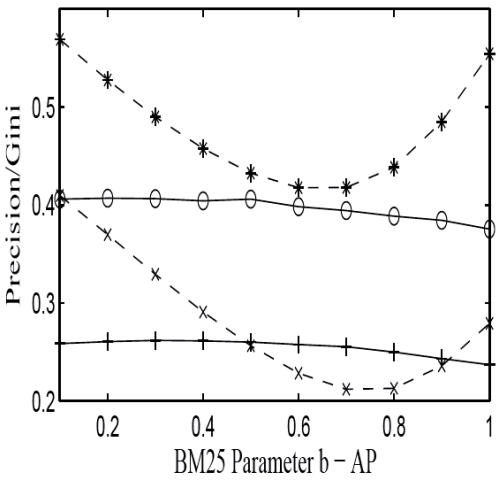
A large amount of very short documents are present in the first histogram bins of the total term length distribution, then a dent occurs and the next bin only contains about 20 percent as many documents. This raises the question if something similar can be observed for the collections Azzopardi and Bache used, as visualized in figure 4.4.



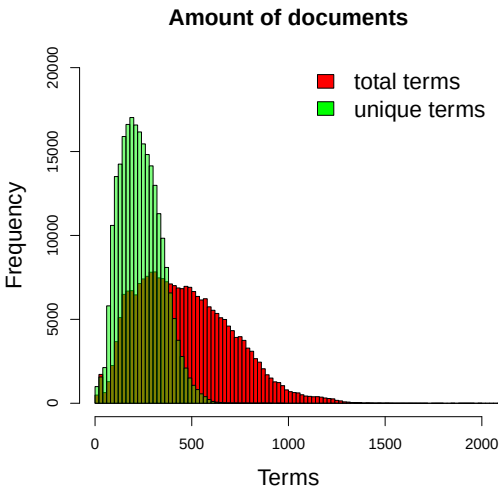
(a) Suggested b parameter higher than optimum



(b) Document length distribution, WSJ



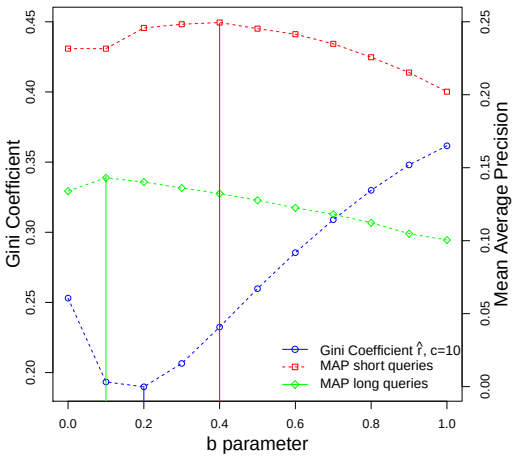
(c) Suggested b parameter higher than optimum



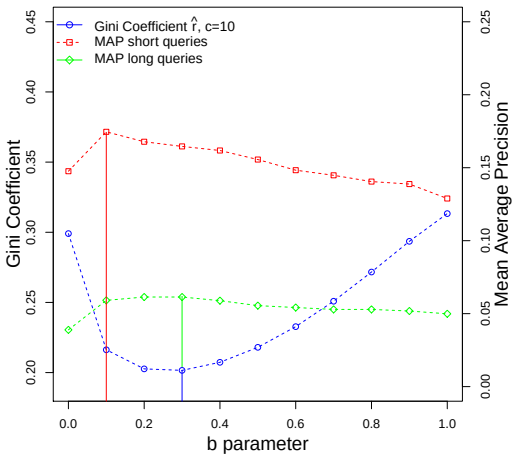
(d) Document length distribution, AP

Figure 4.4: Possible relation between parameter shift and document length distribution, Okapi BM25, Wall Street Journal (WJS) and Associated Press (AP), adapted from Azzopardi and Bache [3].

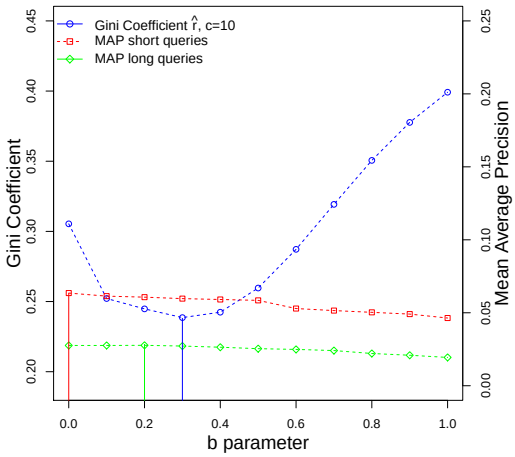
For the Wall Street Journal and Associated Press collections only results for one type of query are available. The dents in the document length distribution histograms are less prominent then observed with the Federal Register collection. However, there seems to be a proportional relation between the size of the dent in the distribution and the magnitude of the b parameter shift to a higher value than the optimum.



(a) Los Angeles Times



(b) Foreign Broadcast Information Service



(c) Department of Energy

Figure 4.5: High effectiveness at minimum retrievability bias, Okapi BM25.

For the other three collections employed in this thesis, namely Los Angeles Times, Foreign Broadcast Information Service and Department of Energy, the resulting effectiveness values are exceptionally good, for both short and long queries. A very strong correlation between low Gini-Coefficient and high effectiveness is noticeable across all parameter settings, which is shown in figure 4.5, again exemplary by means of the Mean Average Precision. The parameter shift observed here has usually a value of 0.1 or 0.2 and the distribution visualizations show no dents, see figure 3.2. The parameter shift also depends on the optimization goal, whether retrieval should be optimized for short or long queries.

Correlations with other effectiveness measures, which generally behave the same, are shown in the appendix in figures A.2, A.4, A.5, A.6

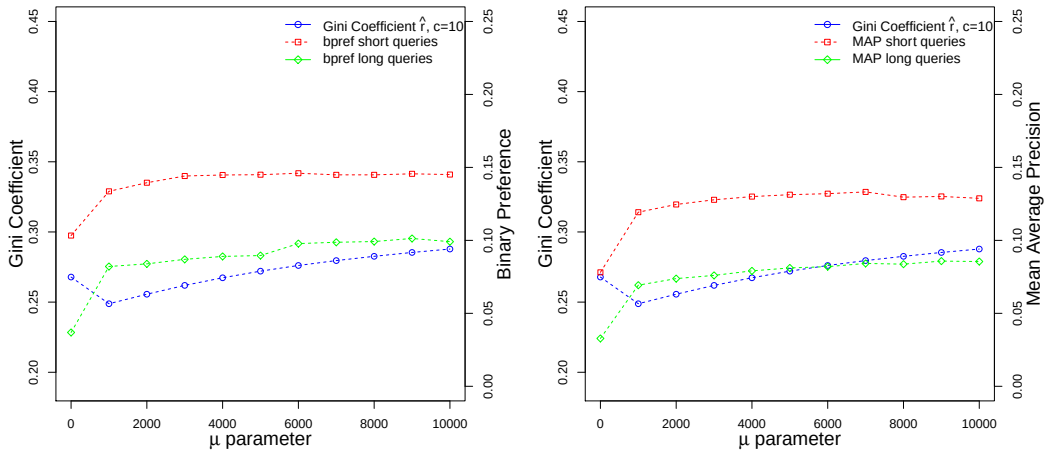
4.3 Language Model with Dirichlet Smoothing

Table 4.2: Effectiveness measures ranking at minimum Gini-Coefficient, Dirichlet Smoothing.

	Short Queries					Long Queries				
	Binary Preference	Mean Average Precision	Precision at 30 documents	Precision at Recall 50	Precision at Recall 100	Binary Preference	Mean Average Precision	Precision at 30 documents	Precision at Recall 50	Precision at Recall 100
LA Times	3 97.0%	3 98.0%	3 98.7%	3 95.5%	3 97.0%	4 99.5%	8 98.6%	8 98.6%	3 98.8%	8 92.2%
FBIS	1 100%	3 99.8%	1 100%	1 100%	9 95.8%	4 94.2%	9 94.0%	1 100%	9 91.9%	9 84.7%
Federal Register	10 91.5%	10 89.6%	10 91.2%	10 94.3%	10 92.4%	10 81.0%	10 80.8%	10 82.3%	10 72.3%	10 83.8%
DOE	1 100%	1 100%	1 100%	1 100%	1 100%	10 94.1%	10 79.8%	10 92.3%	10 71.6%	10 76.9%
Average	3.75 97.1%	4.25 96.9%	3.75 97.5%	3.75 97.5%	5.75 96.3%	7 92.2%	9.25 88.3%	7,25 93.3%	8 83.7%	9.25 84.4%

The results with Dirichlet Smoothing show only slight fluctuations in performance for most parameter settings. Nonetheless, the suggested settings perform exceptionally well for short queries. A difference in the slope of short and long queries issued is noticeable, with long queries always having slightly better performance at high parameter settings. The retrievability bias suggests low settings of μ between 1000 and 3000 for all four collections. Which raises the question if this is due to the fact that the exhaustive set of queries used for retrievability calculations only consisted of one-term and two-term queries and thus their suggested settings might favor short queries.

Again the Federal Register collection is an exception where the best effectiveness values for both short and long queries can actually be found at the setting with the highest retrievability bias. However, in this case the bias difference at different parameter settings is very small and therefore probably not the most important factor influencing the effectiveness, as shown in figure 4.6.



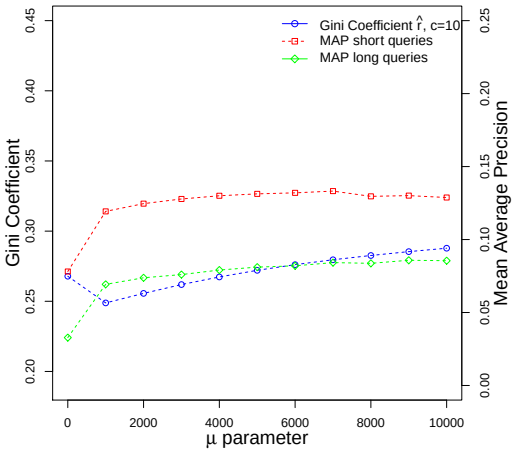
(a) Binary Preference

(b) Mean Average Precision

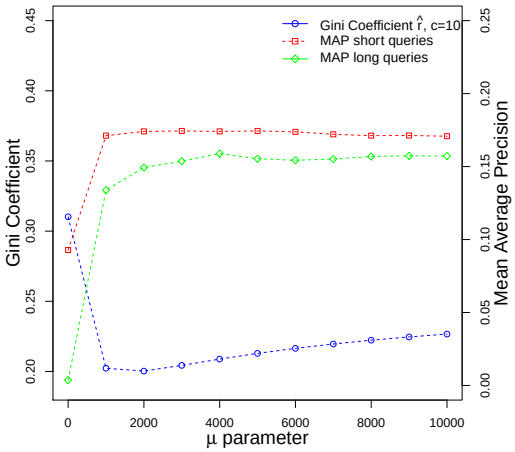
Figure 4.6: High effectiveness at maximum retrievability bias, Dirichlet Smoothing, Federal Register.

Comparing the results of different collections with each other shows that a strong change in the retrievability bias value causes a strong change in effectiveness. The extend of this change seems to be proportional as shown in figure 4.7 by means of the Mean Average Precision value of three collections.

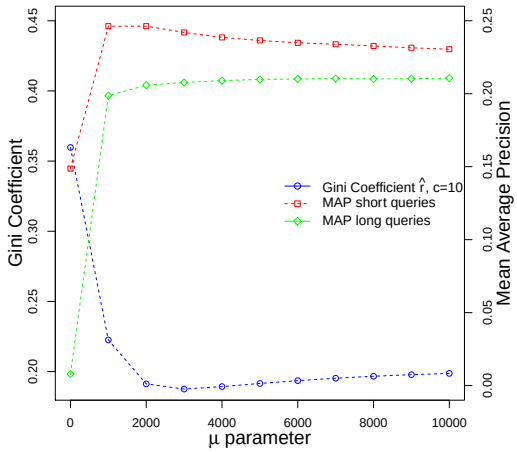
Additionally, this figure shows that with Dirichlet Smoothing the collections with lower retrievability bias generally have noticeable higher Mean Average Precision values.



(a) Federal Register



(b) Foreign Broadcast Information Service



(c) Los Angeles Times

Figure 4.7: A strong change in the retrievability bias at low μ parameters causes a proportional change in the Mean Average Precision value, Dirichlet Smoothing.

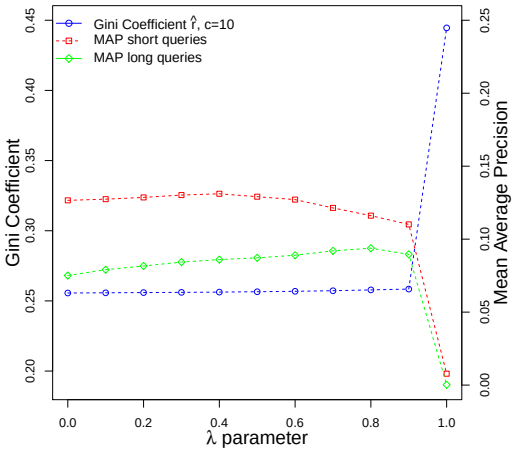
4.4 Language Model with TwoStage Smoothing

Table 4.3: Effectiveness measures ranking at minimum Gini-Coefficient, TwoStage Smoothing.

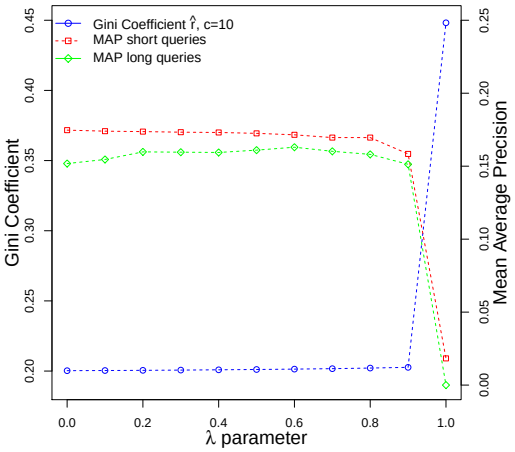
	Short Queries					Long Queries				
	Binary Preference	Mean Average Precision	Precision at 30 documents	Precision at Recall 50	Precision at Recall 100	Binary Preference	Mean Average Precision	Precision at 30 documents	Precision at Recall 50	Precision at Recall 100
LA Times	1 100%	1 100%	1 100%	1 100%	2 99.7%	10 98.0%	10 95.5%	10 95.5%	10 96.7%	10 79.2%
FBIS	1 100%	1 100%	1 100%	1 100%	10 92.0%	7 94.9%	8 93.1%	8 97.3%	10 90.0%	10 86.9%
Federal Register	6 97.4%	7 96.6%	6 96.6%	2 99.9%	8 92.6%	9 79.4%	10 80.0%	10 86.6%	10 73.6%	10 74.8%
DOE	10 91.7%	10 89.7%	10 77.5%	10 85.6%	10 91.9%	10 92.7%	7 89.9%	10 94.6%	4 88.7%	5 94.2%
Average	4,5 97.3%	4.75 96.6%	4.5 93.5%	3.5 96.4%	7.5 94.1%	9 91.3%	8.75 89.6%	9.5 93.5%	8,5 87.3%	8,75 83.8%

While the retrievability bias results for TwoStage Smoothing are even more linear than with Dirichlet Smoothing, the setting at the lowest Gini-Coefficient once again favors the performance of short queries over those of long ones.

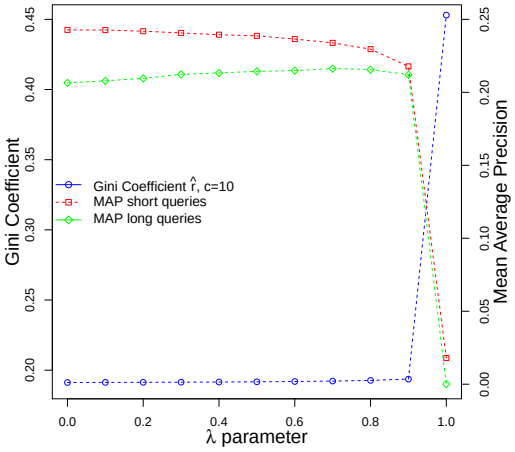
The same strong correlations already presented in figure 4.7 for Dirichlet Smoothing can also be observed for TwoStage Smoothing across the collections. A noticeable lower retrievability bias leads to noticeable higher effectiveness, as shown in figure 4.8 with Mean Average Precision.



(a) Federal Register



(b) Foreign Broadcast Information Service



(c) Los Angeles Times

Figure 4.8: A strong change in the retrievability bias at high λ parameters causes a proportional change in the Mean Average Precision value, TwoStage Smoothing.

Due to the fact that there are only very minor changes in the Gini-Coefficient values at ten out of eleven λ parameter settings for TwoStage Smoothing, obtained results offer no clear guideline to help to select the best performing setting, only to avoid the worst.

CHAPTER 5

Conclusions

This section will sum up major findings while trying to answer the original three questions from subsection 1.2 that served as foundation for the experiments conducted in this thesis. Finally, incitations for further work to be conducted based on these findings are given.

5.1 Summary

For all three retrieval models both visual and arithmetic comparisons clearly show that there is a very strong relationship between a low retrievability bias represented by a normalized Gini-Coefficient and the effectiveness measures. This relationship is both observable at different parameter settings on the same collection as well as across different document collections. Nonetheless, the assumption that the setting with the lowest retrievability bias always leads to a top effectiveness value cannot be confirmed in all cases. While the parameter settings can be tuned this way and often a good result is obtained, sometimes the results can also turn out slightly below average. The correlation between retrievability bias and effectiveness seems to be most prominent with Okapi BM25 retrieval. However, tuning the parameters via the retrievability bias always helps to avoid the worst settings, on all three retrieval methods employed in the experiments.

The observations in this thesis raise the suspicion that there might be other important factors influencing effectiveness which have to be considered and weighted in addition

to the retrievability bias in order to improve the derivation of top performing parameter settings. A parameter shift away from the top performing one was especially noticed with the Federal Register collection and Okapi BM25 retrieval. Observations showed dents in the collection distribution histograms which are proportional to these shifts. With these, it might be possible to effectively predict and compensate major shifts and thus always allow to select good performing settings for Okapi BM25, making it the most promising retrieval model investigated here.

The retrievability bias values were computed by issuing sets of very short queries, as they are less time consuming to create and process than exhaustive sets of long ones. The derived parameter settings were for retrieval with both Dirichlet and TwoStage Smoothing usually better suited for the effectiveness optimization of short user issued queries, while for Okapi BM25 retrieval they slightly favored long queries. Consequently, it also seems to be important to know for what kind of use-case and query formulation the system should be optimized. Very short and precise or rather long and rough queries clearly require different parameter settings for maximum performance.

5.2 Future work to be done

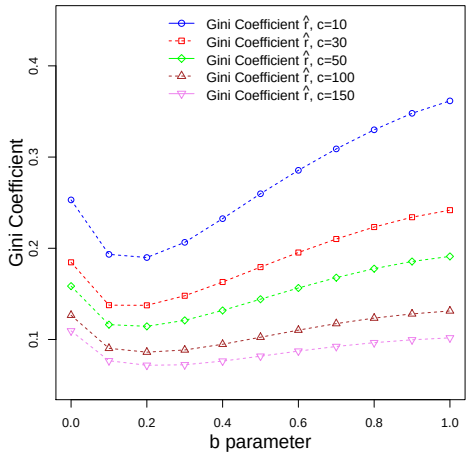
Further investigation should be conducted on the way the query generation for the retrievability bias calculation influences the predicted parameter settings and if certain query generation processes are better suited for specific use-cases like different types of user issued queries. While the Gini-Coefficient rankings are nearly the same for different retrievability cutoff levels, there also exist some minor fluctuations as to where the lowest can be found. Which cutoff level is best suited for predicting the optimum settings for effectiveness and if this is consistent across different collections requires additional studies.

For the phenomenon of suggested parameter settings with Okapi BM25 sometimes clearly not corresponding to those needed for maximum effectiveness, a hypothesis for a possible error compensation was presented by paying attention to dents in the total term document length distribution. This needs further investigation across a broad spectrum of different collections. Also other collection surface level features that influence either the effectiveness or the retrievability bias might help to further tune and correct parameter shifts.

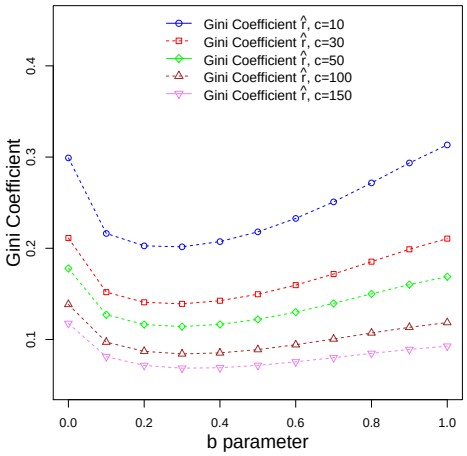
Another approach worth pursuing might be the combination of different methods. The ranking by pseudo-relevance judgments proposed by Soboroff in [49] or by the retrieval system similarity as proposed by Aslam in [2] could also be adapted for different parameter settings of the same retrieval system. While both methods were not good at predicting the top performers, they might help refining the parameter selection of the retrievability bias method by indicating and thus preventing those occasional below average settings.

APPENDIX A

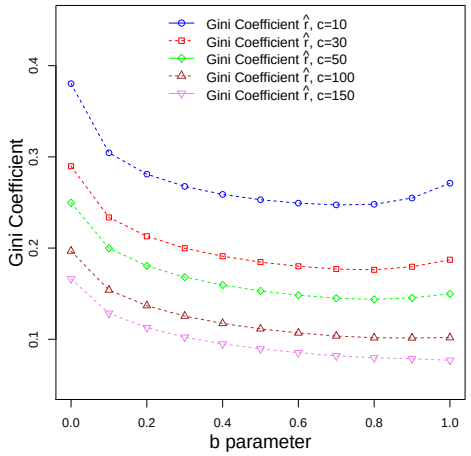
Experimental Result Figures



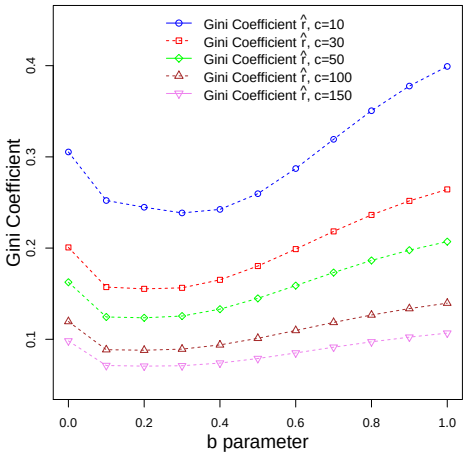
(a) Los Angeles Times



(b) Foreign Broadcast Information Service

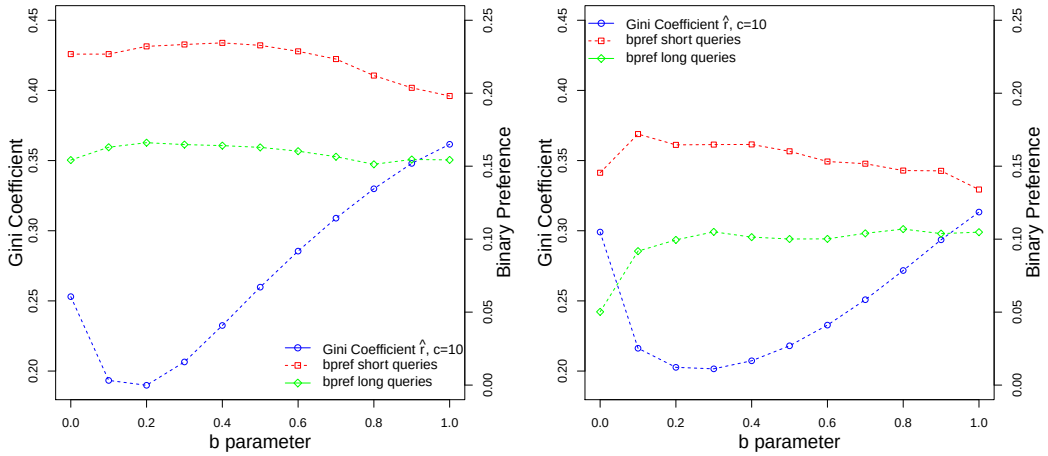


(c) Federal Register



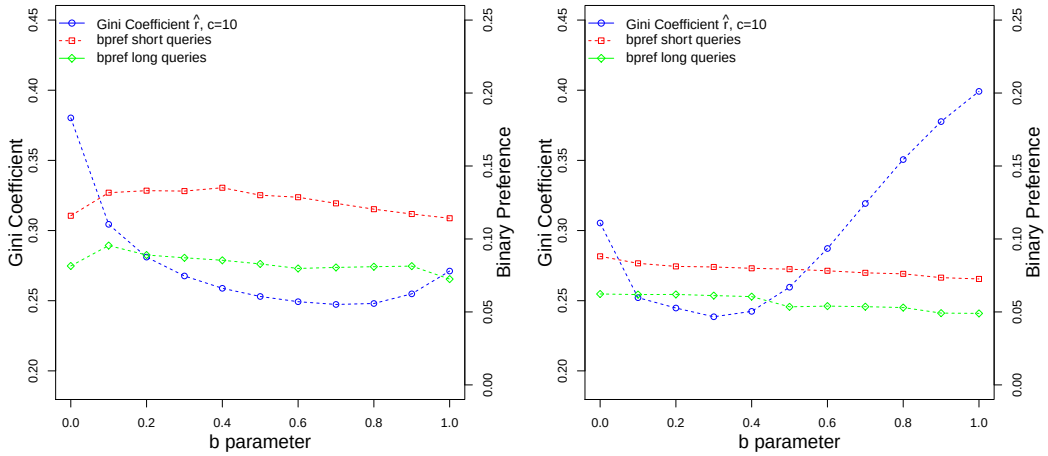
(d) Department of Energy

Figure A.1: Gini-Coefficient at five cutoff levels across different parameter values b with Okapi BM25. The general shapes remain the same while the parameter setting resulting in the lowest Gini-Coefficient slightly fluctuates.



(a) Los Angeles Times

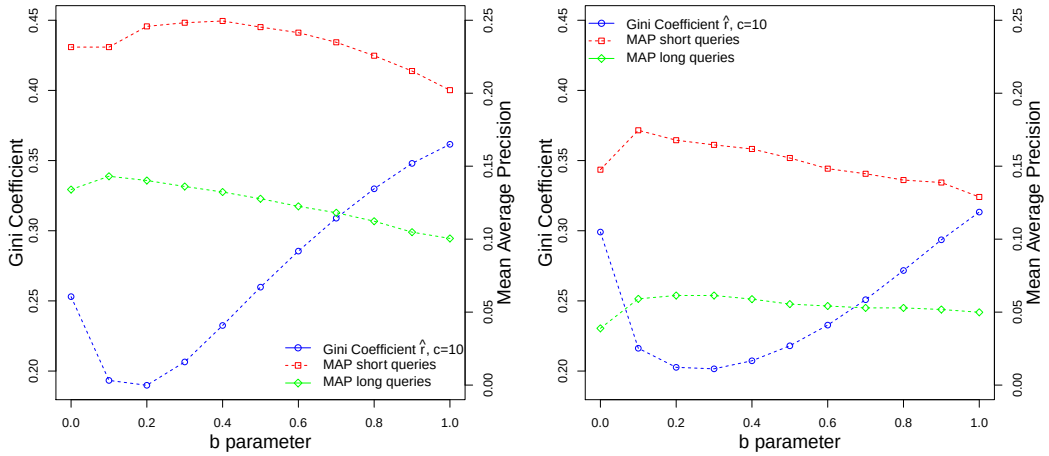
(b) Foreign Broadcast Information Service



(c) Federal Register

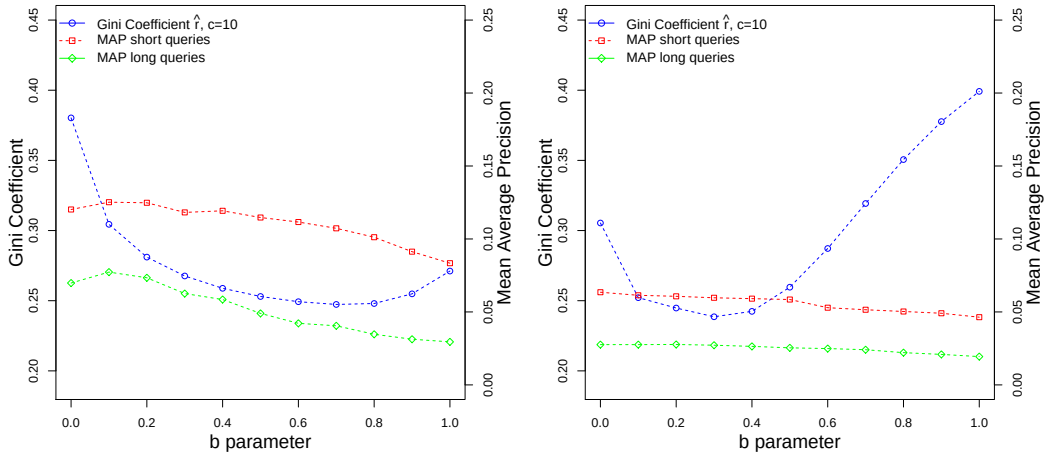
(d) Department of Energy

Figure A.2: Relationship between Gini-Coefficient and Binary Preference across different parameter values b with Okapi BM25. A low Gini-Coefficient usually correlates with a high Binary Preference value, both within the same collection and across different ones. For the Federal Register collection a parameter shift occurs which results in low effectiveness at the minimum Gini-Coefficient, especially with long queries.



(a) Los Angeles Times

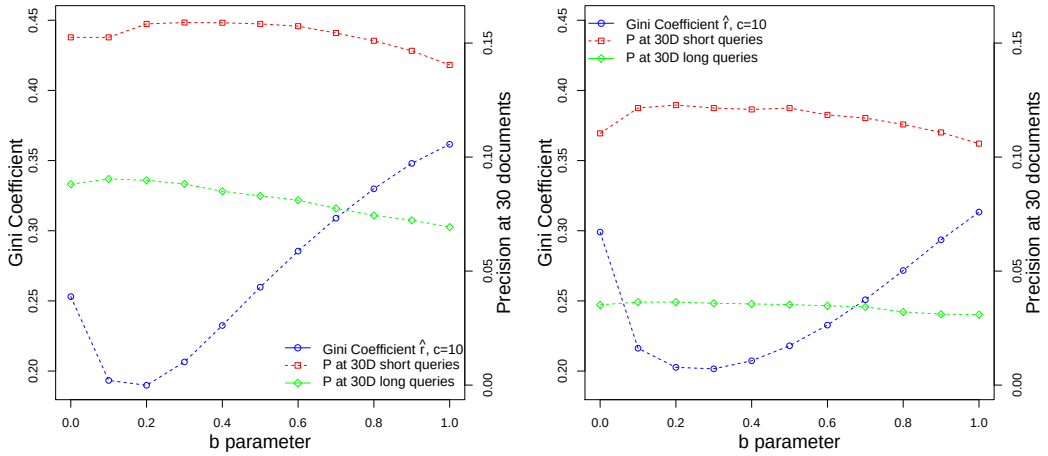
(b) Foreign Broadcast Information Service



(c) Federal Register

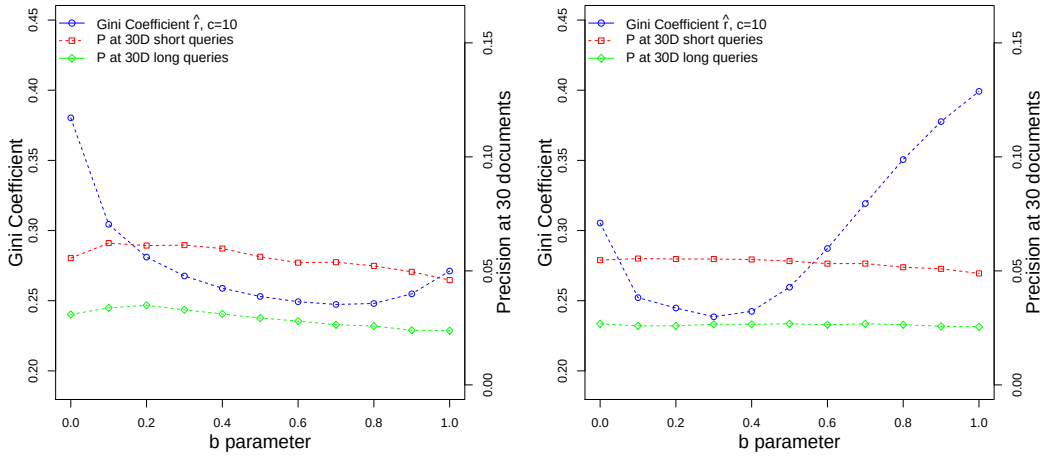
(d) Department of Energy

Figure A.3: Relationship between Gini-Coefficient and Mean Average Precision across different parameter values b with Okapi BM25. A low Gini-Coefficient usually correlates with a high Mean Average Precision value, both within the same collection and across different ones. For the Federal Register collection a parameter shift occurs which results in low effectiveness at the minimum Gini-Coefficient, especially with long queries.



(a) Los Angeles Times

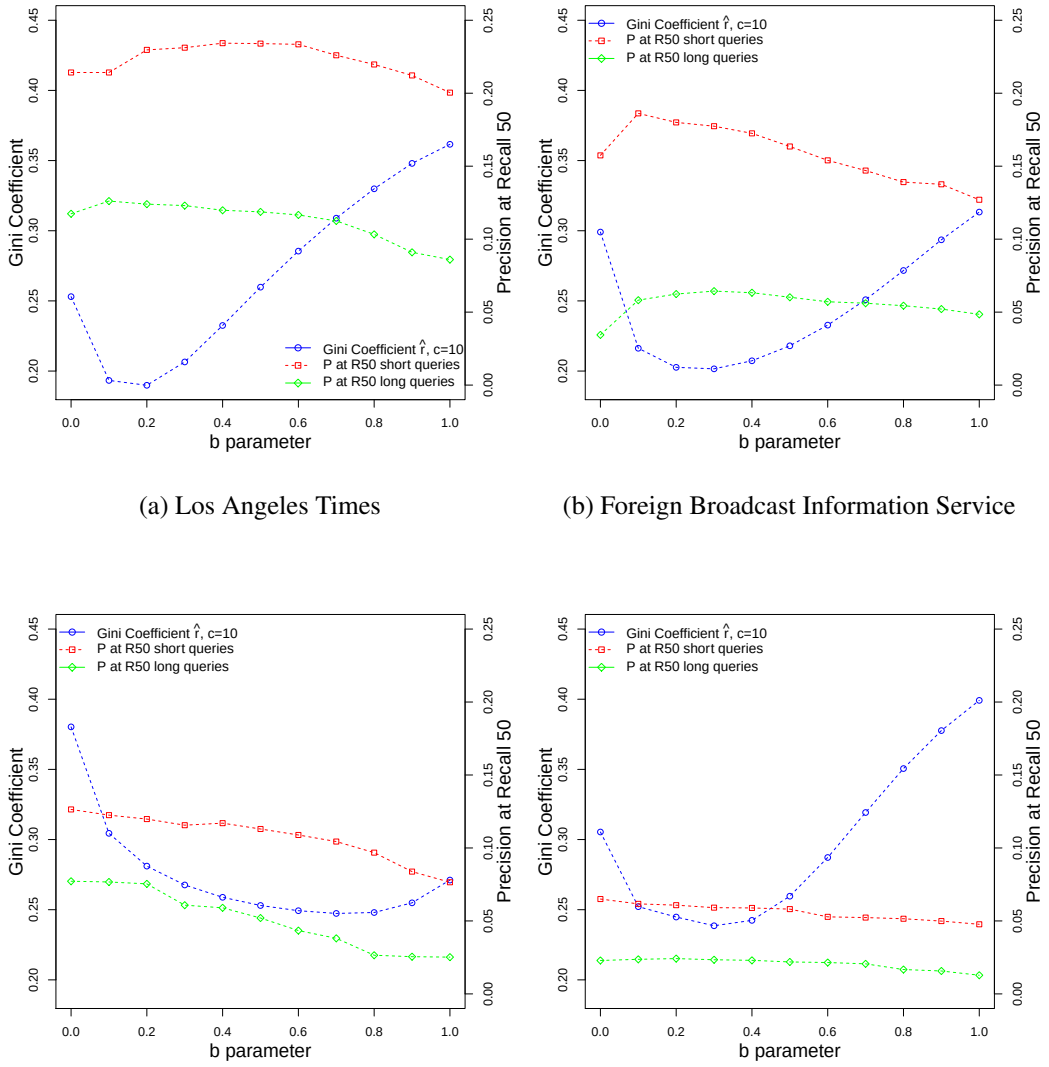
(b) Foreign Broadcast Information Service



(c) Federal Register

(d) Department of Energy

Figure A.4: Relationship between Gini-Coefficient and Precision at 30 documents across different parameter values b with Okapi BM25. A low Gini-Coefficient usually correlates with a high Precision at 30 documents value, both within the same collection and across different ones. For the Federal Register collection a parameter shift occurs which results in low effectiveness at the minimum Gini-Coefficient, especially with long queries.



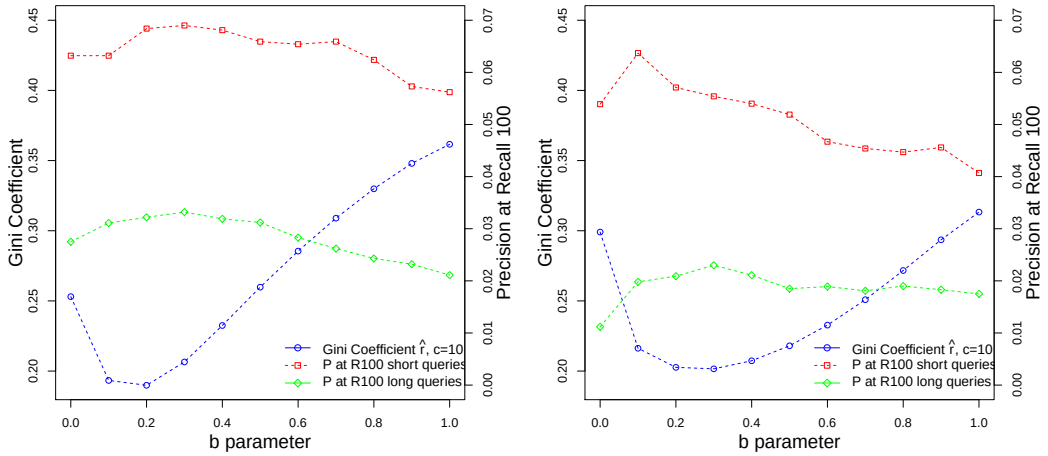
(a) Los Angeles Times

(b) Foreign Broadcast Information Service

(c) Federal Register

(d) Department of Energy

Figure A.5: Relationship between Gini-Coefficient and Precision at Recall 50 across different parameter values b with Okapi BM25. A low Gini-Coefficient usually correlates with a high Precision at Recall 50 value, both within the same collection and across different ones. For the Federal Register collection a parameter shift occurs which results in low effectiveness at the minimum Gini-Coefficient, especially with long queries.



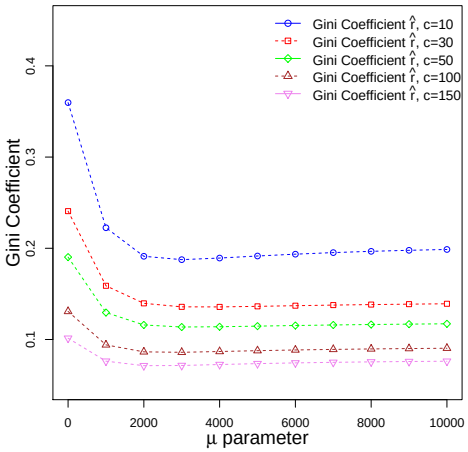
(a) Los Angeles Times

(b) Foreign Broadcast Information Service

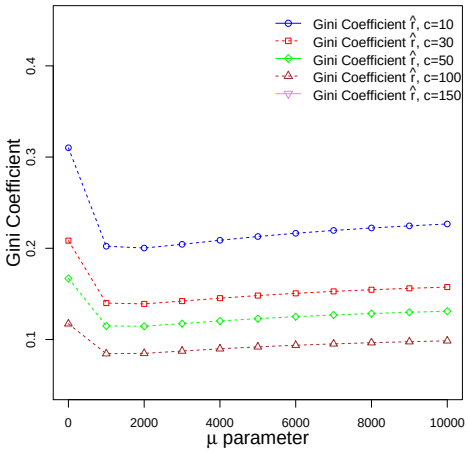
(c) Federal Register

(d) Department of Energy

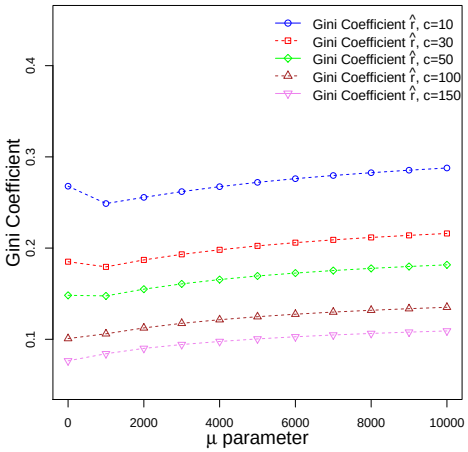
Figure A.6: Relationship between Gini-Coefficient and Precision at Recall 100 across different parameter values b with Okapi BM25. A low Gini-Coefficient usually correlates with a high Precision at Recall 100 value, both within the same collection and across different ones. For the Federal Register collection a parameter shift occurs which results in low effectiveness at the minimum Gini-Coefficient, especially with long queries.



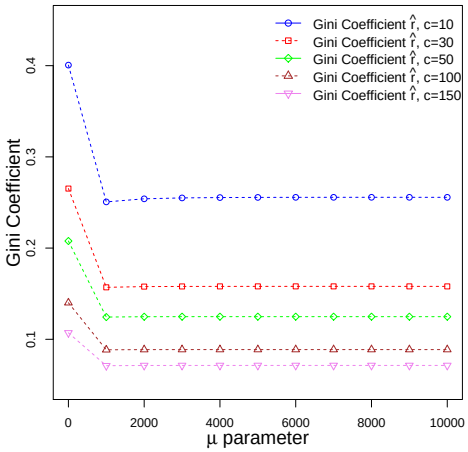
(a) Los Angeles Times



(b) Foreign Broadcast Information Service

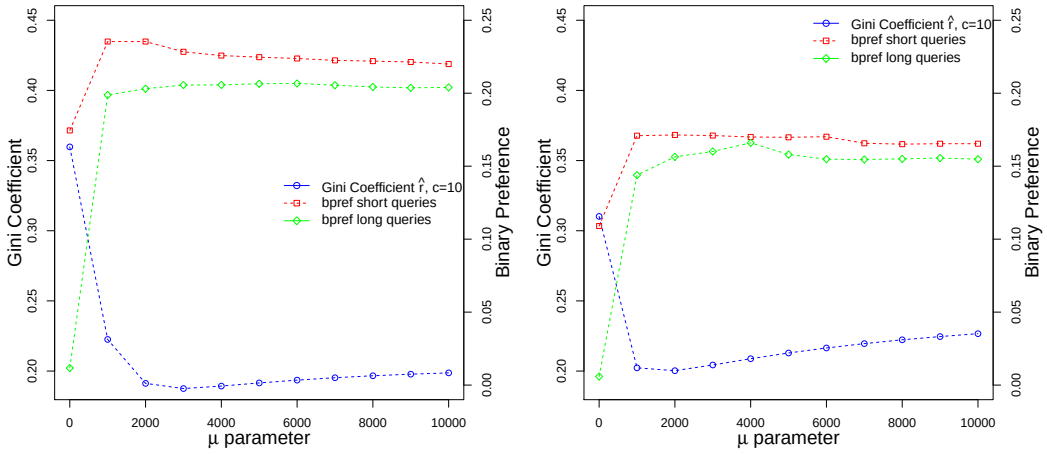


(c) Federal Register



(d) Department of Energy

Figure A.7: Gini-Coefficient at five cutoff levels across different parameter values μ with Dirichlet Smoothing. The general shapes remain the same while the parameter setting resulting in the lowest Gini-Coefficient slightly fluctuates.



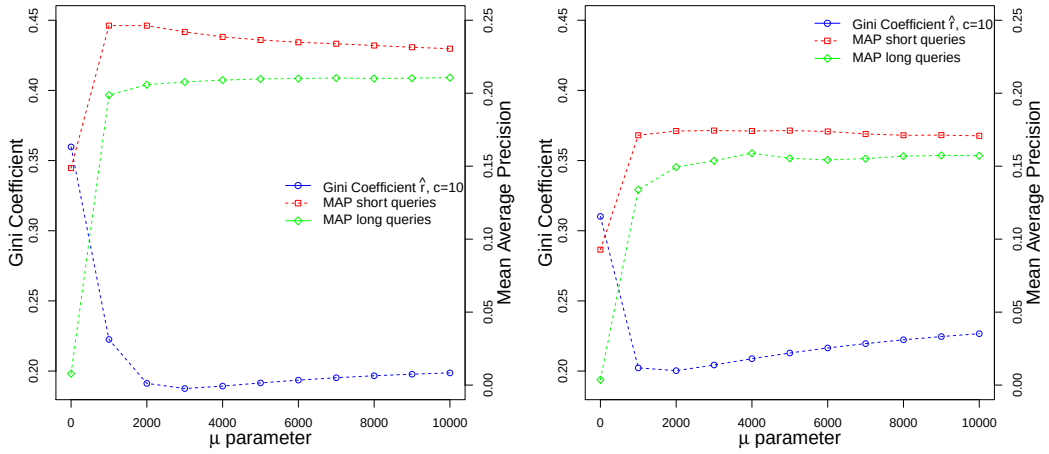
(a) Los Angeles Times

(b) Foreign Broadcast Information Service

(c) Federal Register

(d) Department of Energy

Figure A.8: Relationship between Gini-Coefficient and Binary Preference across different parameter values μ with Dirichlet Smoothing. A low Gini-Coefficient usually correlates with a high Binary Preference value, both within the same collection and across different ones.



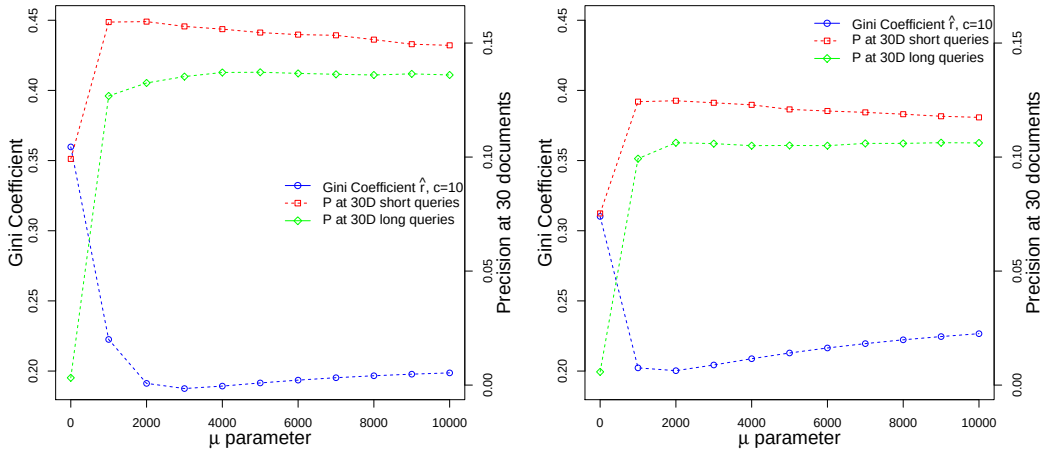
(a) Los Angeles Times

(b) Foreign Broadcast Information Service

(c) Federal Register

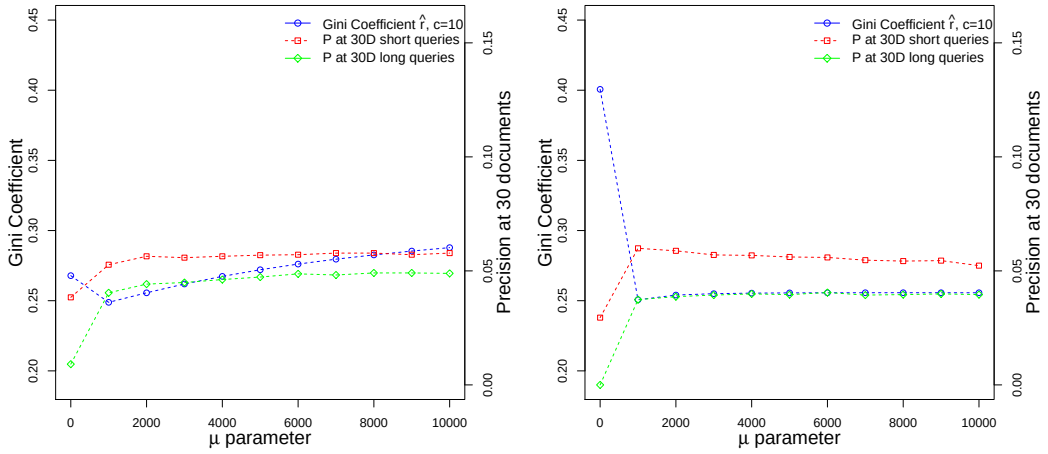
(d) Department of Energy

Figure A.9: Relationship between Gini-Coefficient and Mean Average Precision across different parameter values μ of Dirichlet Smoothing. A low Gini-Coefficient usually correlates with a high Mean Average Precision value, both within the same collection and across different ones.



(a) Los Angeles Times

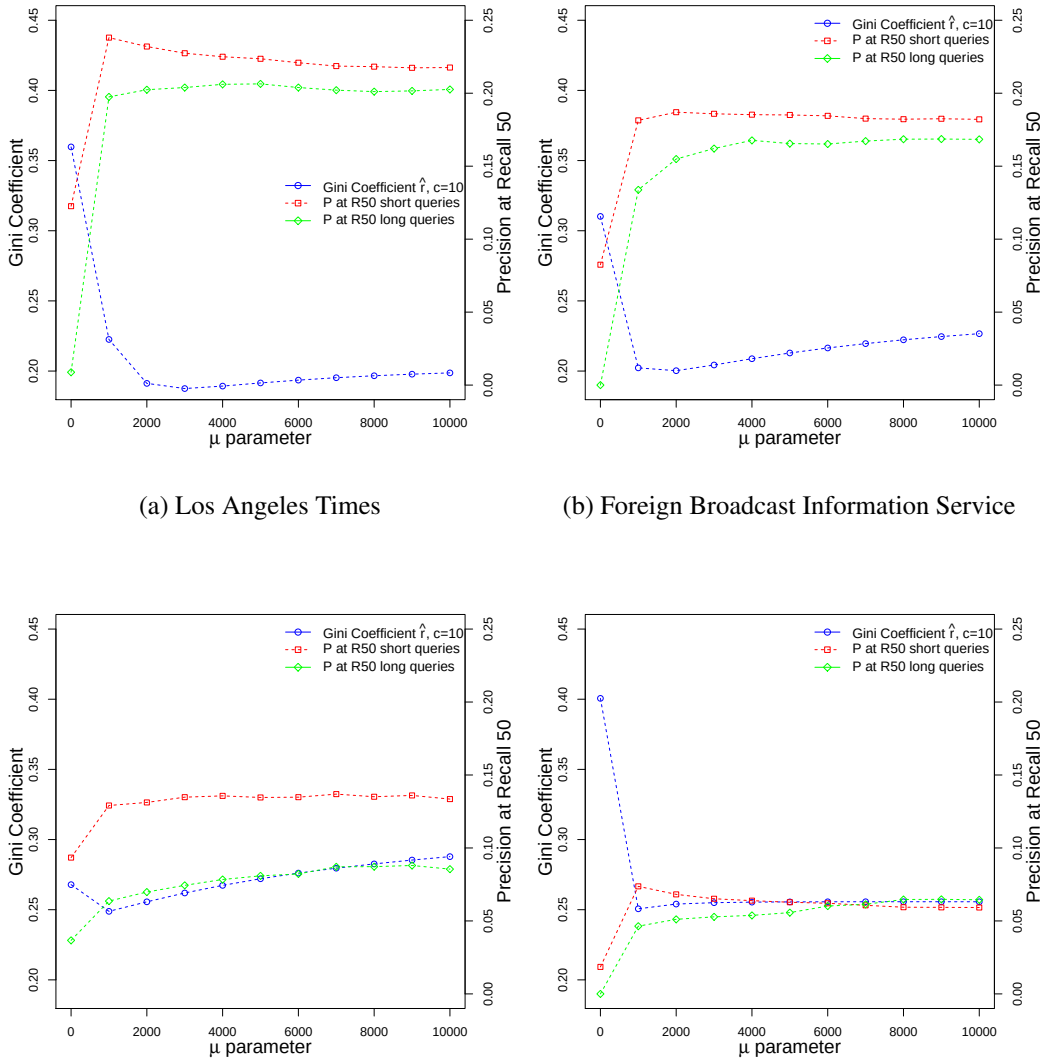
(b) Foreign Broadcast Information Service



(c) Federal Register

(d) Department of Energy

Figure A.10: Relationship between Gini-Coefficient and Precision at 30 documents across different parameter values μ of Dirichlet Smoothing. A low Gini-Coefficient usually correlates with a high Precision at 30 documents value, both within the same collection and across different ones.



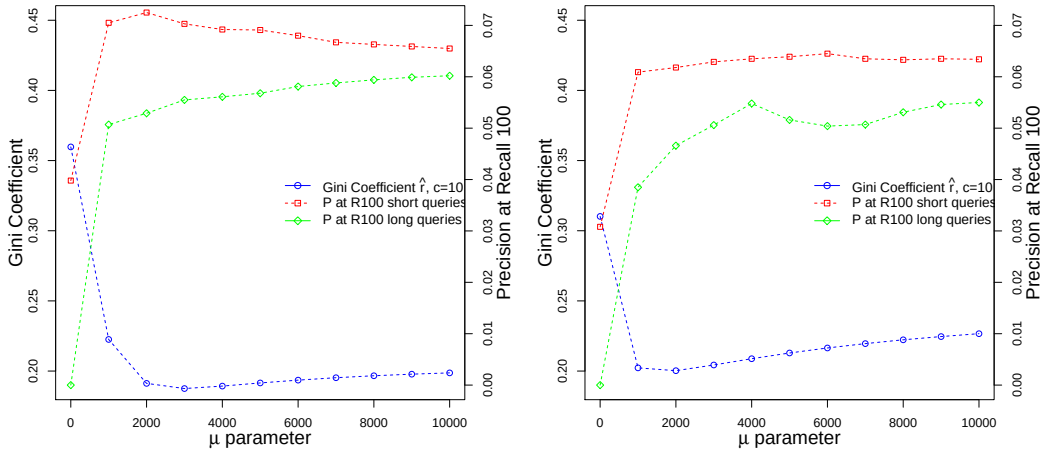
(a) Los Angeles Times

(b) Foreign Broadcast Information Service

(c) Federal Register

(d) Department of Energy

Figure A.11: Relationship between Gini-Coefficient and Precision at Recall 50 across different parameter values μ of Dirichlet Smoothing. A low Gini-Coefficient usually correlates with a high Precision at Recall 50 value, both within the same collection and across different ones.



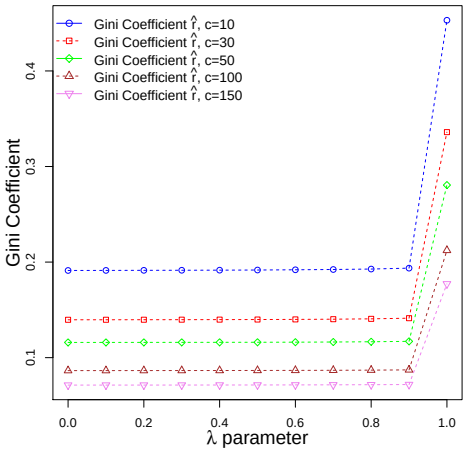
(a) Los Angeles Times

(b) Foreign Broadcast Information Service

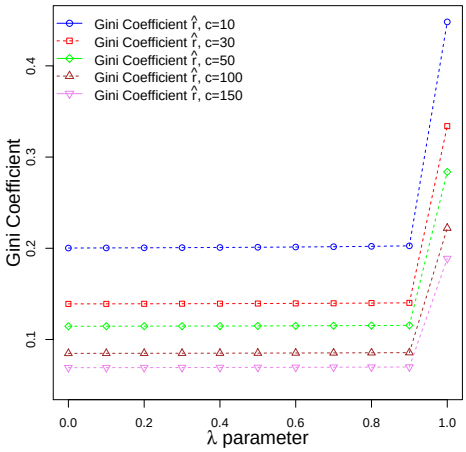
(c) Federal Register

(d) Department of Energy

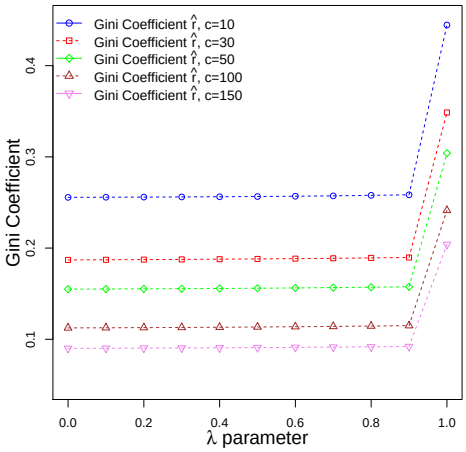
Figure A.12: Relationship between Gini-Coefficient and Precision at Recall 100 across different parameter values μ of Dirichlet Smoothing. A low Gini-Coefficient usually correlates with a high Precision at Recall 100 value, both within the same collection and across different ones.



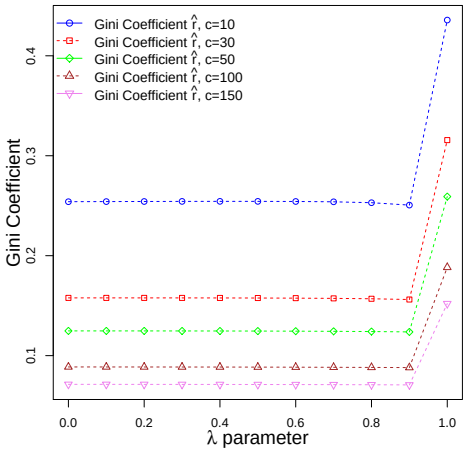
(a) Los Angeles Times



(b) Foreign Broadcast Information Service

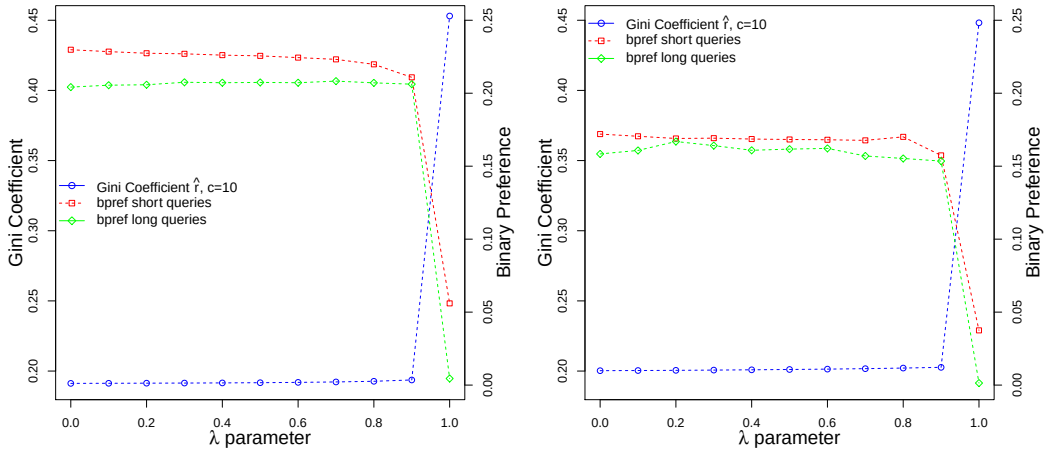


(c) Federal Register



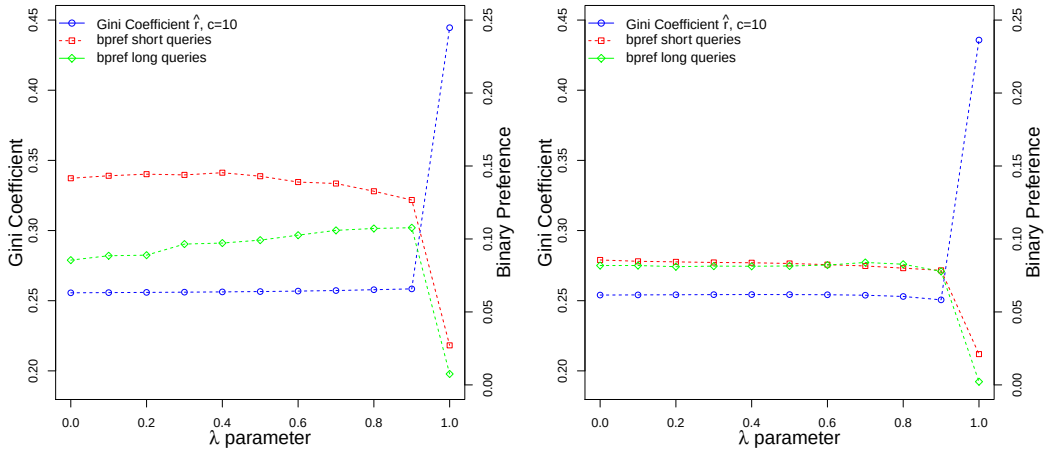
(d) Department of Energy

Figure A.13: Gini Coefficient at five cutoff levels across different parameter values λ with TwoStage Smoothing. The general shapes remain the same while the parameter setting resulting in the lowest Gini-Coefficient slightly fluctuates.



(a) Los Angeles Times

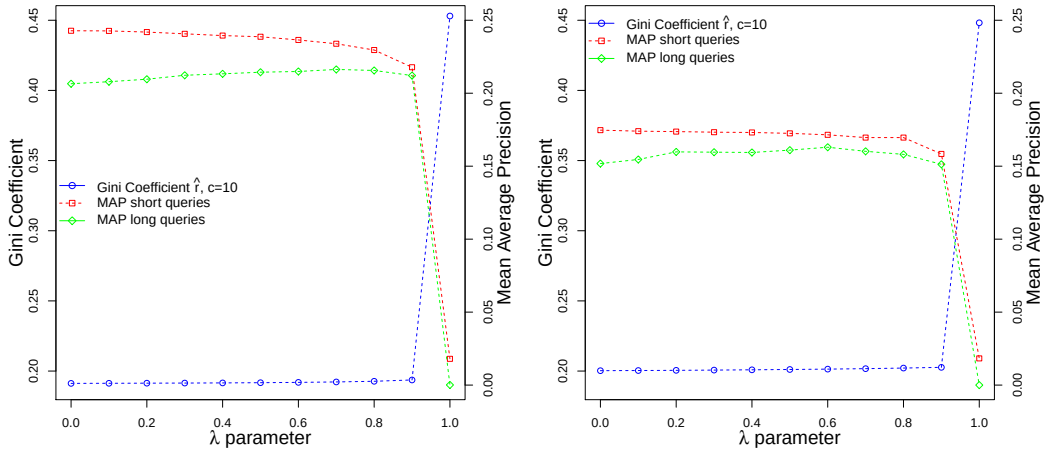
(b) Foreign Broadcast Information Service



(c) Federal Register

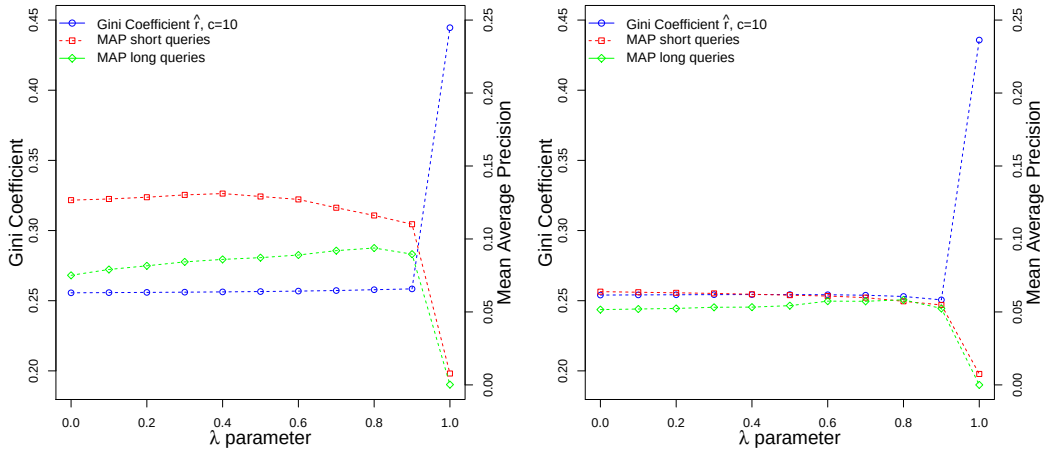
(d) Department of Energy

Figure A.14: Relationship between Gini-Coefficient and Binary Preference across different parameter values λ with TwoStage Smoothing. A low Gini-Coefficient usually correlates with a high Binary Preference value, both within the same collection and across different ones.



(a) Los Angeles Times

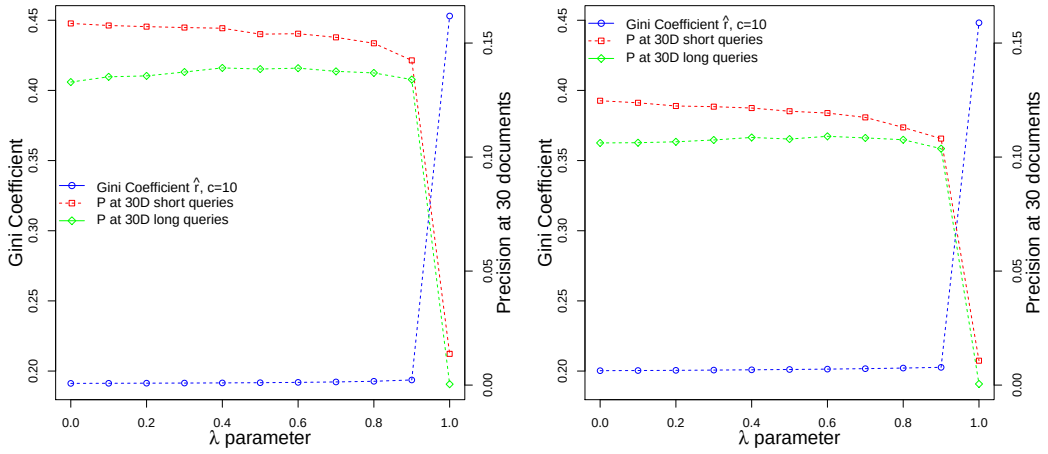
(b) Foreign Broadcast Information Service



(c) Federal Register

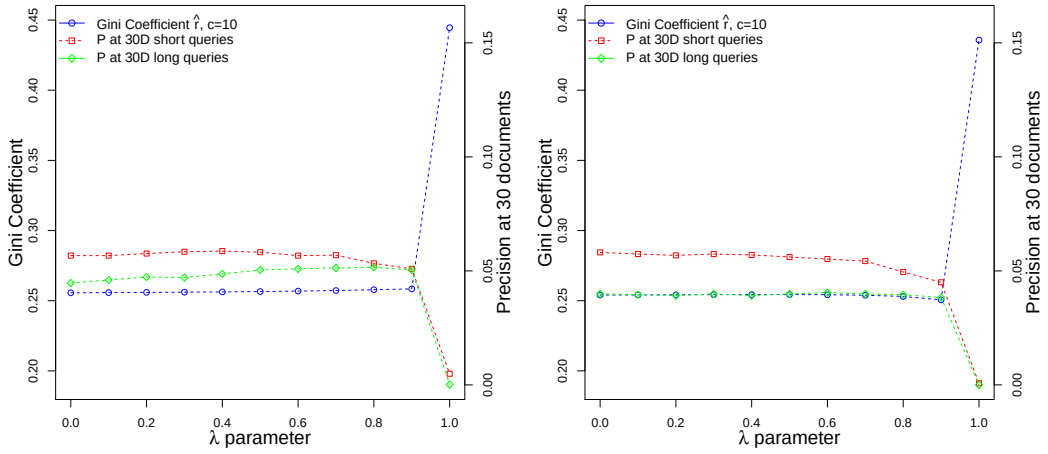
(d) Department of Energy

Figure A.15: Relationship between Gini-Coefficient and Mean Average Precision across different parameter values λ with TwoStage Smoothing. A low Gini-Coefficient usually correlates with a high Mean Average Precision value, both within the same collection and across different ones.



(a) Los Angeles Times

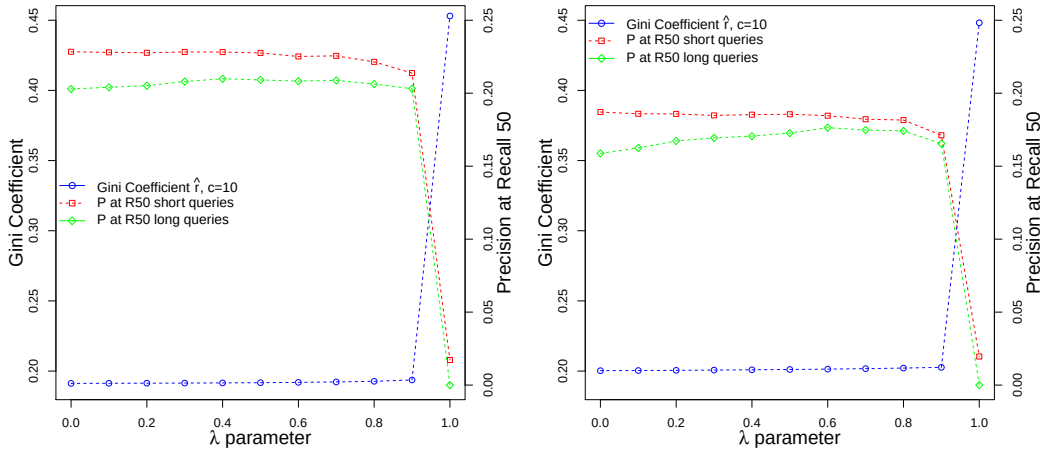
(b) Foreign Broadcast Information Service



(c) Federal Register

(d) Department of Energy

Figure A.16: Relationship between Gini-Coefficient and Precision at 30 documents across different parameter values λ with TwoStage Smoothing. A low Gini-Coefficient usually correlates with a high Precision at 30 documents value, both within the same collection and across different ones.



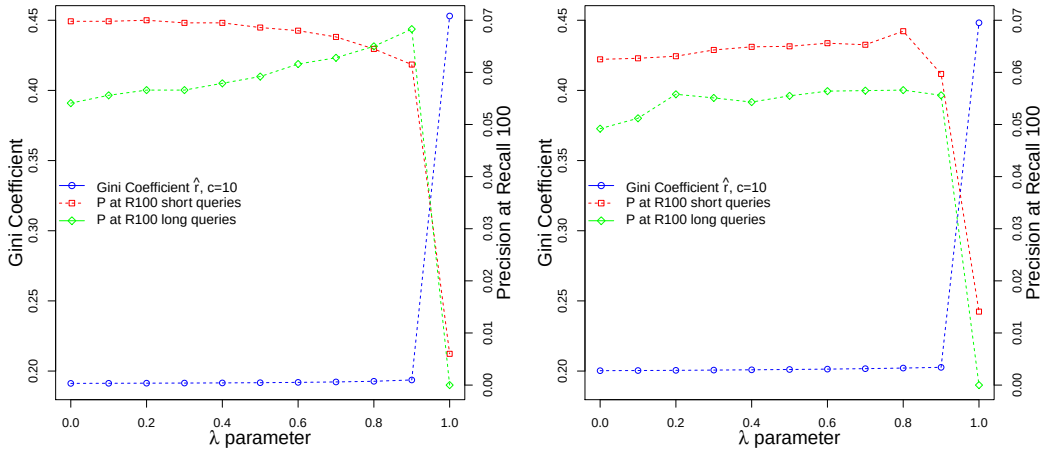
(a) Los Angeles Times

(b) Foreign Broadcast Information Service

(c) Federal Register

(d) Department of Energy

Figure A.17: Relationship between Gini-Coefficient and Precision at Recall 50 across different parameter values λ with TwoStage Smoothing. A low Gini-Coefficient usually correlates with a high Precision at Recall 50 value, both within the same collection and across different ones.



(a) Los Angeles Times

(b) Foreign Broadcast Information Service

(c) Federal Register

(d) Department of Energy

Figure A.18: Relationship between Gini-Coefficient and Precision at Recall 100 across different parameter values λ with TwoStage Smoothing. A low Gini-Coefficient usually correlates with a high Precision at Recall 100 value, both within the same collection and across different ones.

APPENDIX B

Experimental Result Tables

Table B.1: Los Angeles Times with Okapi BM25. Parameter value $b = 0.2$ results in the lowest Gini-Coefficient at $c = 10$ and is employed as the supposed optimum for comparisons.

b parameter	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Gini Coefficient c= 10	0.2530	0.1933	0.1899	0.2065	0.2325	0.2598	0.2855	0.3090	0.3299	0.3480	0.3616
Gini Coefficient c= 30	0.1846	0.1376	0.1375	0.1479	0.1630	0.1794	0.1953	0.2102	0.2233	0.2341	0.2418
Gini Coefficient c= 50	0.1585	0.1163	0.1144	0.1210	0.1319	0.1442	0.1565	0.1678	0.1777	0.1855	0.1911
Gini Coefficient c=100	0.1266	0.0903	0.0861	0.0885	0.0948	0.1025	0.1102	0.1174	0.1234	0.1280	0.1313
Gini Coefficient c=150	0.1095	0.0767	0.0717	0.0723	0.0763	0.0817	0.0873	0.0923	0.0965	0.0997	0.1018
Binary Preference (short queries)	0.2268	0.2268	0.2321	0.2334	0.2345	0.2328	0.2287	0.2234	0.2121	0.2037	0.1981
Binary Preference (long queries)	0.1542	0.1630	0.1661	0.1648	0.1641	0.1629	0.1603	0.1565	0.1513	0.1546	0.1543
Mean Average Precision (short queries)	0.2316	0.2316	0.2458	0.2483	0.2495	0.2453	0.2415	0.2349	0.2257	0.2152	0.2021
Mean Average Precision (long queries)	0.1340	0.1431	0.1402	0.1361	0.1323	0.1277	0.1225	0.1181	0.1123	0.1048	0.1005
Precision at 30 documents (short queries)	0.1525	0.1525	0.1584	0.1590	0.1589	0.1584	0.1574	0.1544	0.1510	0.1466	0.1404
Precision at 30 documents (long queries)	0.0881	0.0904	0.0898	0.0882	0.0850	0.0830	0.0811	0.0775	0.0744	0.0723	0.0693
Precision at Recall 50 (short queries)	0.2142	0.2142	0.2297	0.2312	0.2343	0.2340	0.2335	0.2260	0.2197	0.2122	0.2004
Precision at Recall 50 (long queries)	0.1174	0.1261	0.1240	0.1230	0.1198	0.1187	0.1166	0.1125	0.1033	0.0910	0.0860
Precision at Recall 100 (short queries)	0.0632	0.0632	0.0684	0.0690	0.0681	0.0659	0.0654	0.0659	0.0624	0.0573	0.0562
Precision at Recall 100 (long queries)	0.0275	0.0311	0.0322	0.0332	0.0319	0.0312	0.0283	0.0262	0.0243	0.0232	0.0211

Table B.2: Los Angeles Times with Dirichlet Smoothing. The parameter value $\mu = 3000$ results in the lowest Gini-Coefficient at $c = 10$ and is employed as the supposed optimum for comparisons.

μ parameter	0	1000	2000	3000	4000	5000	6000	7000	8000	9000	10000
Gini Coefficient c= 10	0.3597	0.2225	0.1912	0.1875	0.1893	0.1915	0.1935	0.1953	0.1967	0.1978	0.1987
Gini Coefficient c= 30	0.2407	0.1589	0.1396	0.1358	0.1357	0.1363	0.1371	0.1377	0.1383	0.1388	0.1392
Gini Coefficient c= 50	0.1903	0.1296	0.1159	0.1137	0.1140	0.1146	0.1153	0.1159	0.1164	0.1168	0.1172
Gini Coefficient c=100	0.1308	0.0941	0.0864	0.0859	0.0868	0.0877	0.0885	0.0891	0.0896	0.0901	0.0904
Gini Coefficient c=150	0.1014	0.0762	0.0713	0.0715	0.0726	0.0735	0.0743	0.0749	0.0755	0.0759	0.0762
Binary Preference (short queries)	0.1745	0.2354	0.2354	0.2284	0.2258	0.2247	0.2238	0.2225	0.2220	0.2214	0.2200
Binary Preference (long queries)	0.0117	0.1988	0.2031	0.2056	0.2057	0.2065	0.2067	0.2055	0.2043	0.2037	0.2040
Mean Average Precision (short queries)	0.1487	0.2463	0.2463	0.2420	0.2386	0.2365	0.2349	0.2339	0.2327	0.2315	0.2305
Mean Average Precision (long queries)	0.0079	0.1987	0.2059	0.2077	0.2090	0.2098	0.2101	0.2104	0.2101	0.2103	0.2107
Precision at 30 documents (short queries)	0.0992	0.1592	0.1594	0.1573	0.1561	0.1546	0.1537	0.1534	0.1515	0.1495	0.1490
Precision at 30 documents (long queries)	0.0032	0.1268	0.1325	0.1353	0.1371	0.1372	0.1367	0.1363	0.1360	0.1365	0.1360
Precision at Recall 50 (short queries)	0.1226	0.2381	0.2320	0.2274	0.2250	0.2236	0.2209	0.2186	0.2182	0.2174	0.2176
Precision at Recall 50 (long queries)	0.0088	0.1975	0.2024	0.2039	0.2061	0.2064	0.2039	0.2021	0.2011	0.2016	0.2026
Precision at Recall 100 (short queries)	0.0398	0.0705	0.0725	0.0703	0.0692	0.0691	0.0680	0.0667	0.0663	0.0659	0.0655
Precision at Recall 100 (long queries)	0.0000	0.0507	0.0529	0.0555	0.0561	0.0568	0.0581	0.0588	0.0594	0.0599	0.0602

Table B.3: Los Angeles Times with TwoStage Smoothing. The parameter value $\lambda = 0.0$ results in the lowest Gini-Coefficient at $c = 10$ and is employed as the supposed optimum for comparisons.

λ parameter	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Gini Coefficient c= 10	0.1912	0.1913	0.1914	0.1915	0.1916	0.1917	0.1919	0.1922	0.1927	0.1937	0.4530
Gini Coefficient c= 30	0.1396	0.1396	0.1397	0.1397	0.1398	0.1399	0.1400	0.1402	0.1406	0.1413	0.3360
Gini Coefficient c= 50	0.1159	0.1160	0.1160	0.1160	0.1160	0.1161	0.1162	0.1163	0.1166	0.1170	0.2806
Gini Coefficient c=100	0.0864	0.0865	0.0865	0.0865	0.0865	0.0866	0.0867	0.0868	0.0869	0.0872	0.2122
Gini Coefficient c=150	0.0713	0.0713	0.0713	0.0713	0.0713	0.0714	0.0715	0.0716	0.0717	0.0719	0.1772
Binary Preference (short queries)	0.2298	0.2285	0.2274	0.2270	0.2261	0.2256	0.2244	0.2232	0.2198	0.2109	0.0560
Binary Preference (long queries)	0.2042	0.2055	0.2058	0.2075	0.2072	0.2074	0.2072	0.2083	0.2071	0.2062	0.0045
Mean Average Precision (short queries)	0.2428	0.2427	0.2419	0.2407	0.2395	0.2387	0.2365	0.2339	0.2296	0.2178	0.0180
Mean Average Precision (long queries)	0.2065	0.2079	0.2096	0.2123	0.2133	0.2144	0.2149	0.2163	0.2156	0.2121	0.0001
Precision at 30 documents (short queries)	0.1586	0.1577	0.1572	0.1568	0.1565	0.1539	0.1541	0.1525	0.1499	0.1424	0.0137
Precision at 30 documents (long queries)	0.1329	0.1352	0.1356	0.1373	0.1391	0.1386	0.1390	0.1376	0.1369	0.1340	0.0004
Precision at Recall 50 (short queries)	0.2284	0.2280	0.2277	0.2283	0.2282	0.2276	0.2252	0.2256	0.2215	0.2138	0.0172
Precision at Recall 50 (long queries)	0.2029	0.2041	0.2052	0.2080	0.2099	0.2091	0.2083	0.2088	0.2063	0.2031	0.0000
Precision at Recall 100 (short queries)	0.0698	0.0698	0.0700	0.0695	0.0695	0.0686	0.0680	0.0668	0.0645	0.0615	0.0060
Precision at Recall 100 (long queries)	0.0541	0.0556	0.0566	0.0566	0.0579	0.0592	0.0616	0.0628	0.0650	0.0683	0.0000

Table B.4: Foreign Broadcast Information Service with Okapi BM25. The parameter value $b = 0.3$ results in the lowest Gini-Coefficient at $c = 10$ and is employed as the supposed optimum for comparisons.

b parameter	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Gini Coefficient c= 10	0.2990	0.2163	0.2026	0.2016	0.2073	0.2180	0.2327	0.2509	0.2717	0.2936	0.3133
Gini Coefficient c= 30	0.2112	0.1519	0.1408	0.1390	0.1425	0.1496	0.1596	0.1718	0.1853	0.1989	0.2106
Gini Coefficient c= 50	0.1779	0.1272	0.1164	0.1142	0.1166	0.1221	0.1300	0.1396	0.1500	0.1602	0.1688
Gini Coefficient c=100	0.1384	0.0971	0.0870	0.0842	0.0853	0.0889	0.0942	0.1005	0.1071	0.1133	0.1185
Gini Coefficient c=150	0.1176	0.0812	0.0715	0.0686	0.0690	0.0716	0.0755	0.0801	0.0848	0.0890	0.0927
Binary Preference (short queries)	0.1455	0.1720	0.1646	0.1648	0.1649	0.1602	0.1532	0.1517	0.1470	0.1468	0.1340
Binary Preference (long queries)	0.0502	0.0918	0.0995	0.1050	0.1014	0.1001	0.1002	0.1040	0.1069	0.1038	0.1048
Mean Average Precision (short queries)	0.1476	0.1746	0.1678	0.1646	0.1618	0.1556	0.1483	0.1448	0.1405	0.1388	0.1289
Mean Average Precision (long queries)	0.0389	0.0591	0.0614	0.0614	0.0589	0.0555	0.0542	0.0529	0.0529	0.0518	0.0499
Precision at 30 documents (short queries)	0.1104	0.1215	0.1228	0.1215	0.1209	0.1215	0.1185	0.1171	0.1143	0.1108	0.1059
Precision at 30 documents (long queries)	0.0352	0.0364	0.0364	0.0359	0.0356	0.0353	0.0348	0.0343	0.0321	0.0311	0.0309
Precision at Recall 50 (short queries)	0.1574	0.1862	0.1800	0.1774	0.1725	0.1635	0.1540	0.1470	0.1391	0.1376	0.1270
Precision at Recall 50 (long queries)	0.0344	0.0582	0.0624	0.0644	0.0633	0.0602	0.0571	0.0561	0.0544	0.0521	0.0485
Precision at Recall 100 (short queries)	0.0539	0.0637	0.0571	0.0554	0.0540	0.0519	0.0467	0.0454	0.0447	0.0456	0.0407
Precision at Recall 100 (long queries)	0.0112	0.0198	0.0209	0.0230	0.0211	0.0185	0.0189	0.0181	0.0190	0.0183	0.0175

Table B.5: Foreign Broadcast Information Service with Dirichlet Smoothing. The parameter value $\mu = 2000$ results in the lowest Gini-Coefficient at $c = 10$ and is employed as the supposed optimum for comparisons.

μ parameter	0	1000	2000	3000	4000	5000	6000	7000	8000	9000	10000
Gini Coefficient c= 10	0.3102	0.2023	0.2003	0.2043	0.2088	0.2129	0.2165	0.2196	0.2223	0.2246	0.2266
Gini Coefficient c= 30	0.2084	0.1399	0.1390	0.1421	0.1453	0.1482	0.1506	0.1527	0.1545	0.1561	0.1575
Gini Coefficient c= 50	0.1670	0.1148	0.1145	0.1174	0.1203	0.1229	0.1250	0.1269	0.1284	0.1298	0.1310
Gini Coefficient c=100	0.1172	0.0845	0.0848	0.0873	0.0898	0.0919	0.0937	0.0952	0.0965	0.0976	0.0985
Gini Coefficient c=150	0.0916	0.0685	0.0691	0.0714	0.0736	0.0754	0.0770	0.07830	0.0794	0.0804	0.0811
Binary Preference (short queries)	0.1090	0.1709	0.1714	0.1710	0.1700	0.1698	0.1702	0.1657	0.1651	0.1654	0.1654
Binary Preference (long queries)	0.0059	0.1439	0.1564	0.1601	0.1660	0.1580	0.1548	0.1546	0.1550	0.1556	0.1548
Mean Average Precision (short queries)	0.0928	0.1712	0.1741	0.1744	0.1741	0.1744	0.1738	0.1721	0.1712	0.1713	0.1708
Mean Average Precision (long queries)	0.0037	0.1339	0.1494	0.1537	0.1589	0.1554	0.1543	0.1552	0.1569	0.1574	0.1572
Precision at 30 documents (short queries)	0.0753	0.1243	0.1247	0.1238	0.1229	0.1209	0.1202	0.1196	0.1188	0.1179	0.1174
Precision at 30 documents (long queries)	0.0058	0.0993	0.1063	0.1059	0.1050	0.1051	0.1050	0.1060	0.1060	0.1063	0.1062
Precision at Recall 50 (short queries)	0.0825	0.1814	0.1870	0.1859	0.1853	0.1851	0.1845	0.1826	0.1822	0.1825	0.1821
Precision at Recall 50 (long queries)	0.0000	0.1338	0.1549	0.1621	0.1677	0.1655	0.1652	0.1672	0.1685	0.1686	0.1684
Precision at Recall 100 (short queries)	0.0308	0.0609	0.0618	0.0629	0.0635	0.0639	0.0645	0.0635	0.0633	0.0635	0.0634
Precision at Recall 100 (long queries)	0.0000	0.0385	0.0466	0.0506	0.0548	0.0516	0.0504	0.0507	0.0531	0.0546	0.0550

Table B.6: Foreign Broadcast Information Service with TwoStage Smoothing. The parameter value $\lambda = 0.0$ results in the lowest Gini-Coefficient at $c = 10$ and is employed as the supposed optimum for comparisons.

λ parameter	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Gini Coefficient c= 10	0.2003	0.2004	0.2005	0.2007	0.2009	0.2011	0.2014	0.2017	0.2021	0.2027	0.4482
Gini Coefficient c= 30	0.1390	0.1391	0.1391	0.1392	0.1393	0.1394	0.1396	0.1397	0.1399	0.1402	0.3339
Gini Coefficient c= 50	0.1145	0.1146	0.1146	0.1147	0.1148	0.1149	0.1150	0.1151	0.1153	0.1155	0.2838
Gini Coefficient c=100	0.0848	0.0848	0.0849	0.0849	0.0850	0.0851	0.0851	0.0853	0.0854	0.0856	0.2221
Gini Coefficient c=150	0.0691	0.0691	0.0692	0.0692	0.0693	0.0694	0.0695	0.0696	0.0697	0.0699	0.1888
Binary Preference (short queries)	0.1720	0.1705	0.1690	0.1692	0.1686	0.1683	0.1681	0.1677	0.1701	0.1574	0.0375
Binary Preference (long queries)	0.1584	0.1608	0.1669	0.1641	0.1609	0.1617	0.1622	0.1570	0.1553	0.1535	0.0014
Mean Average Precision (short queries)	0.1747	0.1740	0.1737	0.1733	0.1731	0.1725	0.1715	0.1696	0.1696	0.1584	0.0183
Mean Average Precision (long queries)	0.1518	0.1546	0.1598	0.1596	0.1594	0.1610	0.1630	0.1602	0.1581	0.1513	0.0000
Precision at 30 documents (short queries)	0.1247	0.1238	0.1224	0.1221	0.1215	0.1201	0.1193	0.1174	0.1130	0.1081	0.0107
Precision at 30 documents (long queries)	0.1062	0.1063	0.1067	0.1075	0.1086	0.1079	0.1091	0.1084	0.1076	0.1037	0.0005
Precision at Recall 50 (short queries)	0.1871	0.1859	0.1858	0.1848	0.1853	0.1856	0.1846	0.1822	0.1816	0.1712	0.0196
Precision at Recall 50 (long queries)	0.1588	0.1625	0.1673	0.1694	0.1706	0.1727	0.1764	0.1748	0.1742	0.1655	0.0000
Precision at Recall 100 (short queries)	0.0625	0.0627	0.0631	0.0643	0.0649	0.0650	0.0656	0.0653	0.0679	0.0597	0.0141
Precision at Recall 100 (long queries)	0.0492	0.0512	0.0558	0.0551	0.0543	0.0555	0.0564	0.0565	0.0566	0.0556	0.0000

Table B.7: Federal Register with Okapi BM25. The parameter value $b = 0.7$ results in the lowest Gini-Coefficient at $c = 10$ and is employed as the supposed optimum for comparisons.

b parameter	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Gini Coefficient c= 10	0.3803	0.3044	0.2811	0.2676	0.2588	0.2530	0.2493	0.2473	0.2480	0.2549	0.2711
Gini Coefficient c= 30	0.2898	0.2336	0.2130	0.1998	0.1911	0.1847	0.1800	0.1770	0.1762	0.1795	0.1872
Gini Coefficient c= 50	0.2498	0.1999	0.1805	0.1681	0.1595	0.1530	0.1484	0.1451	0.1437	0.1454	0.1499
Gini Coefficient c=100	0.1968	0.1540	0.1369	0.1254	0.1174	0.1115	0.1070	0.1036	0.1017	0.1015	0.1018
Gini Coefficient c=150	0.1664	0.1285	0.1128	0.1024	0.0949	0.0894	0.08528	0.0819	0.0798	0.07865	0.0770
Binary Preference (short queries)	0.1159	0.1317	0.1331	0.1328	0.1351	0.1300	0.1286	0.1244	0.1204	0.1171	0.1142
Binary Preference (long queries)	0.0815	0.0955	0.0890	0.0871	0.0854	0.0829	0.0798	0.0805	0.0810	0.0814	0.0725
Mean Average Precision (short queries)	0.1202	0.1252	0.1248	0.1182	0.1193	0.1147	0.1116	0.1073	0.1012	0.0913	0.0834
Mean Average Precision (long queries)	0.0698	0.0773	0.0734	0.0626	0.0585	0.0490	0.0422	0.0405	0.0347	0.0313	0.0295
Precision at 30 documents (short queries)	0.0556	0.0622	0.0611	0.0613	0.0598	0.0562	0.0536	0.0538	0.0522	0.0496	0.0460
Precision at 30 documents (long queries)	0.0309	0.0338	0.0349	0.0329	0.0311	0.0293	0.0280	0.0264	0.0258	0.0240	0.0238
Precision at Recall 50 (short queries)	0.1264	0.1225	0.1198	0.1156	0.1170	0.1130	0.1089	0.1044	0.0968	0.0838	0.0765
Precision at Recall 50 (long queries)	0.0772	0.0767	0.0754	0.0608	0.0590	0.0520	0.0434	0.0381	0.0265	0.0255	0.0252
Precision at Recall 100 (short queries)	0.0584	0.0592	0.0598	0.0578	0.0580	0.0543	0.0520	0.0495	0.0450	0.0385	0.0352
Precision at Recall 100 (long queries)	0.0300	0.0302	0.0262	0.0211	0.0193	0.0106	0.0055	0.0052	0.0036	0.0032	0.0022

Table B.8: Federal Register with Dirichlet Smoothing. The parameter value $\mu = 1000$ results in the lowest Gini-Coefficient at $c = 10$ and is employed as the supposed optimum for comparisons.

μ parameter	0	1000	2000	3000	4000	5000	6000	7000	8000	9000	10000
Gini Coefficient c= 10	0.2679	0.2488	0.2556	0.2619	0.2673	0.2721	0.2761	0.2796	0.2826	0.2854	0.2878
Gini Coefficient c= 30	0.1851	0.1794	0.1870	0.1931	0.1981	0.2024	0.2060	0.2090	0.2117	0.2139	0.2160
Gini Coefficient c= 50	0.1482	0.1476	0.1550	0.1608	0.1655	0.1694	0.1726	0.1754	0.1778	0.1798	0.1817
Gini Coefficient c=100	0.1008	0.1060	0.1125	0.1175	0.1215	0.1248	0.1275	0.1299	0.1318	0.1336	0.1351
Gini Coefficient c=150	0.0764	0.0842	0.0900	0.0943	0.0976	0.1004	0.1028	0.1047	0.1064	0.1079	0.1092
Binary Preference (short queries)	0.1033	0.1336	0.1395	0.1441	0.1448	0.1450	0.1460	0.1449	0.1449	0.1456	0.1451
Binary Preference (long queries)	0.0369	0.0821	0.0839	0.0870	0.0890	0.0896	0.0978	0.0987	0.0992	0.1013	0.0991
Mean Average Precision (short queries)	0.0781	0.1193	0.1246	0.1278	0.1300	0.1313	0.1320	0.1332	0.1296	0.1301	0.1288
Mean Average Precision (long queries)	0.0328	0.0693	0.0738	0.0760	0.0791	0.0811	0.0820	0.0842	0.0837	0.0858	0.0855
Precision at 30 documents (short queries)	0.0384	0.0527	0.0564	0.0558	0.0564	0.0569	0.0571	0.0578	0.0578	0.0571	0.0578
Precision at 30 documents (long queries)	0.0091	0.0404	0.0442	0.0449	0.0462	0.0473	0.0487	0.0482	0.0491	0.0491	0.0489
Precision at Recall 50 (short queries)	0.0934	0.1291	0.1312	0.1348	0.1357	0.1346	0.1348	0.1369	0.1351	0.1360	0.1335
Precision at Recall 50 (long queries)	0.0367	0.0636	0.0698	0.0744	0.0784	0.0809	0.0822	0.0871	0.0872	0.0880	0.0855
Precision at Recall 100 (short queries)	0.0238	0.0570	0.0590	0.0602	0.0608	0.0612	0.0615	0.0617	0.0579	0.0583	0.0572
Precision at Recall 100 (long queries)	0.0100	0.0218	0.0227	0.0232	0.0240	0.0250	0.0252	0.0258	0.0251	0.0259	0.0260

Table B.9: Federal Register with TwoStage Smoothing. The parameter value $\lambda = 0.0$ results in the lowest Gini-Coefficient at $c = 10$ and is employed as the supposed optimum for comparisons.

λ parameter	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Gini Coefficient c= 10	0.2556	0.2558	0.2559	0.2561	0.2563	0.2565	0.2569	0.2572	0.2578	0.2584	0.4446
Gini Coefficient c= 30	0.1870	0.1872	0.1874	0.1876	0.1878	0.1881	0.1884	0.1888	0.1892	0.1897	0.3487
Gini Coefficient c= 50	0.1550	0.1551	0.1553	0.1555	0.1557	0.1560	0.1563	0.1567	0.1571	0.1576	0.3040
Gini Coefficient c=100	0.1125	0.1126	0.1128	0.1130	0.1132	0.1135	0.1138	0.1141	0.1145	0.1149	0.2413
Gini Coefficient c=150	0.0900	0.0902	0.0903	0.0905	0.0907	0.0909	0.0911	0.0914	0.0917	0.0921	0.2040
Binary Preference (short queries)	0.1416	0.1433	0.1444	0.1439	0.1454	0.1430	0.1390	0.1380	0.1327	0.1267	0.0271
Binary Preference (long queries)	0.0855	0.0885	0.0889	0.0965	0.0972	0.0992	0.1026	0.1059	0.1072	0.1077	0.0075
Mean Average Precision (short queries)	0.1266	0.1274	0.1286	0.1302	0.1311	0.1291	0.1271	0.1214	0.1161	0.1101	0.0078
Mean Average Precision (long queries)	0.0751	0.0791	0.0816	0.0843	0.0860	0.0872	0.0890	0.0920	0.0938	0.0896	0.0002
Precision at 30 documents (short queries)	0.0567	0.0567	0.0576	0.0584	0.0587	0.0582	0.0567	0.0569	0.0533	0.0509	0.0049
Precision at 30 documents (long queries)	0.0447	0.0460	0.0473	0.0471	0.0487	0.0504	0.0509	0.0513	0.0516	0.0504	0.0002
Precision at Recall 50 (short queries)	0.1337	0.1329	0.1339	0.1337	0.1336	0.1288	0.1249	0.1219	0.1178	0.1134	0.0079
Precision at Recall 50 (long queries)	0.0725	0.0769	0.0809	0.0833	0.0858	0.0885	0.0929	0.0980	0.0985	0.0897	0.0000
Precision at Recall 100 (short queries)	0.0591	0.0600	0.0612	0.0633	0.0635	0.0637	0.0638	0.0608	0.0523	0.0483	0.0031
Precision at Recall 100 (long queries)	0.0232	0.0242	0.0249	0.0270	0.0283	0.0284	0.0286	0.0293	0.0306	0.0310	0.0000

Table B.10: Department of Energy with Okapi BM25. The parameter value $b = 0.3$ results in the lowest Gini-Coefficient at $c = 10$ and is employed as the supposed optimum for comparisons.

b parameter	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Gini Coefficient c= 10	0.3054	0.2522	0.2448	0.2386	0.2424	0.2596	0.2872	0.3193	0.3506	0.3777	0.3992
Gini Coefficient c= 30	0.2007	0.1573	0.1553	0.1564	0.1652	0.1803	0.1989	0.2183	0.2364	0.2517	0.2643
Gini Coefficient c= 50	0.1626	0.1245	0.1236	0.1255	0.1330	0.1449	0.1589	0.1732	0.1865	0.1977	0.2071
Gini Coefficient c=100	0.1195	0.0886	0.0880	0.0892	0.0938	0.1011	0.1097	0.1185	0.1266	0.1337	0.1396
Gini Coefficient c=150	0.0982	0.0712	0.0706	0.0711	0.0739	0.0789	0.0850	0.0913	0.0972	0.1024	0.1068
Binary Preference (short queries)	0.0881	0.0833	0.0812	0.0808	0.0799	0.0793	0.0782	0.0768	0.0761	0.0735	0.0726
Binary Preference (long queries)	0.0623	0.0619	0.0620	0.0612	0.0605	0.0535	0.0540	0.0536	0.0530	0.0492	0.0490
Mean Average Precision (short queries)	0.0635	0.0614	0.0607	0.0597	0.0591	0.0585	0.0529	0.0515	0.0503	0.0491	0.0464
Mean Average Precision (long queries)	0.0276	0.0276	0.0277	0.0272	0.0264	0.0254	0.0249	0.0241	0.0221	0.0209	0.0194
Precision at 30 documents (short queries)	0.0547	0.0554	0.0552	0.0552	0.0550	0.0543	0.0532	0.0532	0.0516	0.0509	0.0489
Precision at 30 documents (long queries)	0.0268	0.0259	0.0259	0.0266	0.0266	0.0268	0.0264	0.0268	0.0264	0.0257	0.0255
Precision at Recall 50 (short queries)	0.0650	0.0617	0.0608	0.0591	0.0589	0.0581	0.0528	0.0523	0.0515	0.0499	0.0477
Precision at Recall 50 (long queries)	0.0229	0.0237	0.0242	0.0234	0.0230	0.0219	0.0215	0.0206	0.0167	0.0157	0.0128
Precision at Recall 100 (short queries)	0.0185	0.0169	0.0168	0.0166	0.0166	0.0164	0.0126	0.0125	0.0115	0.0108	0.0096
Precision at Recall 100 (long queries)	0.0030	0.0026	0.0020	0.0020	0.0014	0.0014	0.0014	0.0013	0.0013	0.0012	0.0011

Table B.11: Department of Energy with Dirichlet Smoothing. The parameter value $\mu = 1000$ results in the lowest Gini-Coefficient at $c = 10$ and is employed as the supposed optimum for comparisons.

μ parameter	0	1000	2000	3000	4000	5000	6000	7000	8000	9000	10000
Gini Coefficient c= 10	0.4006	0.2506	0.2540	0.2550	0.2554	0.2556	0.2556	0.2557	0.2557	0.2557	0.2557
Gini Coefficient c= 30	0.2652	0.1570	0.1578	0.1579	0.1580	0.1580	0.1580	0.1580	0.1580	0.1580	0.1580
Gini Coefficient c= 50	0.2077	0.1244	0.1248	0.1248	0.1248	0.1248	0.1248	0.1248	0.1248	0.1248	0.1248
Gini Coefficient c=100	0.1400	0.0885	0.0887	0.0887	0.0887	0.0887	0.0887	0.0887	0.0887	0.0887	0.0887
Gini Coefficient c=150	0.1071	0.0711	0.0713	0.0713	0.0713	0.0713	0.0713	0.0714	0.0714	0.0713	0.0713
Binary Preference (short queries)	0.0361	0.0862	0.0860	0.0844	0.0840	0.0833	0.0829	0.0821	0.0817	0.0816	0.0811
Binary Preference (long queries)	0.0022	0.0791	0.0817	0.0817	0.0815	0.0816	0.0827	0.0822	0.0841	0.0837	0.0838
Mean Average Precision (short queries)	0.0260	0.0688	0.0656	0.0632	0.0624	0.0616	0.0612	0.0606	0.0597	0.0591	0.0589
Mean Average Precision (long queries)	0.0000	0.0465	0.0506	0.0522	0.0529	0.0542	0.0566	0.0572	0.0582	0.0583	0.0582
Precision at 30 documents (short queries)	0.0295	0.0599	0.0588	0.0570	0.0568	0.0561	0.0559	0.0547	0.0543	0.0545	0.0523
Precision at 30 documents (long queries)	0.0000	0.0374	0.0387	0.0394	0.0399	0.0396	0.0405	0.0394	0.0396	0.0399	0.0396
Precision at Recall 50 (short queries)	0.0185	0.0737	0.0682	0.0652	0.0639	0.0628	0.0621	0.0607	0.0594	0.0593	0.0592
Precision at Recall 50 (long queries)	0.0000	0.0464	0.0511	0.0528	0.0538	0.0557	0.0602	0.0614	0.0647	0.0648	0.0645
Precision at Recall 100 (short queries)	0.0084	0.0209	0.0209	0.0209	0.0209	0.0209	0.0209	0.0209	0.0208	0.0209	0.0209
Precision at Recall 100 (long queries)	0.0000	0.0166	0.0185	0.0197	0.0205	0.0214	0.0215	0.0215	0.0216	0.0216	0.0216

Table B.12: Department of Energy with TwoStage Smoothing. The parameter value $\lambda = 0.9$ results in the lowest Gini-Coefficient at $c = 10$ and is employed as the supposed optimum for comparisons.

λ parameter	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Gini Coefficient c= 10	0.2540	0.2541	0.2542	0.2543	0.2544	0.2544	0.2542	0.2539	0.2530	0.2506	0.4358
Gini Coefficient c= 30	0.1578	0.1578	0.1578	0.1578	0.1578	0.1577	0.1575	0.1573	0.1569	0.1562	0.3157
Gini Coefficient c= 50	0.1248	0.1247	0.1247	0.1247	0.1247	0.1246	0.1245	0.1243	0.1241	0.1237	0.2591
Gini Coefficient c=100	0.0887	0.0887	0.0886	0.0886	0.0886	0.0885	0.0884	0.0883	0.0882	0.0880	0.1883
Gini Coefficient c=150	0.0713	0.0713	0.0712	0.0712	0.0712	0.0711	0.0711	0.0710	0.0709	0.0707	0.1521
Binary Preference (short queries)	0.0856	0.0847	0.0843	0.0839	0.0837	0.0832	0.0824	0.0815	0.0800	0.0785	0.0211
Binary Preference (long queries)	0.0818	0.0818	0.0810	0.0814	0.0813	0.0815	0.0821	0.0839	0.0827	0.0778	0.0022
Mean Average Precision (short queries)	0.0639	0.0635	0.0631	0.0627	0.0621	0.0615	0.0609	0.0598	0.0573	0.0548	0.0075
Mean Average Precision (long queries)	0.0516	0.0520	0.0524	0.0532	0.0533	0.0543	0.0574	0.0573	0.0583	0.0524	0.0000
Precision at 30 documents (short queries)	0.0581	0.0574	0.0568	0.0574	0.0570	0.0561	0.0552	0.0543	0.0495	0.0450	0.0007
Precision at 30 documents (long queries)	0.0399	0.0396	0.0392	0.0399	0.0392	0.0399	0.0405	0.0399	0.0396	0.0383	0.0000
Precision at Recall 50 (short queries)	0.0661	0.0655	0.0651	0.0643	0.0635	0.0627	0.0613	0.0592	0.0581	0.0566	0.0077
Precision at Recall 50 (long queries)	0.0519	0.0523	0.0530	0.0536	0.0538	0.0558	0.0617	0.0619	0.0652	0.0578	0.0000
Precision at Recall 100 (short queries)	0.0209	0.0209	0.0209	0.0209	0.0209	0.0209	0.0208	0.0208	0.0208	0.0192	0.0069
Precision at Recall 100 (long queries)	0.0192	0.0195	0.0198	0.0204	0.0205	0.0215	0.0225	0.0226	0.0221	0.0213	0.0000

List of Figures

List of Tables

Listings

Bibliography

[1] Apache. Apache Lucene Overview. `http://lucene.apache.org/java/docs/index.html`, 2012. [Online; accessed 01-February-2012].

[2] Javed A. Aslam and Robert Savell. On the effectiveness of evaluating retrieval systems in the absence of relevance judgments. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '03, pages 361–362, New York, NY, USA, 2003. ACM.

[3] Leif Azzopardi and Richard Bache. On the relationship between effectiveness and accessibility. In *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '10, pages 889–890, New York, NY, USA, 2010. ACM.

[4] Leif Azzopardi and Vishwa Vinay. Accessibility in information retrieval. In *Proceedings of the IR research, 30th European conference on Advances in information retrieval*, ECIR'08, pages 482–489, Berlin, Heidelberg, 2008. Springer-Verlag.

[5] Leif Azzopardi and Vishwa Vinay. Retrievability: an evaluation measure for higher order information access tasks. In *Proceedings of the 17th ACM conference on Information and knowledge management*, CIKM '08, pages 561–570, New York, NY, USA, 2008. ACM.

[6] Ricardo A. Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[7] Oleg Bartunov. OpenFTS full text search engine. `http://openfts.sourceforge.net/`, 2009. [Online; accessed 01-February-2012].

[8] Shariq Bashir. *Evaluating Retrieval Models using Retrievability Measures*. PhD thesis, Vienna University of Technology, Vienna, 2011.

[9] Shariq Bashir. Tool for Calculating Document Retrievability with Standard Retrieval Models. `http://www.ifs.tuwien.ac.at/~bashir/RetrievabilityToolStandard.htm`, 2011. [Online; accessed 12-December-2011].

[10] Shariq Bashir and Andreas Rauber. Identification of low/high retrievable patents using content-based features. In *Proceedings of the 2nd international workshop on Patent information retrieval*, PaIR '09, pages 9–16, New York, NY, USA, 2009. ACM.

[11] Shariq Bashir and Andreas Rauber. Improving Retrievability and Recall by Automatic Corpus Partitioning. *Transactions on Large-Scale Data- and Knowledge-Centered Systems*, 2:122–140, 2010.

[12] Richard K. Belew. *Finding Out About: A Cognitive Perspective on Search Engine Technology and the WWW*. Cambridge University Press, New York, NY, USA, 1 edition, 2008.

[13] Graham Bennett, Falk Scholer, and Alexandra Uitdenbogerd. A comparative study of probabilistic and language models for information retrieval. In *Proceedings of the nineteenth conference on Australasian database - Volume 75*, ADC '08, pages 65–74, Darlinghurst, Australia, Australia, 2007. Australian Computer Society, Inc.

[14] David C. Blair. STAIRS redux: thoughts on the STAIRS evaluation, ten years after. *Journal of the American Society for Information Science and Technology (JASIST)*, 47:4–22, January 1996.

[15] David C. Blair and M. E. Maron. An evaluation of retrieval effectiveness for a full-text document-retrieval system. *Communications of the ACM*, 28:289–299, March 1985.

[16] Paolo Boldi and Sebastiano Vigna. MG4J at TREC 2005. In *The Fourteenth Text REtrieval Conference (TREC 2005) Proceedings*, number SP 500-266 in Special Publications. NIST, 2005.

[17] Chris Buckley and Ellen M. Voorhees. Retrieval evaluation with incomplete information. In *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '04, pages 25–32, New York, NY, USA, 2004. ACM.

[18] Jamie Callan and Margaret Connell. Query-based sampling of text databases. *ACM Transactions on Information and System Security*, 19:97–130, April 2001.

[19] CLEF. The CLEF Initiative (Conference and Labs of the Evaluation Forum) - Homepage. `http://www.clef-initiative.eu/`, 2011. [Online; accessed 30-December-2011].

[20] Bruce Croft, Donald Metzler, and Trevor Strohman. *Search Engines: Information Retrieval in Practice*. Addison-Wesley Publishing Company, USA, 1st edition, 2009.

[21] Tonya Custis and Khalid Al-Kofahi. A new approach for evaluating query expansion: query-document term mismatch. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '07, pages 575–582, New York, NY, USA, 2007. ACM.

[22] Joseph L Gastwirth. The Estimation of the Lorenz Curve and Gini Index. *The Review of Economics and Statistics*, 54(3):306–16, August 1972.

[23] Mark S. Handcock and Eric M. Aldrich. Applying Relative Distribution Methods in R. University of Washington Working Paper No. 27, December 2002.

[24] Walter G. Hansen. How Accessibility Shapes Land Use. *Journal of the American Institute of Planners*, 25(2):73–76, 1959.

[25] K. Sparck Jones, S. Walker, and S. E. Robertson. A probabilistic model of information retrieval: development and comparative experiments - Part 1. *Information Processing and Management*, 36:779–808, November 2000.

[26] Donald H. Kraft, Gabriella Pasi, and Gloria Bordogna. Vagueness and uncertainty in information retrieval: how can fuzzy sets help? In *Proceedings of the 2006 international workshop on Research issues in digital libraries*, IWRIDL '06, pages 3:1–3:10, New York, NY, USA, 2007. ACM.

[27] Mounia Lalmas, Stefan M. Rueger, Theodora Tsikrika, and Alexei Yavlinsky. Progress in Information Retrieval. In *Advances in Information Retrieval, 28th European Conference on IR Research, ECIR 2006, London, UK, April 10-12, 2006, Proceedings*, volume 3936 of *Lecture Notes in Computer Science*, pages 1–11. Springer, 2006.

[28] Ken Lang and Jason Rennie. Text REtrieval Conference Frequently Asked Questions. `http://www.ai.mit.edu/~jrennie/20Newsgroups/`, 2008. [Online; accessed 30-December-2011].

[29] Lavtech. mnoGoSearch web search engine software. `http://www.mnogosearch.org/`, 2011. [Online; accessed 22-December-2011].

[30] Shuang Liu, Clement Yu, and Weiyi Meng. Word sense disambiguation in queries. In *Proceedings of the 14th ACM international conference on Information and knowledge management*, CIKM '05, pages 525–532, New York, NY, USA, 2005. ACM.

[31] Christopher D. Manning, Prabhakar Reghavan, and Hinrich Schuetze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[32] Christian Middleton and Ricardo Baeza-Yates. A Comparison of Open Source Search Engines. *Evaluation*, 2007.

[33] Abbe Mowshowitz and Akira Kawaguchi. Assessing bias in search engines. *Inf. Process. Manage.*, 38:141–156, January 2002.

[34] Rabia Nuray and Fazli Can. Automatic ranking of information retrieval systems using data fusion. *Information Processing and Management*, 42:595–614, May 2006.

[35] M. F. Porter. Readings in information retrieval. chapter An algorithm for suffix stripping, pages 313–316. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1997.

[36] Richard J. Potocko. An index system of terms with related and associated words. *American Documentation*, 19(2):146–150, 1968.

[37] NTCIR Project. NTCIR Project data. http://research.nii.ac.jp/ntcir/data/data-en.html, 2011. [Online; accessed 30-December-2011].

[38] Terrier Project. Terrier IR Platform v3.5 Homepage. http://terrier.org, 2011. [Online; accessed 27-December-2011].

[39] The Lemur Project. Lemur Project Home. http://www.lemurproject.org/, 2011. [Online; accessed 22-December-2011].

[40] The Lemur Project. The Lemur Toolkit - Lemur Indexing Applications. http://lemurproject.org/lemur/indexing.php, 2012. [Online; accessed 23-January-2012].

[41] The Namazu Project. Namazu: a Full-Text Search Engine. http://www.namazu.org/, 2011. [Online; accessed 01-February-2012].

[42] The R Project. The R Project for Statistical Computing. http://www.r-project.org/, 2011. [Online; accessed 14-February-2012].

[43] The Xapian Project. Xapian Project Home. http://xapian.org/, 2011. [Online; accessed 01-February-2012].

[44] Reidpath. Economics Gini coefficient. http://en.wikipedia.org/wiki/File:Economics_Gini_coefficient2.svg, 2009. [Online; accessed 05-January-2012].

[45] Riclas. Vector space model. http://en.wikipedia.org/wiki/File:Vector_space_model.jpg, 2010. [Online; accessed 28-December-2011].

[46] Tony Rose, Mark Stevenson, and Miles Whitehead. The reuters corpus volume 1 - from yesterday's news to tomorrow's language resources. In *Proceedings of the Third International Conference on Language Resources and Evaluation*, pages 29–31, 2002.

[47] Mark Sanderson and Justin Zobel. Information retrieval system evaluation: effort, sensitivity, and reliability. In *Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '05, pages 162–169, New York, NY, USA, 2005. ACM.

[48] Amit Singhal. Modern Information Retrieval: A Brief Overview. *IEEE Data Eng. Bull.*, 24(4):35–43, 2001.

[49] Ian Soboroff, Charles Nicholas, and Patrick Cahan. Ranking retrieval systems without relevance judgments. In *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '01, pages 66–73, New York, NY, USA, 2001. ACM.

[50] Sourceforge. Lemur Project and Indri Search Engine Wiki. `http://sourceforge.net/apps/trac/lemur/wiki/WikiStart`, 2012. [Online; accessed 23-January-2012].

[51] Amanda Spink and Howard Greisdorf. Regions and levels: measuring and mapping users' relevance judgments. *Journal of the American Society for Information Science and Technology (JASIST)*, 52:161–173, January 2001.

[52] Anselm Spoerri. Using the structure of overlap between search results to rank retrieval systems without relevance judgments. *Information Processing and Management*, 43:1059–1070, July 2007.

[53] Trevor Strohman, Donald Metzler, Howard Turtle, and W. Bruce Croft. Indri: a language-model based search engine for complex queries. Technical report, in Proceedings of the International Conference on Intelligent Analysis, 2005.

[54] TREC. Text REtrieval Conference English Relevance Judgements File List. `http://trec.nist.gov/data/qrels_eng/index.html`, 2010. [Online; accessed 22-January-2012].

[55] TREC. Text REtrieval Conference English Test Questions (Topics) Files List. `http://trec.nist.gov/data/topics_eng/index.html`, 2010. [Online; accessed 22-January-2012].

[56] TREC. Text REtrieval Conference Frequently Asked Questions. `http://trec.nist.gov/faq.html`, 2010. [Online; accessed 30-December-2011].

[57] Howard R. Turtle and W. Bruce Croft. A comparison of text retrieval models. *Computer Journal*, 35(3):279–290, June 1992.

[58] C. J. van Rijsbergen. *Information Retrieval*. Butterworth, 2nd edition, 1979.

[59] Ellen M. Voorhees. Variations in relevance judgments and the measurement of retrieval effectiveness. *Information Processing and Management*, 36:697–716, September 2000.

[60] Ellen M. Voorhees. Overview of TREC 2007. In *Proceedings of The Sixteenth Text REtrieval Conference*, TREC 2007, Gaithersburg, Maryland, USA, 2007.

[61] Ian H. Witten, Alistair Moffat, and Timothy C. Bell. *Managing Gigabytes : Compressing and Indexing Documents and Images*. Morgan Kaufmann, San Francisco, CA, 2nd edition, 1999.

[62] Shengli Wu and Fabio Crestani. Methods for ranking information retrieval systems without relevance judgments. In *Proceedings of the 2003 ACM symposium on Applied computing*, SAC '03, pages 811–816, New York, NY, USA, 2003. ACM.

[63] Maxime Zakharov. DataparkSearch Engine Homepage. `http://www.dataparksearch.org/`, 2011. [Online; accessed 01-February-2012].