Thomas Felber BSc

Context Aware Scientific Literature Recommendation using Information Retrieval Techniques

MASTER'S THESIS

to achieve the university degree of

Diplom-Ingenieur

Master's degree programme: Computer Science

submitted to

Graz University of Technology

Supervisor

Dipl.-Ing. Dr.techn. Roman Kern

Institute of Interactive Systems and Data Science
Head: Univ.-Prof. Dipl.-Inf. Dr. Stefanie Lindstaedt

Graz, February 2019

This document is set in Palatino, compiled with pdfLaTeX2e and `Biber`.

The LaTeX template from Karl Voit is based on KOMA script and can be found online: https://github.com/novoid/LaTeX-KOMA-template

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources. The text document uploaded to TUGRAZonline is identical to the present master's thesis.

Graz, ____________________ ________________________

Date Signature

Eidesstattliche Erklärung[1]

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe. Das in TUGRAZonline hochgeladene Textdokument ist mit der vorliegenden Masterarbeit identisch.

Graz, am ____________________ ________________________

Datum Unterschrift

[1]Beschluss der Curricula-Kommission für Bachelor-, Master- und Diplomstudien vom 10.11.2008; Genehmigung des Senates am 1.12.2008

Abstract

The problem of information overload is widely recognized today. Living in an information society, we are all affected by the increasing amounts of information becoming available every day. The impact of this phenomenon shows itself in several information related tasks, such as conducting a literature search, by making it difficult for people to find information relevant to their interests. In this work, we develop a recommender system capable of providing relevant literature recommendations for a pending citation in a scientific paper. We employ a content-based recommendation approach based on information retrieval techniques. The input to our system consists of the citation context around the pending citation while the output comprises a ranked list of documents serving as citation candidates. Within our experimental setup, we experiment with different query formulation strategies and retrieval models in order to improve the performance of the system. The evaluation of our system shows the potential of this approach, reaching a peak MRR of 0.416. This is further emphasized by the results gained from our contribution to the CL-SciSumm Shared Task 2017 where we achieve top results among all participating systems.

Contents

List of Figures

List of Tables

List of Algorithms

1. Introduction

With the rapidly expanding amount of information in our current era, it becomes increasingly difficult to find the right piece of relevant information and distinguish it from the irrelevant masses. This is a well known phenomenon in our time and is commonly referred to as *information overload*.

There is a broad spectrum of information related tasks falling victim to this phenomenon. One such task, for example, is the process of literature search: A general approach to conduct a literature search is to perform a keyword based search and then follow the chain of citations in the found documents. With the increasing amount of new information becoming available at all times, however, the number of found documents will become larger as well. Therefore, conducting a literature search becomes a more and more labor intensive task.

A similar scenario arises, when someone is writing a scientific paper and is about to cite a passage of text with a source. In this case, either the writer already knows a proper source or she has to perform a search in order to find out. Figure 1.1 illustrates this scenario for an article hosted on Wikipedia where a paragraph is marked with "citation needed". For the editors of this

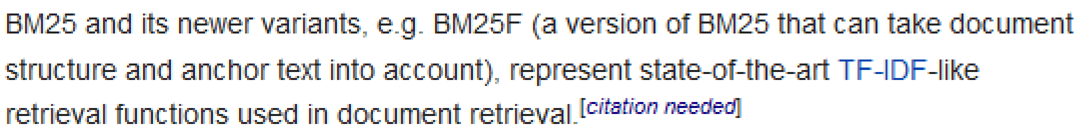

BM25 and its newer variants, e.g. BM25F (a version of BM25 that can take document structure and anchor text into account), represent state-of-the-art TF-IDF-like retrieval functions used in document retrieval.[citation needed]

Figure 1.1.: Example of a pending citation in an article on Wikipedia.

article, it certainly would be useful if a citation recommendation system could provide some candidate papers containing the required information about the used material.

This scenario is, where the focus of this thesis ties in: Over the course of this thesis, we develop a content-based recommender system with the idea of providing a set of documents serving as citation candidates.

Additionally, we participate at the CL-SciSumm Shared Task 2017, where we employ the approach taken in this thesis within the context of bibliometric-enhanced information retrieval and natural language processing for digital libraries in order to solve the problems posed by this task [1]. A closer look at the setup of the shared task and how well our approach performs is discussed in Section 6.1.

All the work done within the scope of this thesis, is contributed to a research question which we formulate as follows:

How reliable are content-based recommendations serving as citation candidates for a pending citation?

Over the next chapters, we provide a background to recommender systems and information retrieval and follow to describe how our system is designed and implemented. Furthermore, we describe the experiments conducted with the system and show the evaluation and results. After a discussion of related work, we come a conclusion of this thesis.

The according structure is as follows:

Chapter 2 This chapter gives an introduction to recommender systems and recommendation techniques followed by an introduction to information retrieval.

Chapter 3 In this chapter, we provide a general description of our system and discuss our data set.

Chapter 4 In this chapter, we cover the technical details of our system and provide algorithmic solutions to the problems we face.

Chapter 5 In this chapter, we show the evaluation of our system and interpret the results.

Chapter 6 In this chapter, we discuss related work on the recommendation of documents within the context of scientific writing and take at closer look at the CL-SciSumm Shared Task 2017.

[1]See Appendix A for the system report to our submission

Chapter 7 In this chapter, we recapitulate on the work conducted within the scope of this thesis and come to a conclusion.

2. Background

2.1. Recommender systems

A Recommender system is a special tool with the goal to help users with their information seeking tasks. This is achieved, by providing useful suggestions for items which best match the user's needs and preferences [46, 58].

The term "item" in this context refers to what kind of thing the system recommends and can be anything like a product, a service, or some piece of information.

Typically, recommender systems focus on only one specific type of item, and all inner mechanics of the system are tweaked and optimized towards the calculation of effective and useful suggestion of items of that particular type [60].

To accomplish its task of generating recommendations, the recommender system must be capable of coming up with a prediction of items which qualify as potentially useful to a user. In order to do that, various different approaches have been evolved since the beginning of recommender systems.

The two approaches most relevant within the scope of this thesis refer to *content-based recommendation* and *collaborative filtering*. How these approaches work, we described in the following sections.

Other more and less commonly used recommendation techniques comprise *community-based recommendation*, *knowledge-based recommendation*, *demographic recommendation*, *hybrid recommendation*. How all of these work, however, is beyond the scope of this thesis.

2.1.1. Content-based Recommender

Content-based recommender systems have their roots in information retrieval. Systems of this group calculate recommendations according to a simple principle: If a user has previously liked an item (e.g., by providing a rating) then similar items will be recommended in the future.

In practice, this works by analyzing the properties of all a user's previously rated items and derive a user profile from this information [50]. Such a user profile serves as a representation of the user's interests and is utilized to recommend new relevant items. This is done by comparing the attributes of the user profile with the attributes of the items. The result of a comparison between the user profile and a particular item reflects the user's degree of interest in that item. The more accurately the user profile represents the user's interests, the more effective an information access process can become. For example, let's consider a user is performing a search on a web shop yielding a set of products. In this case, the user profile could be used to determine whether or not the user is interested in a product, and, if not, then the product could be filtered out.

As described in [60], following a content-based approach exhibits various advantageous and disadvantageous properties compared to other approaches such as collaborative filtering (see Section 2.1.2).

Among the advantageous properties, there is:

User Independence For a target user, only his own ratings are taken into account when it comes to building his user profile. That is, recommendations do not depend on other users as it is the case with collaborative filtering.

Transparency An explanation why an item was recommended or not can be derived by comparing the item attributes with the user profile. This is in contrast to the collaborative filtering approach, where finding an explanation to a recommendation is more obscure, since, in this case, recommendations are based on unknown users with similar tastes and which items they liked.

New Item Items do not have to be rated in order to be considered for a recommendation. This is a major advantage over systems following a collaborative filtering approach. This is because systems of this type require a new item to be rated by a significant amount of users before the system would be able to recommend it.

Among the disadvantageous properties, there is:

Limited Content Analysis Enough discriminating information needs to be present in items in order to decide whether a user likes the item or not. Providing such information often requires special domain knowledge. For instance, for music recommendations, the system would need information such as band members, genre and song writer.

Over Specialization Items are only recommended if they display a good match against the user profile. Thus, only items similar to the ones already rated are going to be recommended. This circumstance is also known as *serendipity problem* and means that the amount of novel recommendations is vastly restricted.

New User Content-based recommenders need access to a sufficient amount of ratings in order to come up with a user profile. For a new user, with only a few ratings, an accurate user profile can not be created, hence, no reliable recommendations can be provided to the user.

2.1.2. Collaborative Filtering

The collaborative filtering approach tries to overcome some of the shortcomings present in content-based recommenders. In content-based recommenders, recommended items solely depend on a user's previously liked items. Systems based on collaborative filtering take a different route, with the basic idea being: If two different users u and v have rated items in a similar fashion then it is likely that the rating of user u for a new item i is similar to that of user v for item i.

Two common techniques of collaborative filtering are *user-user collaborative filtering* and *item-item collaborative filtering*.

User-User Collaborative Filtering

User-user collaborative filtering, or sometimes denoted as *k-NN collaborative filtering* is an algorithmic implementation of the basic idea behind collaborative filtering, namely, finding users with a similar rating pattern to that of the target user and use their ratings to other items in order to determine which items the target user will like.

The first recommender system taking this approach was the Usenet article recommender GroupLens [59], followed up by the Ringo [73] music recommender and the Bellcore video recommender [30].

Formally, what we are interested in, is to come up with a prediction for the rating r_{ui} where r_{ui} is the rating of a user u for an item i.

This is done by considering the nearest-neighbors of u with respect to their rating similarity: Let's consider the value s_{uv} where u and v are two users with $u \neq v$ and s_{uv} representing their rating similarity. Then, the users v with the highest similarity s_{uv} are the k nearest-neighbors of u and shall be denoted as $\mathcal{N}(u)$.

To make a prediction about r_{ui}, however, only users v who have rated the item i can be taken into account, therefore, we are interested in the users v with the highest similarity s_{uv} who have also rated item i. This set is denoted as $\mathcal{N}_i(u)$ and can be used to compute a prediction of r_{ui} [60]:

$$\hat{r}_{ui} = \frac{1}{|\mathcal{N}_i(u)|} \sum_{v \in \mathcal{N}_i(u)} r_{vi}$$

In this case r_{ui} is simply the average rating given to i by the users in $\mathcal{N}_i(u)$.

One drawback of this approach is its scalability problem [26]. Since finding the k nearest-neighbors is an $\mathcal{O}(|U|)$ operation, where $|U|$ is the size of the user base, it becomes increasingly expensive to predict r_{ui} as the user base keeps growing.

To tackle this problem, it was necessary to develop a more scalable approach. As a result an approach known as *item-item collaborative filtering* arised.

Item-Item Collaborative Filtering

Because of scalability issues with user-user collaborative filtering, it was necessary to find a more scalable approach which led to the development of *item-item collaborative filtering*, also known as *item-based collaborative filtering*.

First introduced by Sarwar et al. [72] and Karypis [38] item-item collaborative filtering is among the most widely used collaborative filtering techniques.

In contrast to user-user collaborative filtering, where the rating similarities between users are considered, item-item collaborative filtering focuses on the rating similarities between items. The key thought is, if two items have the same users rating them in a similar way then those items are considered to be similar and it is expected that users have a similar preference towards similar items.

As an example, imagine you wanted to find out if a particular book A is interesting to you. You observe that people who have rated book A have given a similar rating to two other books B and C that you have both read and liked. Thus, you draw the conclusion that you will also like book A.

In a formal sense, this idea can be captured as follows: As with user-user collaborative filtering, the goal is to predict the rating r_{ui} representing the rating of a user u for an item i. Likewise, we consider the value s_{ij} reflecting the similarity between items i and j. A value for s_{ij} is calculated between item i and all of user u's rated items j. Items of j with the highest similarity to i, are denoted by the set $\mathcal{N}_u(i)$. That is, $\mathcal{N}_u(i)$ contains items rated by u with highest similarity to i. With this set, a prediction for r_{ui} can be determined [72]:

$$\hat{r}_{ui} = \frac{\sum\limits_{j \in \mathcal{N}_u(i)} s_{ij} r_{uj}}{\sum\limits_{j \in \mathcal{N}_u(i)} |s_{ij}|}$$

Notably, in its basic form, item-item collaborative filtering does not pose any significant advantage over user-user collaborative filtering. It is still necessary to find the k nearest-neighbors which is an $\mathcal{O}(|I|)$ operation with $|I|$ being the total number of items in the system. At systems with $|U| >> |I|$, it allows us to find the k neareast-neighbors within a smaller dimension, but this is only a minor gain.

The actual advantage of item-item collaborative filtering comes into play when we are about to precompute similarities. The similarity s_{ij} between two items i and j depends on which ratings these items got from users in the system.

With a large enough user to item ratio, changes in a user's ratings are not going to dramatically change s_{ij}, especially if these items already have plenty of ratings. Hence, precomputing these similarities in the form of a similarity matrix is a reasonable task.

Even when users start to change their ratings, it is unlikely to immediately have a significant negative impact on the recommendation quality. Additionally, after some time, the similarity matrix can easily be brought up to date by recomputing it.

Similarity Measures

As described in the previous sections, it is necessary to calculate certain similarities between objects in order to predict a value for the rating r_{ui}. At user-user collaborative filtering, we are interested in similarities s_{uv} between two users u and v whereas at item-item collaborative filtering we are interested in similarities s_{ij} between two item i and j. Two of the most commonly used similarity measures are *Cosine Similarity* and *Pearson Correlation*.

Cosine Similarity Cosine Similarity (CS) takes a vector space approach towards calculating the similarity between two objects. Consider two objects a and b represented as two vectors $\vec{a}$ and $\vec{b}$, then the cosine similarity cs between these vectors is given as follows:

$$\text{CS}(a,b) = \frac{\vec{a} \bullet \vec{b}}{||\vec{a}||\ ||\vec{b}||}$$

where $\vec{a} \bullet \vec{b}$ is the inner product between the two and $\|\vec{a}\|, \|\vec{b}\|$ are the norms of vectors a and b.

Within the scope of user-user collaborative filtering, a user u can be represented by a vector $\vec{u} \in \mathcal{R}^{|I|}$, where $\vec{u}_i = r_{ui}$ in the case that the user has rated the item or 0 otherwise. Thus, the cosine similarity between two users u and v can be determined as:

$$\mathrm{CS}(u,v) = \frac{\vec{u} \bullet \vec{v}}{\|\vec{u}\| \, \|\vec{v}\|} = \frac{\sum\limits_{i \in I} r_{ui} r_{vi}}{\sqrt{\sum\limits_{i \in I} r_{ui}^2} \sqrt{\sum\limits_{i \in I} r_{vi}^2}}$$

In the case of item-item collaborative filtering, the same idea applies:

$$\mathrm{CS}(i,j) = \frac{\vec{i} \bullet \vec{j}}{\|\vec{i}\| \, \|\vec{j}\|} = \frac{\sum\limits_{u \in U} r_{ui} r_{uj}}{\sqrt{\sum\limits_{u \in U} r_{ui}^2} \sqrt{\sum\limits_{u \in U} r_{uj}^2}}$$

Pearson Correlation Pearson Correlation (PC) determines the similarity between two objects a and b by looking at their statistical correlation:

$$\mathrm{PC}(a,b) = \frac{\sum\limits_{i=1}^{n} (a_i - \bar{a})(b_i - \bar{b})}{\sqrt{\sum\limits_{i=1}^{n} (a_i - \bar{a})^2} \sqrt{\sum\limits_{i=1}^{n} (b_i - \bar{b})^2}}$$

where $\bar{a} = \frac{1}{n} \sum\limits_{i=1}^{n} a_i$, and analogously for $\bar{b}$.

For user-user collaborative filtering, where a user u is represented by his rating vector $\vec{u} \in \mathcal{R}^{|I|}$, the Pearson Correlation is given by:

$$\mathrm{PC}(u,v) = \frac{\sum\limits_{i \in I_u \cap I_v} (r_{ui} - \bar{r}_u)(r_{vi} - \bar{r}_v)}{\sqrt{\sum\limits_{i \in I_u \cap I_v} (r_{ui} - \bar{r}_u)^2} \sqrt{\sum\limits_{i \in I_u \cap I_v} (r_{vi} - \bar{r}_v)^2}}$$

And analogously for item-item collaborative filtering:

$$\mathrm{PC}(i,j) = \frac{\sum\limits_{u \in U_i \cap U_j} (r_{ui} - \bar{r}_i)(r_{uj} - \bar{r}_j)}{\sqrt{\sum\limits_{u \in U_i \cap U_j} (r_{ui} - \bar{r}_i)^2} \sqrt{\sum\limits_{u \in U_i \cap U_j} (r_{uj} - \bar{r}_j)^2}}$$

2.2. Information Retrieval

The main focus of information retrieval (IR) is to give users the possibility to access information of their interest in an easy and efficient way. Of course, this concept is not new and has been around for the last thousands of years.

The old Greeks had already realized that the right storage, organization and indexing of all their scrolls and books is vital to make future search and retrieval possible. They went so far as to construct dedicated buildings especially for this purpose, so called libraries. From that time onward, libraries began to spread across all civilizations in the world, and nowadays, a life without libraries is hard to imagine.

In a digitized world such as ours, the scope of IR goes way beyond just libraries. Baeza-Yates et al. summarize this as follows:

> "Information retrieval deals with the representation, storage, organization of, and access to information items such as documents, Web pages, online catalogs, structured and semi-structured records, multimedia objects. The representation and organization of the information items should be such as provide the users with easy access to information of their interest." [17]

The main objective of an IR system is to retrieve all relevant documents with regard to a user query while at the same time retrieving as few irrelevant documents as possible. This circumstance is also known as the *IR Problem* [17].

2.2.1. IR Modeling

When it comes to modeling an IR system, a key part is contributed to the definition of a ranking algorithm. The ranking algorithm is responsible for calculating scores for documents, with respect to a certain query, and plays a key role in the characterization of the model.

Formally, an IR model can be defined as follows [17]:

An IR model (IRM) is a quadruple

$$IRM = [D, Q, F, R(q_i, d_j)]$$

where

1. D is a set of logical views (or representations) of the documents in a collection.
2. Q is a set of logical views (or representations) of the user information needs.
3. F is a framework for modeling the representation of documents, queries and their relationships.
4. $R(q_i, d_j)$ is a ranking function that associates a real number to a query representation $q_i \in Q$ and a document representation $d_j \in D$.

Based on these parts, different models with different properties can be created, such as the *boolean model* (Section 2.2.4), the *vectors space model* (Section 2.2.5) or the *probabilistic model* (Section 2.2.6).

2.2.2. Representing Documents and Queries

As described in the previous section and illustrated in Figure 2.1, two main parts of an IR model are the sets of documents D and queries Q and their representation within the IR system. A basic way to represent documents and queries is the so called *bag of words* approach.

In a classic IR model, documents are represented by *keywords* or so called *index terms*. A keyword is a word or sequence of words in a document. Basically, any word in the document can serve as a keyword, and, in fact, search

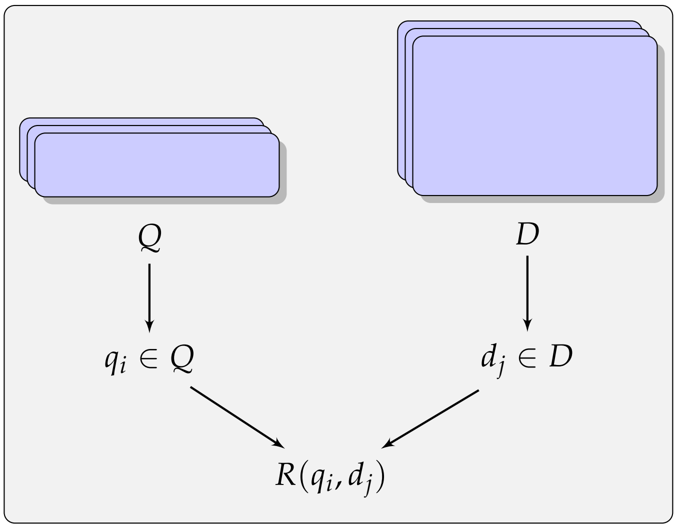


Figure 2.1.: The characterization of an IR model. A collection of documents D and queries Q, where representations of q_i and d_j are modeled according to a framework F. The ranking function $R(q_i, d_j)$ takes q_i and d_j as input and delivers a rank to document d_j with respect to query q_i.

engine designers usually take this approach. In more special cases, carefully selected word groups that best describe the topic of the document serve as keywords. Such an approach is often chosen by information scientists or librarians.

In any case, for each document d_j, we can extract the set of keywords and combine these sets in order to get the set of all distinct keywords across the entire document collection. This set is commonly referred to as the *vocabulary* of the collection and shall be denoted as $V = k_1, \ldots, k_t$, where t is the size of the vocabulary.

Let's consider the keywords k_a, k_b, k_c, k_d are part of document d_j. In this case we say that the pattern $[k_a, k_b, k_c, k_d]$ of term co-occurrence is present. With a vocabulary V of size t there is a total of 2^t different patterns of co-occurrence possible for each document. As an example, the pattern $(1, 0, \ldots, 0)$ means that only keyword k_1 is present in the document but no other, whereas the pattern $(1, 1, \ldots, 1)$ means that all keywords are present.

Each one of these 2^t different patterns is referred to as a *term conjunctive com-*

ponent [17]. A document d_j can be associated with a unique term conjunctive component $c(d_j)$, reflecting which keywords of the vocabulary occur in the document and which do not.

The same mechanic can be applied to a query q_i, where $c(q_i)$ reflects which keywords of the vocabulary are present in the query and which are not.

With this approach, documents and queries are represented by their respective term conjunctive components indicating presence or absence of keywords. This simple representation is the idea behind the bag of words approach mentioned at the beginning of this section.

2.2.3. Term-Document Matrix

The presence of a keyword in a document forms a relation between the two. To quantify this relation, the frequency by which the keyword occurs in the document can be considered. This relation can be expressed for all different document-keyword pairs by a so called term-document matrix [17]

$$\begin{array}{c c} & \begin{array}{ccc} d_1 & \dots & d_n \end{array} \\ \begin{array}{c} k_1 \\ \vdots \\ k_t \end{array} & \begin{bmatrix} f_{11} & \dots & f_{1n} \\ \vdots & \ddots & \vdots \\ f_{t1} & \dots & f_{tn} \end{bmatrix} \end{array}$$

where each f_{ij} is the number of times the keyword k_i appears in document d_j.

Capturing these frequencies provides more information than just recording whether or not a keyword is present in a document.

2.2.4. Boolean Model

In the boolean retrieval model [40] keywords are assumed to be either present or not present within documents. This implies that the corresponding term-document matrix contains only cells with values of either 0 (keyword is not present) or 1 (keyword is present one or more times).

Queries are formulated by creating a boolean expression of keywords, that is, logical operators such as AND, NOT and OR are used to combine the keywords with each other.

As an example, consider the vocabulary V of the document collection to be $V = \{k_1, k_2, k_3\}$ and the query of the user to be $q_i = (k_1 \wedge k_2) \vee \neg k_3$. With the query being a boolean expression, it is possible to rewrite it in its disjunctive normal form (DNF). To do this, we can construct the truth table corresponding to the query and directly extract the DNF from the table.

The DNF of query q_i is further denoted as q_{iDNF}. In our example, $q_{iDNF} = (1,1,0) \vee (1,1,1) \vee (0,1,0) \vee (1,0,0) \vee (0,0,0)$ and the corresponding truth table is shown in Table 2.1.

Table 2.1.: The truth table for the query q_i.

k_1	k_2	$k3$	$q_i = (k_1 \wedge k_2) \vee \neg k_3$
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

Let's consider there is a document d_j which contains the keywords k_1, k_2 but not k_3. This fact can be expressed as a logical clause of $(1,1,0)$ and is equivalent to conjunctive component $c(d_j)$ of document d_j. In this case, $c(d_j)$ is part of q_{iDNF} and thus document d_j satisfies the query q_i.

The similarity between a query q_i and a document d_j within this model is given as

$$sim(d_j, q_i) = \begin{cases} 1 & : \text{ if } c(d_j) \text{ part of } q_{iDNF} \\ 0 & : \text{otherwise} \end{cases}$$

This means that documents are either relevant (if $sim(d_j, q_i) = 1$) or not relevant (if $sim(d_j, q_i) = 0$) to a query. A measure for partial relevance is not supported in this model which often leads to a scenario where either too few or too many documents are retrieved. This circumstance is one of the main disadvantages of the boolean model. It's main advantages are that the model has a neat formalism and is simple to implement.

2.2.5. Vector Space Model

In the vector space model [69, 71], documents and queries are represented as vectors in the $t-$dimensional space where t corresponds to the number of keywords in the vocabulary V.

Each keyword k_i is associated with an orthonormal unit vector $\vec{k_i}$ in the $t-$dimensional space. This means $\vec{k_i}$ contains all 0s except for one 1 corresponding to the dimension that k_i is mapped to. For example, consider a $5-$dimensional space with a vocabulary of

$$V = \{\text{information}, \text{retrieval}, \text{word}, \text{system}, \text{context}\}$$

In this case, the keyword $k_{information}$ can be represented as a vector $\vec{k}_{information} = (1, 0, 0, 0, 0)$. Here, the keyword is mapped to the first dimension, similarly, the keyword $k_{context}$ could be mapped to the fourth dimension, yielding a vector representation of $\vec{k}_{context} = (0, 0, 0, 1, 0)$.

A document d_j or query q, containing several different keywords, can be represented in the $t-$dimensional vector space as illustrated in Figure 2.2. Formally, this idea can be captured as follows:

$$d_j = \sum_{i=1}^{t=|V|} k_i w_{ij} \quad \equiv \quad \vec{d_j} = (w_{1j}, w_{2j}, \ldots, w_{tj})$$

$$q = \sum_{i=1}^{t=|V|} k_i w_{iq} \quad \equiv \quad \vec{q_i} = (w_{1q}, w_{2q}, \ldots, w_{tq})$$

The elements w_{ij} and w_{iq} are weights with $w_{ij}, w_{iq} \geq 0$ chosen under a particular weighting scheme and are assigned to each keyword-document pair (k_i, d_j) and keyword-query pair (k_i, q) respectively.

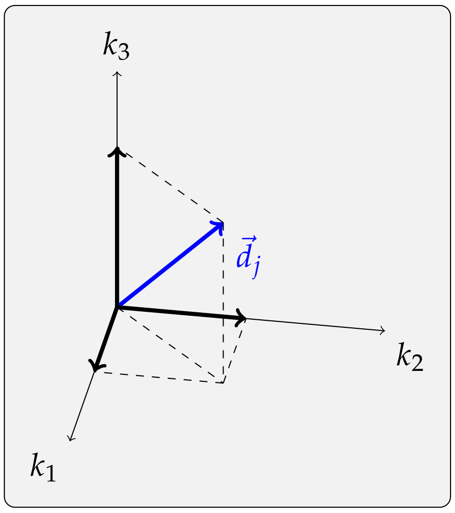


Figure 2.2.: The document vector $\vec{d_j}$ for document d_j.

To get a measure for the degree of similarity between a document d_j and a query q, the vector space model correlates the vectors $\vec{d_j}$ and $\vec{q}$ with each other. The quantification of this correlation can be done, for example, by taking the cosine of the angle between $\vec{d_j}$ and $\vec{q}$ as indicated in Figure 2.3.

This method of quantification is also called *cosine similarity* and is defined as:

$$\begin{aligned} sim_{cos}(d_j, q) &= \frac{\vec{d_j} \bullet \vec{q}}{\|\vec{d_j}\| \; \|\vec{q}\|} \\ &= \frac{\sum_{i=1}^{t=|V|}(w_{ij}w_{iq})}{\sqrt{\sum_{i=1}^{t=|V|} w_{ij}^2}\sqrt{\sum_{i=1}^{t=|V|} w_{iq}^2}} \end{aligned}$$

where $\vec{d_j} \bullet \vec{q}$ is the inner product between the document and query vector and $\|\vec{d_j}\|$, $\|\vec{q}\|$ are their respective vector norms. Note that since $w_{ij}, w_{iq} \geq 0$, $sim_{cos}(d_j, q)$ takes only values from 0 to 1.

Furthermore, note that with this definition, the vector space model can rank documents with respect to their *degree of similarity* to the query, meaning

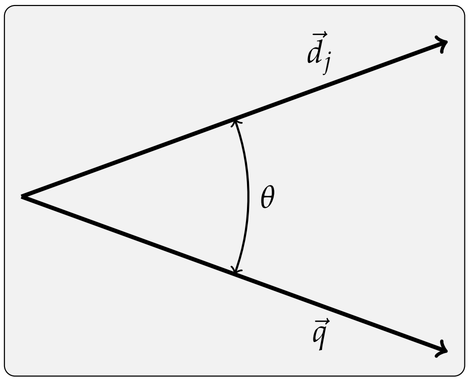


Figure 2.3.: The cosine of the angle θ between the document vector $\vec{d}_j$ and query vector $\vec{q}$ is used to calculate the cosine similarity.

that also documents can be retrieved which only partially match a query. This is a major improvement over the boolean model.

Besides cosine similarity, there are various other similarity functions such as Jaccard similarity [31] and Dice similarity [25].

Jaccard similarity compares the similarity and diversity of sets and is defined as the size of the intersection between two sets divided by their union. Consider Q to be the set of keywords in a query and D to be the set of all keywords in a document, then the Jaccard similarity can be computed by:

$$sim_{Jac}(D,Q) = \frac{|D \cap Q|}{|D \cup Q|}$$

Dice similarity uses an analogue approach and is defined as:

$$sim_{Dice}(D,Q) = \frac{2|D \cap Q|}{|D + Q|}$$

Weighting Schemes

Not only the choice of the similarity measure plays a role when i comes to evaluating how relevant a document is to a query, but also the adopted weighting scheme that is used to determine the weights w_{ij} and w_{iq}. This is because different weights mean a different orientation of the vectors $\vec{d_j}$ and $\vec{q}$ in the $t-$dimensional space.

The most popular weighting scheme in IR, and originally proposed by Salton et al. [68], is called TF-IDF weighting. It is based on term frequency (TF) and inverse document frequency (IDF) and defines the weight w_{ij} of a keyword k_i in a document d_j as:

$$w_{ij} = tf_{ij} \times idf_i$$

where tf_{ij} is a measure for the frequency of k_i in d_j and idf_i is a measure for the inverse of the frequency of documents containing k_i with regard to all documents in the collection.

The tf_{ij} and idf_i parts of above expression are also referred to as *TF weights* and *IDF weights* and depending on how these weights are defined take different values.

There is a variety of different ways how these weights can be defined, as, for instance, discussed in [70].

A whole list of different TF and IDF has been compiled from literature and is summarized in [17]. Some of these variants are listed in Table 2.2 and Table 2.3 and are described below, starting with TF.

The binary weighting method uses either 1 or 0 as TF weight, i.e. if a keyword k_i occurs in document d_j its TF weight is 1 and 0 otherwise. This method is used in the boolean model for example (see Section 2.2.4). A similar approach is taken by the raw frequency method. In this case the count of how often the keyword occurs in the document is used as TF weight. The log normalization method uses raw frequencies as base and applies the logarithm on it. This ensures a decreasing weight gain as the raw frequency grows. The double normalization 0.5 method ensures that weights are normalized by the maximum frequency of a keyword in the document. Additionally, it enforces that the weight takes only values between 0.5 and 1.

When it comes to IDF weighting schemes, the first method is the unary method. This one uses the constant 1 as IDF weight which means that IDF has no impact in TF-IDF weighting. The inverse frequency method is

Table 2.2.: Variants of TF weighting.

Weighting Method	TF Weight
binary	$0, 1$
raw frequency	f_{ij}
log normalization	$1 + \log f_{ij}$
double normalization 0.5	$0.5 + 0.5\frac{f_{ij}}{max_i f_{ij}}$

Table 2.3.: IDF weighting methods where N is the total amount of documents in the collection and n_i is the number of documents that contain keyword k_i.

Weighting Method	IDF Weight
unary	1
inverse frequency	$\log \frac{N}{n_i}$
inverse frequency smooth	$\log \left(1 + \frac{N}{n_i}\right)$

defined as the logarithm on the inverse of the fraction of documents in the collection that contain the keyword k_i. This is the standard IDF definition and is derived from the observation that the word frequencies in a natural language text follow a mathematical distribution, a circumstance that is also knows as Zipf's Law [81]. The inverse frequency smooth method works in a similar fashion by adding the constant 1 to the fraction. This improves the behavior of the function under extreme values of n_i.

By combining different TF and IDF variants, different TF-IDF weighting schemes can be obtained. Depending on the nature of the document collection, it can vary what the most appropriate weighting scheme is.

Table 2.4.: Recommended ways to determine TF-IDF weights for documents and queries.

Weighting Scheme	Document TF-IDF weights	Query TF-IDF weights
1	$f_{ij} \log_2 \frac{N}{n_i}$	$\left(0.5 + 0.5\frac{f_{iq}}{max_i\ f_{iq}}\right) \log_2 \frac{N}{n_i}$
2	$1 + \log_2 f_{ij}$	$\log_2 \left(1 + \frac{N}{n_i}\right)$
3	$(1 + \log_2 f_{ij}) \log_2 \frac{N}{n_i}$	$(1 + \log_2 f_{iq}) \log_2 \frac{N}{n_i}$

Salton analyzed different combinations of TF and IDF variants in [67] and proposes some well working combinations as shown in Table 2.4.

2.2.6. Probabilistic Model

The probabilistic model [65] tries to solve the IR problem by following a probabilistic approach in order to compute the similarity between documents and queries. Specifically, the similarity in this model is described as the probability that a document d_j is relevant to a query q.

The rationale behind this model has its roots in the uncertainty that is involved when it comes to matching a user's information need to the contents of documents [47]. In an IR system, a user's information need is translated to a query and given as its query representation. Likewise, documents are transformed to their document representations.

With these two representations, the system tries to figure out how well documents match the user's information need, however, doing so is a process plagued by uncertainty. This is because given solely the query representation of a query q, the system has only a vague understanding of the true information need. Subsequently, the determination of relevance between the contents of documents and the user's information need can be interpreted as the system performing a series of uncertain guesses.

With probability theory being well suited for domains where uncertainty is involved, methods have been researched in order to estimate how likely it is that a document d_j is relevant to a query q. One way that originated from such research expresses this likelihood as:

$$sim(d_j, q) = \frac{P(R|\vec{d_j}, q)}{P(\overline{R}|\vec{d_j}, q)}$$

where R is a set of initially guessed documents which are considered relevant to the user, $\overline{R}$ is the complement of R, $\vec{d_j}$ is the vector representation of document d_j using binary weights which indicate absence or presence of keywords in d_j, $P(R|\vec{d_j})$ is the probability that the document d_j is relevant to the query q, $P(\overline{R}|\vec{d_j}, q)$ is the probability that the document d_J is not relevant to the query q, and $sim(d_j, q)$ is the similarity between document d_j and query q.

This formula is the starting point used to derive one of the key expressions in the probabilistic model when it comes to ranking computation. How this derivation works is illustrated in [17] and starts with applying Bayes' rule such that:

$$sim(d_j, q) = \frac{P(\vec{d_j}|R, q)P(R|q)}{P(\vec{d_j}|\overline{R}, q)P(\overline{R}|q)}$$

Since $P(R|q)$ and $P(\overline{R}|q)$ are constant for all documents, this can be simplified to

$$sim(d_j, q) = \frac{P(\vec{d_j}|R, q)}{P(\vec{d_j}|\overline{R}, q)}$$

where $P(\vec{d_j}|R, q)$ denotes the probability that a randomly selected document of the set R with respect to query q has the representation d_j and $P(R|q)$ denotes the probability that a randomly selected document of the set R is relevant to q.

As we have mentioned above, the document representation of d_j is a vector of binary weights reflecting presence or absence of a keyword in the document. Under the assumption that keywords are independent among each other $sim(d_j, q)$ can be written as

$$sim(d_j, q) = \frac{(\prod_{k_i|w_{ij}=1} P(k_i|R, q))(\prod_{k_i|w_{ij}=0} P(\overline{k_i}|R, q))}{(\prod_{k_i|w_{ij}=1} P(k_i|\overline{R}, q))(\prod_{k_i|w_{ij}=0} P(\overline{k_i}|\overline{R}, q))}$$

Here, $P(k_i|R, q)$ denotes the probability that a randomly selected document of the set R with respect to query q contains the keyword k_i. Analogously, $P(\overline{k_i}|R, q)$ denotes the denotes the probability that a randomly selected document of the set R with respect to query q does not contain the keyword k_i.

Because $P(k_i|R, q) + P(\overline{k_i}|R, q) = 1$ and $P(k_i|\overline{R}, q) + P(\overline{k_i}|\overline{R}, q) = 1$ another rewrite step yields

$$sim(d_j, q) = \frac{(\prod_{k_i|w_{ij}=1} P(k_i|R, q))(\prod_{k_i|w_{ij}=0}(1 - P(k_i|R, q)))}{(\prod_{k_i|w_{ij}=1} P(k_i|\overline{R}, q))(\prod_{k_i|w_{ij}=0}(1 - P(k_i|\overline{R}, q)))}$$

Table 2.5.: Contingency table where R is the number of documents relevant to a query q, r_i is the number of relevant documents that contain the keyword k_i, N is the number of document in the collection, n_i is the number of document in the collection that contain the keyword k_i.

	#relevant	#not relevant	#total
Documents containing k_i	r_i	$n_i - r_i$	n_i
Documents not containing k_i	$R - r_i$	$N - n_i - (R - r_i)$	$N - n_i$
All documents	R	$N - R$	N

By applying the logarithm (which does not affect the ranking itself but only the absolute rank values) and assuming that

$$\forall k_i \notin q, P(k_i|R,q) = P(k_i|\overline{R},q)$$

we can transform into

$$sim(d_j,q) = \sum_{k_i \in d_j \wedge k_i \in q} \log \frac{P(k_i|R,q)}{1 - P(k_i|R,q)} + \log \frac{1 - P(k_i|\overline{R},q)}{P(k_i|\overline{R},q)}$$

This expression is a key expression in the probabilistic model when it comes to ranking calculation.

One problem with this approach is that initially we do not know the set R and therefore, we also can not directly calculate the probabilities $P(k_i|R,q)$ and $P(k_i|\overline{R},q)$.

Robertson et al. [65] investigated various statistical approaches in order to take advantage of relevance information during term weighting and came up with a contingency table as shown in table 2.5.

Using this contingency table, we can reinterpret our formula from before with

$$P(k_i|R,q) = \frac{r_i}{R}$$

and

$$P(k_i|\overline{R},q) = \frac{n_i - r_i}{N - R}$$

yielding

$$sim(d_j, q) = \sum_{k_i \in d_j \wedge k_i \in q} \log \frac{r_i(N - n_i - R + r_i)}{(R - r_i)(n_i - r_i)}$$

In order to mitigate extreme conditions related to small values of r_i a constant value of 0.5 is added to each instance of r_i resulting in

$$sim(d_j, q) = \sum_{k_i \in d_j \wedge k_i \in q} \log \frac{(r_i + 0.5)(N - n_i - R + r_i + 0.5)}{(R - r_i + 0.5)(n_i - r_i + 0.5)}$$

Note, that to this point, we still do not know exact values for r_i and R which makes it impossible to compute an actual value for $sim(d_j, q)$. One way around this is to assume $r_i = 0$ and $R = 0$ leading to

$$sim(d_j, q) = \sum_{k_i \in d_j \wedge k_i \in q} \log \frac{(N - n_i + 0.5)}{(n_i + 0.5)} \tag{2.1}$$

which is the formula used in the probabilistic model in order to do ranking computation. Many experiments have been conducted using variations of this formula and improvements have been discovered leading to the BM25 model which we discuss now.

BM25 Model

The BM25 model evolved out of experiments conducted on variations of the ranking equation 2.1 in the probabilistic model as described in Section 2.2.6. These experiments were driven by observations in the vector space model, where good term weighting is determined by inverse document frequency (IDF), term frequency (TF) and document length normalization. The ranking equation 2.1, however, neither accounts for TF nor for document length normalization.

The experiments mentioned above were conducted in conjunction with the Okapi system [62, 63, 64]. At first, equation 2.1 was used as is, which was referred to as BM1 (BM is short for *Best Match*).

$$sim(d_j,q)_{BM1} = \sum_{k_i \in d_j \wedge k_i \in q} \log \frac{(N - n_i + 0.5)}{(n_i + 0.5)}$$

As a first step of improvement, TF was introduced via a factor

$$F_{ij} = S_1 \frac{f_{ij}}{K_1 + f_{ij}}$$

where f_{ij} is the frequency of keyword k_i in document d_j, K_1 is a constant chosen experimentally for the collection, and S_1 is a scaling factor with $S_1 = K_1 + 1$.

The same idea was followed in order to introduce TF to queries:

$$F_{iq} = S_2 \frac{f_{iq}}{K_2 + f_{iq}}$$

where f_{iq} is the frequency of keyword k_i in query q, K_2 is a constant and S_2 is a scaling factor with $S_2 = K_2 + 1$

As a next step, document length normalization was accounted for, yielding

$$F\prime_{ij} = S_1 \frac{f_{ij}}{\frac{K_1 \, L(d_j)}{\alpha} + f_{ij}}$$

where $L(d_j)$ is the length of document d_j, and α is the average document length across the whole document collection.

Another proposal was to introduce a factor

$$G_{jq} = K_3 \, L(q) \frac{\alpha - L(d_j)}{\alpha + L(d_j)}$$

to account for document and query length, where $L(q)$ is the length of the query and K_3 is a constant.

Adding these factors to the BM1 equation led to the development of other BM variants such as BM11 and BM15 [17]:

$$sim(d_j,q)_{BM11} = G_{jq} + \sum_{k_i \in d_j \wedge k_i \in q} F\prime_{ij} \, F_{iq} \log \left(\frac{N - n_i + 0.5}{n_i + 0.5} \right)$$

$$sim(d_j,q)_{BM15} = G_{jq} + \sum_{k_i \in d_j \wedge k_i \in q} F_{ij}\, F_{iq} \log\left(\frac{N - n_i + 0.5}{n_i + 0.5}\right)$$

Experiments in [66] reflect that the best value for K_3 is 0 which eliminates G_{iq} from the equations above. Furthermore, they suggest that large values for K_2 work better for the scaling factor S_2 with $S_2 = K_2 + 1$, thus F_{iq} is simplified to f_{iq}.

With these insights the BM11 and BM15 ranking equation can be simplified to

$$sim(d_j,q)_{BM11} = \sum_{k_i \in d_j \wedge k_i \in q} \frac{f_{ij}(K_1+1)}{K1 + f_{ij}} \log\left(\frac{N - n_i + 0.5}{n_i + 0.5}\right)$$

$$sim(d_j,q)_{BM15} = \sum_{k_i \in d_j \wedge k_i \in q} \frac{f_{ij}(K_1+1)}{\frac{K_1\, L(d_j)}{\alpha} + f_{ij}} \log\left(\frac{N - n_i + 0.5}{n_i + 0.5}\right)$$

These equations are the ones used in practice when it comes to ranking computation in conjunction with the BM11 and BM15 model. In general the BM11 model outperforms the BM15 model which can be explained by the lack of document length normalization of the BM15 model.

The BM25 model combines the ranking equations of BM11 and BM15 via

$$sim(d_j,q)_{BM25} = \sum_{k_i \in d_j \wedge k_i \in q} \frac{f_{ij}(K_1+1)}{K_1\left[(1-b) + b\frac{L(d_j)}{\alpha}\right] + f_{ij}} \log\left(\frac{N - n_i + 0.5}{n_i + 0.5}\right)$$

where the newly added value b is a constant taking values in the closed interval $[0,1]$. With $b = 1$ the equation transforms into standard BM11 whereas with $b = 0$ it transforms into standard BM15. For all other cases it transforms into a combination of BM11 and BM15.

Empirical evidence shows that $K_1 = 1$ [66] and $b = 0.75$ [64] work well with real collections. Experimenting with these parameters under a particular collection and tuning them can increase the performance even further.

Nowadays, BM25 is thought to yield even better results than the classic vector space model and serves as a baseline for evaluating new ranking approaches [17].

2.2.7. Textual IR

A textual IR system deals with documents and queries, both expressed in natural language [22]. The typical retrieval process for this kind of system is illustrated in Figure 2.4 and can be summarized as follows:

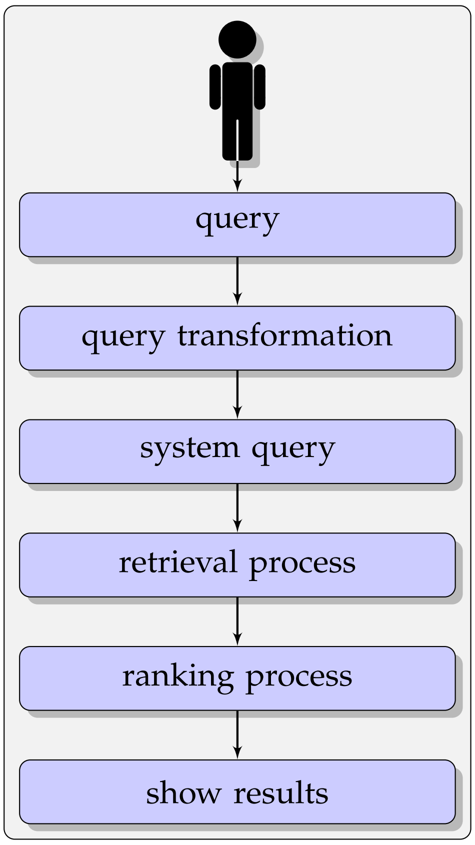


Figure 2.4.: Typical IR process.

1. In a first step, the user formulates a textual query Q_{usr} (usually a set of keywords) as a representation of his information need and passes it to the user interface of the IR system.
2. Then, the query Q_{usr} is processed by the system. That is, the system parses the query and translates it into a new query $\widetilde{Q}_{usr}$, by applying textual operations on it.

3. In a final transformation step, the query $\widetilde{Q}_{usr}$ is adapted in a way such that it can be executed on the system, thus, yielding a system representation of the query denoted as Q_{sys}.
4. In this step, the IR system runs the query Q_{sys} on top of the document collection D and returns a set of relevant documents R. In order to achieve good performance, a previously created index I is used for this task. The index I is a special data structure that consist of all documents in the document corpus.
5. Next, all the documents in R are sorted according to their rank, where the rank of a document expresses how relevant it is to the query.
6. Finally, the sorted List of documents in R is shown to the user. By manually marking documents as "definitely of interest", the user can give feedback to the system.

A closer look at indexing and textual operations is presented in the next sections.

2.2.8. Indexing

Searching through all the documents in the collection every time a query is performed in order to generate results is an inefficient and sometimes even impossible solution. Imagine your document collection consists of billions of entries. Scanning through all of them would take an unacceptable amount of time. Therefore, a special data structure is needed which allows rapid query processing. Such a data structure is called *index*.

The index is a representation of all documents in the collection by a set of keywords. In general, the keywords of a document are any words in the document, in a more restrictive environment, however, the keywords are those words that best describe the core concept of the document. This method is commonly chosen by librarians or information scientists.

As already discussed in Section 2.2.2, the vocabulary V is the set of all distinct keywords of the whole document collection:

$$V = (k_1, k_2, k_3, ..., k_n)$$

In its simplest form, a term-document matrix as discussed in Section 2.2.3 can be used to fulfill the role of an index. Using such a matrix offers a very fast way to determine, whether or not a keyword k_i exists in a document d_j.

A problem that comes along with this approach is that the size of the matrix is proportional to the number of documents in the collection multiplied by the number of keywords in the vocabulary. Additionally, the matrix is very sparse. This is because each document only contains few distinct keywords in relation to the size of the vocabulary.

A way to circumvent this problem is to use a so called *inverted index*. An inverted index associates each keyword k_i of the vocabulary V with a list L_i. The list L_i comprises all documents containing k_i. Optionally, L_i can also hold information about how often k_i occurred in the document. Figure 2.5 shows the relation between the term-document matrix and the inverted index.

Vocabulary	**Documents**				
	d_1	d_2	d_3	d_4	d_5
search	3		1		
retrieval	4		1		2
set		3		3	
is	2	3	2	1	4
data			3		
index	3				3
the		2			
do	5	2	4	3	
to	3				1
term		1		1	

Vocabulary	**Occurrences**
search	[1,3],[3,1]
retrieval	[1,4],[3,1],[5,2]
set	[2,3],[4,3]
is	[1,2],[2,3],[3,2],[4,1],[5,4]
data	[3,3]
index	[1,3],[5,3]
the	[2,2]
do	[1,5],[2,2],[3,4],[4,3]
to	[1,3],[5,1]
term	[2,1],[4,1]

Figure 2.5.: On the left hand side: The term-document matrix for an exemplary document collection and its vocabulary. Each cell contains the frequency of the keyword in the document. Empty cells indicate absence of the keyword in the document. On the right hand side: The corresponding inverted index consisting of lists whose entries reflect the document containing the keyword and the corresponding frequency.

When it comes to retrieval efficiency, inverted indexes are hardly rivaled as the reduce storage requirements. The size of an inverted index is proportional to occurrences of words in documents which is notably smaller than the size of the text.

Operations on Text

Considering the textual content of a document, it is noticeable that not every word carries the same amount of information. For instance, high frequency words such as conjunctions and prepositions only play an insignificant role when it comes to describing the semantic of a document. Additionally, these types of words appear many times across all documents, thus, providing only minor discriminative power.

In contrast, nouns are typically the words which carry the highest amount of information. Moreover, nouns potentially have high discriminative power. This is because some nouns only appear in certain contexts which are only relevant for specific documents. For example, think of a special technical term, that only appears in documents related to a particular technology.

These observations are taken into account when an index is created. Two properties that an index should fulfill are *exhaustiveness* and *specificity* [22]. Exhaustiveness describes the property where sufficiently many terms should be chosen to describe a document. Specificity means that generic words should be excluded as they only carry little information and only inflate the size of the index.

Figure 2.6 shows different text processing phases in an IR system entered when a document is indexed. What theses phases are responsible for is discussed in [22] and is summarized below.

Parsing During this phase so called unit documents are created where the structure of the documents is broken down into individual components. For instance, an email with attachments is split up into a document representing the email and as many documents as there are attachments.

Lexical Analysis During this phase, documents are tokenized into words. Difficulties of this task involve the correct recognition of different time and date formats, abbreviations, accents, cases, diacritics and

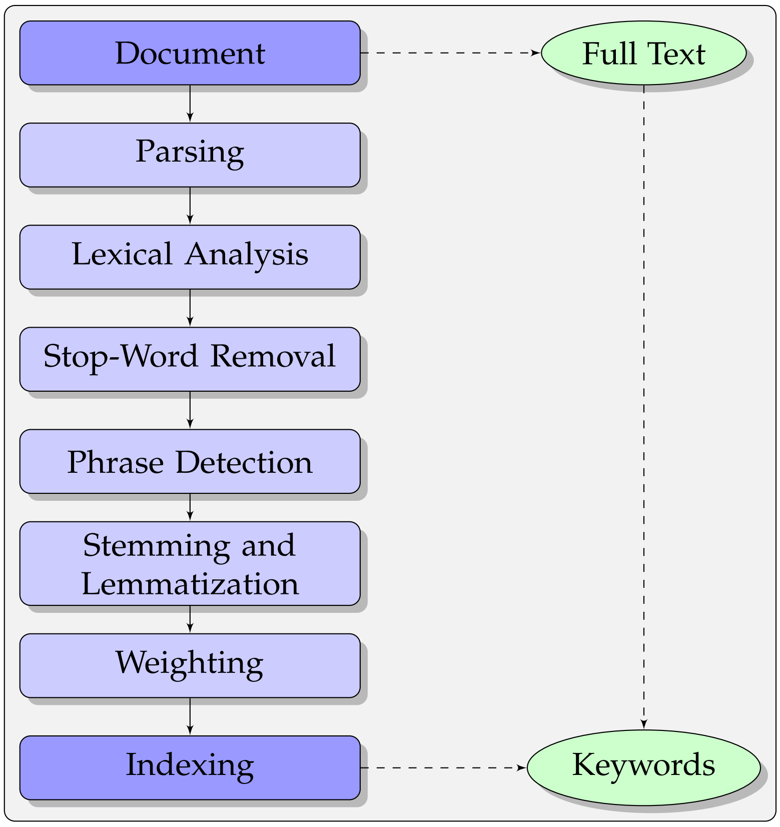


Figure 2.6.: Text processing steps in an IR system.

so on. Depending on different, language properties this can be an increasingly difficult task.

Stop-Word Removal During this phase, stop-words are removed from the previously tokenized documents. Stop-words are high frequency words that do not bear a lot of information such as conjunctions and prepositions. A possible problem induced by this phase is that the recall of the system may be decreased. Because of this, search engines usually do not remove stop-words [47].

Phrase Detection This phase tries to identify noun groups or other phrases. For instance, for a sentence like "An information retrieval system tries to help users find information" the noun group "information retrieval" should be identified.

Stemming and Lemmatization The purpose of this phase is to normalize words by reducing them to their word stem. Stemming, in particular, tries to do this by cutting off word ends in the hope that the resulting word is reduced correctly. A classic algorithm following this approach is known as Porter stemmer [51].
Lemmatization tries to reduce words to their stem by using dictionaries and morphological analysis.

Weighting This phase deals with the weighting of keywords. As already mentioned at the beginning of this section, different words have different discriminative power. To account for this, keywords are weighted differently in order to reflect their significance within a document or document collection.

3. System Description

Within the scope of this thesis, we design and implement a recommender system capable of generating literature suggestions on the fly for a pending citation. A pending citation in this context refers to the scenario where someone who is writing a scientific paper is about to cite a passage of text with a source. In this situation, either the writer already knows the proper source to cite and is done or she has to perform a literature search in order to find out.

This is where the idea behind creating such a recommender system ties in: Instead of having the writer perform a time consuming literature search, our goal is to have a recommender system available to suggest a correct source to cite on request. In order to realize such a system, we define several requirements which we deem necessary to be met for the successful implementation and operation of such a system:

Data Set The availability of a data set is the most fundamental requirement for our system. This is because no recommendations can be computed if there is no data. Furthermore, having a data set turns out quite useful when it comes to evaluating the performance of the system. Section 3.1 discusses the data set that we end up using for our system.

Responsiveness Our system has to have a low response time when recommendations are requested. Otherwise, it would not be very convenient to use such a system in practice, especially when recommendations are requested at a high frequency. In Section 3.2, we describe how we meet this requirement by creating an index.

Accurate Recommendations If the requested recommendations are not accurate, the resulting system is not of much use. To tackle this problem, we take advantage of well researched techniques within the field of information retrieval. See Section 3.3 for details about that.

3.1. The Data Set

For our recommender system, we use a data set obtained from the Pubmed Central Open Access Subset. Pubmed Central (PMC) is an archive at the U.S. National Institutes of Health's National Library of Medicine (NIH/NLM). This archive provides free full-text journal literature of biomedical and life sciences [54].

The PMC Open Access Subset contains a subset of articles available in PMC. Articles within this subset are made available under a Creative Commons or similar license. This allows more liberal redistribution and reuse of the articles compared to a collection with more traditional copyright in place. As of 2015, over 1 million articles are available in this collection [56].

Every article in the PMC Open Access Subset is available in two file formats. These formats are XML and PDF. Supplementary pieces of data such as image files, background research data, mathematical equations or videos are included as well. XML files are structured according to the NLM/JATS DTD [36]. This ensures a well defined structure across all articles.

Access to the articles in the PMC Open Access Subset is provided by the PMC FTP Service [55]. The base URL to access this service is `ftp://ftp.ncbi.nlm.nih.gov/pub/pmc`. We used this service to download all available articles, in particular, the archives named *Articles.A-B.tar.gz*, *Articles.C-H.tar.gz*, *Articles.I-N.tar.gz* and *Articles.O-Z.tar.gz* are acquired. These archives, when extracted, yield the desired articles and serve as a foundation for our data set.

3.2. Indexing

In order to efficiently access information contained in our data set, we utilize a special data structure known as index. The main benefits offered by this structure are great size reduction and rapid data access as already discussed in Section 2.2.8.

To build such an index, we use a software library known as Apache Lucene [76]. Apache Lucene is a Java-based cross platform programming library offering a solution to scalable, high-performance indexing. Other noteworthy features of Lucene involve:

- ranked searching where best results are returned at first
- various supported query types such as phrase queries, wildcard queries, proximity queries, range queries and more
- fielded searching (e.g. title, author, contents)
- sorting by any field
- various ranking models, including the Vector Space Model and BM25

Recall Figure 2.6 in Section 2.2.8 where we show that a document (or in this context, an article) typically undergoes several text processing steps before it is indexed. The first step is concerned with parsing the document, which is a step we perform as well. The reason why we do this, is to convert our articles to our own well defined XML structure. This allows us to access the information we need more easily during later tasks.

For the parsing procedure, we use a tool specifically developed for parsing Pubmed articles [19]. This tool parses the articles and saves the contained data to a database. Via a separate feature of this tool, it is possible to export the parsed data into a set of three XML files named *Articles.XML*, *ExtendedArticles.XML* and *Citations.XML*. The structure of these XML files is described in [19] and summarized below:

The *Articles.XML* file contains metadata about every parsed article. These metadata are associated with the following properties:

- **ID:** A unique number that the article can be identified by
- **Title:** The title of the article
- **Authors:** The authors that contributed to the article
- **Publication date:** The publication date (year, month, day) of the article

The corresponding Document Type Definition (DTD) is given as:

```
<!ELEMENT articles (article*)>
<!ELEMENT article (articleId, title, pubDate, authors)>
<!ELEMENT articleId (#PCDATA)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT pubDate (year, month, day)>
```

```
<!ELEMENT year (#PCDATA)>
<!ELEMENT month (#PCDATA)>
<!ELEMENT day (#PCDATA)>
<!ELEMENT authors (author*)>
<!ELEMENT author (firstname, lastname)>
<!ELEMENT firstname (#PCDATA)>
<!ELEMENT lastname (#PCDATA)>
```

The *ExtendedArticles.XML* file contains additional information about every parsed article such as abstract or body. This is reflected via the following properties:

- **ID:** A unique number that the article can be identified by (this number matches the ID in Articles.XML if the information belongs to the same)
- **Keywords:** The keywords associated with the article
- **Abstract:** Full text of the abstract of the article
- **Body:** Full text of the content of the article
- **References:** Other articles referenced in this article. The referenced articles are given by ID.

The corresponding DTD is given as:

```
<!ELEMENT extendedArticles (extendedArticle*)>
<!ELEMENT extendedArticle (articleId, keywords, abstract,
    articleBody?, references)>
<!ELEMENT articleId (#PCDATA)>
<!ELEMENT keywords (kwd*)>
<!ELEMENT kwd (#PCDATA)>
<!ELEMENT abstract (#PCDATA)>
<!ELEMENT articleBody (#PCDATA)>
<!ELEMENT references (referencedArticleId*)>
<!ELEMENT referencedArticleId (#PCDATA)>
```

The *Citations.XML* file contains each and every citation present within any parsed article. The following properties are associated with each citation:

- **Source article:** The ID of the source article. This is the article where the citation occurred in

- **Referenced article:** The ID of the article that is referenced by the citation
- **Context:** Full text of the paragraph that the citation occurred in
- **Section:** Section that the citation occurred in

The corresponding DTD is give as:

```
<!ELEMENT citations (citation*)>
<!ELEMENT citation (sourceArticleId, referencedArticleId, context,
    section)>
<!ELEMENT sourceArticleId (#PCDATA)>
<!ELEMENT referencedArticleId (#PCDATA)>
<!ELEMENT context (#PCDATA)>
<!ELEMENT section (sectionType?, sectionTitle, section?)>
<!ELEMENT sectionType (#PCDATA)>
<!ELEMENT sectionTitle (#PCDATA)>
```

Over the course of approximately one month, we were able to parse 152456 articles with the tool mentioned above. The parsed data has been saved to a database and was subsequently exported to the files *Articles.XML*, *ExtendedArticles.XML* and *Citations.XML*.

With these files at hand, we can begin to plan some basic characteristics of our index. Within the context of Apache Lucene, objects called *Documents* are the central units of indexing and search and are defined as follows:

> "Documents are the unit of indexing and search. A Document is a set of fields. Each field has a name and a textual value. A field may be stored with the document, in which case it is returned with search hits on the document. Thus each document should typically contain one or more stored fields which uniquely identify it."[6]

Furthermore, *Fields* are defined as follows:

> " A field is a section of a Document. Each field has three parts: name, type and value. Values may be text (String, Reader or pre-analyzed TokenStream), binary (byte[]), or numeric (a Number). Fields are optionally stored in the index, so that they may be returned with hits on the document."[8]

With this information in mind, we plan the following fields to be available in our index:

- **ArticleID:** Stores the unique identifier of an article
- **Title:** Stores the title of an article
- **Authors:** Stores the list of authors on an article
- **Publication_Day:** Stores the day of the publication of an article
- **Publication_Month:** Stores the month of the publication of an article
- **Publication_Year:** Stores the year of the publication of an article
- **Abstract:** Stores the full text of the abstract of an article
- **Body:** Stores the full text content of an article
- **Keywords:** Stores the keywords of an article
- **References:** Stores the references of an article

This structure essentially combines the information available in the files *Articles.XML* and *ExtendedArticles.XML* and, in conjunction with an index, allows us to access it in a fraction of time.

3.3. Recommendation Generation

In order to compute literature recommendations for a pending citation, we take advantage of well researched techniques within the field of information retrieval. In particular, we use the vector space model (see Section 2.2.5) and the BM25 model (see Section 2.2.6) in conjunction with our index to find the most suitable literature to a pending citation. The way we do this can be summarized by the following steps:

1. In the first step, we determine a *citation context* for the pending citation. This is done under the consideration of various aspects as illustrated in Table 3.1.
2. In the second step, the citation context is used to formulate a query. The index subsequently processes this query and performs an information retrieval task. As a result, the index provides a set of documents. These documents are ranked according to a score, where documents with a high score are on top. The score reflects how relevant a document is

with respect to the query, that is, a high score signifies high relevance and vice versa.

3. In the third and last step, a subset, consisting of the top *n* highest scored documents, is returned as result and presented to the user. The value of *n* should not be chosen to be too large as returning too many documents would make it difficult for the user to find the best match (the more documents are returned the more documents the user has to go through and decide if it is a good match or not).

How these steps are implemented in detail, is discussed in the next chapter.

Table 3.1.: Aspects of Citation Contexts

Option	Example Setting	Remark
Length	30, 50, 100, etc.	Determines how many words the citation context comprises.
Preprocessing	Stop word removal, stemming, lemmatization, etc.	Which Preprocessing steps (if any) to perform on the citation context.
Scope	before citation, after citation, before and after citation, etc.	Determines which words to consider for the citation context. E.g.: Only words after the citation, only words before the citation, words before and after the citation, ...

4. System Implementation

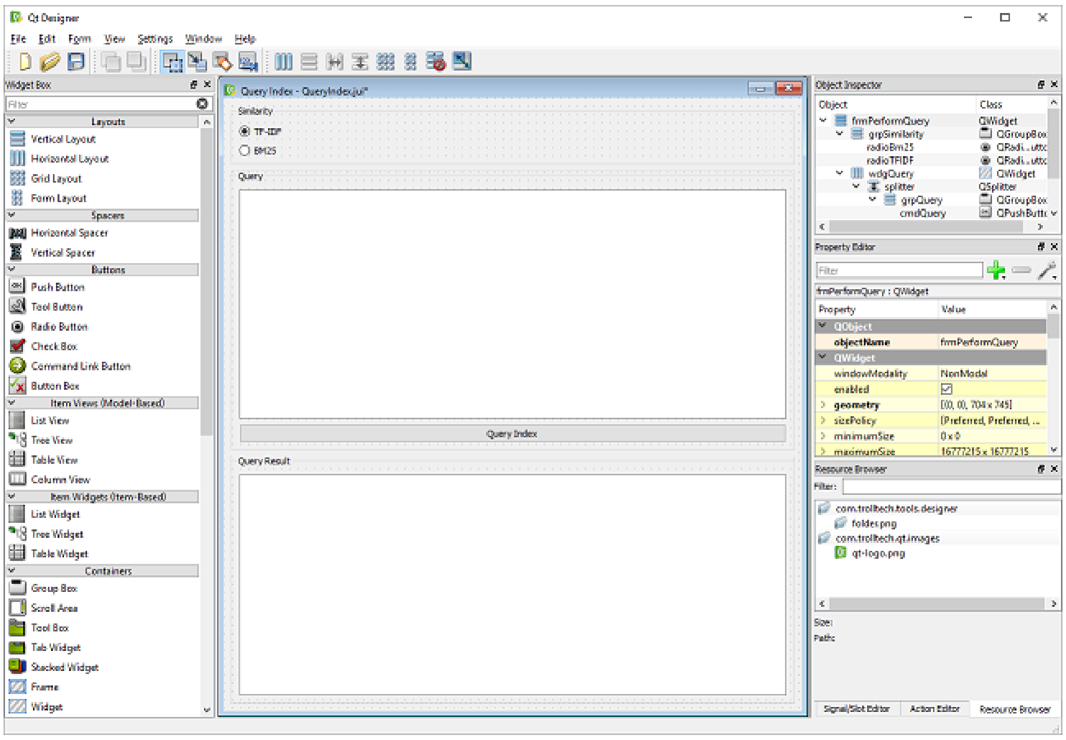
In this chapter, we provide a closer look on the implementation side of the recommender system we develop within the scope of this thesis. A general description of the system is already covered in chapter 3. Here, we discuss more specific information such as software versions, parameter choices, query types, algorithms, etc..

4.1. Programming Language and Frameworks

In Section 3.2 we mentioned that we use the Apache Lucene programming library for tasks like indexing and recommendation generation. Since this library is written purely in Java, we choose to implement our recommender system in Java as well. In particular, Java version 1.8 is our version of choice. For Apache Lucene we use version 6.5.0 which is the latest version available at the time when we start with the implementation of the system.

In Section 3.2, we also mention that in order to parse the articles of the data set, a special tool is used which requires the availability of a database. To meet this requirement, we installed a *MariaDB* [48] in version 10.0.28, which is the latest version of this database at the time we start with the parsing procedure.

For the creation of a graphical user interface (GUI) we use *Qt Jambi* [57]. Qt Jambi is a Java binding of the Qt [78] framework. As Qt is a cross-platform framework, it allows us to run the GUI of our system on a variety of platforms such as Windows, Linux and Mac. Qt Jambi comes with additional software such as *Qt Designer*. Qt Designer allows users to design and build GUIs using Qt Widgets as illustrated in Figure 4.1. These GUIs, when saved, are stored as XML files and are handed over to the Java User



Object	Class
frmPerformQuery	QWidget
grpSimilarity	QGroupBox
radioBm25	QRadi...utto
radioTFIDF	QRadi...utto
wdgQuery	QWidget
splitter	QSplitter
grpQuery	QGroupBox
cmdQuery	QPushButto

frmPerformQuery : QWidget

Property	Value
QObject	
objectName	frmPerformQuery
QWidget	
windowModality	NonModal
enabled	☑
geometry	[(0, 0), 704 x 745]
sizePolicy	[Preferred, Preferred, ...
minimumSize	0 x 0
maximumSize	16777215 x 16777215

Figure 4.1.: Designing our system interface in Qt Designer.

Interface Compiler (JUIC). This program is responsible for converting the XML files to the corresponding Java source files.

4.2. Indexing

In this section, we take a closer look at the field definitions discussed in Section 3.2 as well as the exact procedure of adding documents to the index.

Let's recall the required parts of an Apache Lucene field from Section 3.2:

> "[...] Each field has three parts: name, type and value. Values may be text (String, Reader or pre-analyzed TokenStream), binary (byte[]), or numeric (a Number). [...]"

How we meet these requirements for our predefined fields is illustrated in the overview provided by Table 4.1. When it comes to defining an index in

Table 4.1.: Field properties of the index.

Fields			
Name	**Type**	**Value**	**Store in Index**
ArticleID	IntField	Numeric	YES
Title	TextField	String	YES
Authors	TextField	String	YES
Publication_Day	IntField	Numeric	YES
Publication_Month	IntField	Numeric	YES
Publication_Year	IntField	Numeric	YES
Abstract	TextField	String	YES
Body	TextField	String	YES
Keywords	TextField	String	YES
References	TextField	String	YES

Apache Lucene, not only the structure of the index has to be considered but also the so called *Analyzer* [6] that is used.

The role of the analyzer is to analyze every input prior to indexing and, if necessary, transform or even remove that input. A common transformation is to automatically transform everything to lower case letters. This has the advantage that there can not be any duplicated words only distinguished by the case of their letters.

Another useful transformation is to perform stemming on each word which further prevents the index from having redundant entries. A case where the complete removal of a word can be useful is when said word is a stop-word. These and other transformations are discussed in Section 2.2.8.

Apache Lucene comes with various predefined analyzers for different languages and purposes. The analyzer we use in our system is the standard *EnglishAnalyzer* [7]. According to its implementation, it automatically removes stop-words and performs stemming using the Porter stemmer algorithm [51]. Additionally, everything is transformed to lower case and possessives (trailing 's) are removed from all words.

From a programming point of view, we can divide our indexing procedure into the following steps.

1. At the beginning, we create an Apache Lucene *IndexWriter* [10] object and configure it to use the *EnglishAnalyzer* as its default analyzer. The purpose of the IndexWriter is to maintain and create an index. It takes Apache Lucene Document objects as input, processes them and adds them to the index.
2. After that, we iterate over all articles contained in *Articles.XML* and *ExtendedArticles.XML* and create Apache Lucene Document objects along the way. All these Document objects are provided with the field definitions listed in Table 4.1 and all fields are initialized with the corresponding information contained in *Articles.XML* and *ExtendedArticles.XML*.
3. Finally, each Apache Lucene Document object is passed to the IndexWriter object. This causes the Document objects to be added to the index.

In order to make the field initialization during step 2 as easy as possible, we use the Java Architecture for XML Binding (JAXB) framework [77]. This framework provides a way to establish a mapping between XML documents and Java classes.

In our case, the idea is to access information contained in *Articles.XML* and *ExtendedArticles.XML* via simple operations on Java classes. A useful tool provided by this framework is the JAXB Binding Compiler (XJC). This tool is capable of translating XML schemas or Document Type Definitions (DTD) to fully annotated Java classes.

We take advantage of this feature by compiling the underlying DTDs of *Articles.XML* and *ExtendedArticles.XML* into their corresponding Java classes (refer to Section 3.2 on how these DTDs are composed).

With these classes in place, a so called *Unmarshaller*, provided by the JAXB framework, is used to read the contents of the XML files at run-time and automatically instantiate the necessary classes providing access to the available information. From that moment on, everything contained in the XML files can be accessed in an object oriented way.

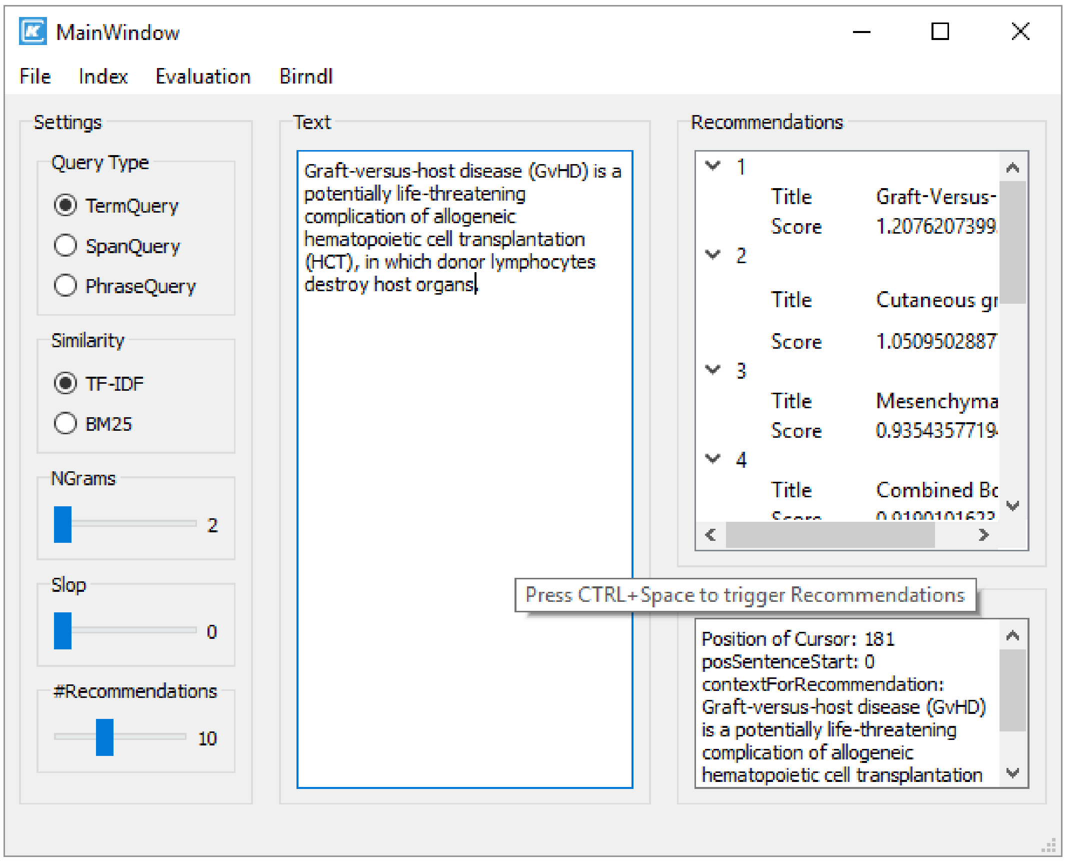
To further clarify all the steps taken over the course of the index creation, refer to the pseudocode provided by Algorithm 1.

Algorithm 1: Index creation

```
Input: Articles.XML file $f_a$, ExtendedArticles.XML $f_e$
1: Set $A \leftarrow$ new EnglishAnalyzer()
2: Set $W \leftarrow$ new IndexWriter($A$)
3: Set $U \leftarrow$ new Unmarshaller($f_a, f_e$)
4: while $U.hasNext()$ do
5:     Set $R \leftarrow U.next()$
6:     Set $D \leftarrow$ new Document()
7:     foreach Field $f$ specified in Table 4.1 do
8:         Set $n \leftarrow$ Name of Field$f$
9:         Set $v \leftarrow R.getValueForField(n)$
10:        Set $F \leftarrow$ new Field($n, v$)
11:        $D.add(F)$
12:    end
13:    $W.addDocument(D)$
14: end
15: $W.commit()$
```

4.3. Recommendation Generation

In Section 3.3, we mentioned that we use the citation context of a pending citation to formulate a query. This query is then handed over to the index who processes the query and performs an information retrieval task in order to return the top n ranked articles. In our system we choose the citation context to consist of the whole sentence of where the citation should be placed. The default value for n in our system is 10, however, this value can also be adjusted at run-time by the user. Figure 4.2 shows a screen shot of the main view of our system.



Query Type: [✓ TermQuery] SpanQuery PhraseQuery

Similarity: [✓ TF-IDF] BM25

NGrams: 2

Slop: 0

#Recommendations: 10

Graft-versus-host disease (GvHD) is a potentially life-threatening complication of allogeneic hematopoietic cell transplantation (HCT), in which donor lymphocytes destroy host organs.

1		
	Title	Graft-Versus-
	Score	1.2076207399
2		
	Title	Cutaneous gr
	Score	1.0509502887
3		
	Title	Mesenchyma
	Score	0.9354357719
4		
	Title	Combined Bc

```
Position of Cursor: 181
posSentenceStart: 0
contextForRecommendation:
Graft-versus-host disease (GvHD)
is a potentially life-threatening
complication of allogeneic
hematopoietic cell transplantation
```

Figure 4.2.: The main view of our system prototype. Via the controls on the left, users can switch between different query types used during the recommendation process of the system. Additionally, different query parameters can be supplied and the scoring model can be chosen. The middle part comprises the input area. This area is where users write their text and where they can request recommendations for a citation they want to make. The area on the right side shows the retrieved recommendations as well as some debug information.

4.3.1. Query Formulation and Document Retrieval

As soon as we have determined the citation context, we begin with the query formulation task. To complete this task, we make use of the classic Apache Lucene *QueryParser* [13] object. The purpose of this object is to take a query string as input and return an Apache Lucene *Query* [12] object. The

Algorithm 2: Query formulation and document retrieval

```
   Input: Citation context $C$
   Result: Documents $D$ most relevant to $ctx$
1: Set $A \leftarrow$ new EnglishAnalyzer()
2: Set $QP \leftarrow$ new QueryParser(A)
3: Set $Q \leftarrow QP.parse(ctx)$
4: Set $S \leftarrow$ new IndexSearcher()
5: Set $Z \leftarrow$ Similarity.BM25 (or any other supported model)
6: $S.setSimilarity(Z)$
7: Set $D \leftarrow S.search(Q)$
8: return $D$
```

query object is then passed to an Apache Lucene *IndexSearcher* [9] instance which performs the actual document retrieval.

Thanks to Apache Lucene's support for pluggable ranking models, it is possible to specify the model to use for this retrieval at run-time. Among the natively supported models in Apache Lucene there is the vector space model (covered in Section 2.2.5) and the BM25 model (covered in Section 2.2.6). After the IndexSearcher has done its job, the retrieved documents with the highest rank are shown to the user.

One important aspect when it comes to using the QueryParser object is that we also have to provide an analyzer to the constructor of the object. Recall the analyzer used Section 4.2 for creating the index. We use the same analyzer (which is the EnglishAnalyzer) here in order to stay consistent. The reason behind this is because otherwise keywords in the query would potentially undergo different transformations than keywords in the index; and this would most certainly have a negative impact on the retrieval performance.

To further clarify all these steps, refer to the pseudocode provided by Algorithm 2.

5. System Evaluation

This chapter is concerned with the experiments we conduct in order to measure the performance of our system. Within our experiments, we tweak various parameters related to the retrieval of documents and then use these settings to gather a set of documents D returned by the index after executing a query q. How we accomplish this, what parameters we tweak, and what the results are is explained in the following sections.

5.1. Experiments

For all of our experiments we make use of the *Citations.XML* file described in Section 3.2. This file contains every citation made within all articles in our index as well as the corresponding citation contexts. Having this information allows us to take the following approach:

1. At first, we open the *Citations.XML* file and iterate over every citation available. For each citation, we are interested in the corresponding citation context, the ID of the source article, and the ID of the referenced article.
2. Once we have this information, we formulate a query based on the citation context and gather results using a similar method as to what is shown in Algorithm 2. During this step, we also tweak various parameters such as which query type to use or under which scoring model the ranking should take place.
3. After we gather the results, we use a common evaluation metric for information retrieval systems to measure the performance.

The pseudocode provided by Algorithm 3 gives an idea how the approach taken above is realized within our system. On Line 11 of the algorithm, all

Algorithm 3: Run experiment and gather result

```
    Input: Citations.XML file $f_c$, QueryFormulationStrategy $T$,
           Similarity $Z$
    Data: Citation $C$
    Result: List $L$ containing Lists of documents $D$
 1: Set $U \leftarrow$ new Unmarshaller$(f_c)$
 2: Set $A \leftarrow$ new EnglishAnalyzer$()$
 3: Set $QP \leftarrow$ new QueryParser$(\mathrm{A})$
 4: Set $S \leftarrow$ new IndexSearcher$()$
 5: $S.setSimilarity(Z)$
 6: Set $L \leftarrow \{\}$
 7: while $U.hasNext()$ do
 8:     Set $\mathrm{C} \leftarrow U.next()$
 9:     Set $ctx \leftarrow C.getCitationContext()$
10:     Set $Q \leftarrow QP.parse(ctx)$
11:     Set $K_Q \leftarrow Q.getKeywords()$
12:     Set $Q1 \leftarrow T.formulateQuery(K_Q)$
13:     Set $D \leftarrow S.search(Q1)$
14:     $L.add([D, C])$
15: end
16: return $L$
```

keywords contained in the parsed query are extracted. On Line 12, a new query is created from these keywords using the QueryFormulationStrategy object passed as an argument to the algorithm. The role of this object is to build a query of a particular type and with particular parameters. Also, note the argument Z being passed to the algorithm. This argument determines the scoring model according to which the articles should be ranked.

Table 5.1 lists the different combinations of query types, parameters, and models that we run our experiments with.

All of our test are conducted under the BM25 model as well as the vector space model. Regarding the BM25 model, we use the parameters $b = 0.75$ and $K_1 = 1.2$ (see Section 2.2.6 about the BM25 model to find the purpose of these parameters). These are the default values in Apache Lucene's BM25

Table 5.1.: Query formulation strategies and models.

Query type	Parameters		Model
	n-grams	Slop	
TermQuery	-	-	BM25, vector space
PhraseQuery	2,3,4,5	-	BM25, vector space
SpanOrQuery	-	10,25,50	BM25, vector space

implementation [2].

When it comes to TF-IDF weighting under the vector space model (see Section 2.2.5), we use the default implementation provided by Apache Lucene's *ClassicSimilarity* class [5], that is, the document TF weight tf_{ij} for a term k_i in a document d_j is determined by

$$tf_{ij} = \sqrt{f_{ij}}$$

where f_{ij} represents the number of occurrences of term k_i in document d_j. As a query TF weight, a constant 1 is used. The IDF weight idf_i for a term k_i contained in a document or query, is determined by

$$idf_i = 1 + \log\left(\frac{N+1}{n_i+1}\right)$$

where N is the total number of documents in the index and n_i is the number of documents containing the term k_i. When it comes to length normalization, the adopted strategy is to normalize via a factor of

$$lengthNorm = \frac{1}{\sqrt{\text{number of terms}}}$$

The different query types we use are provided by Apache Lucene and comprise TermQuery [16], SpanOrQuery [15] and PhraseQuery [11]. We use these queries in combination with different parameters as described below:

TermQuery This query type matches documents containing the term associated with the query. This is the simplest form of query. In our tests, we extract all terms from the citation context, create a TermQuery instance for each one, and combine them to a BooleanQuery [3] via OR-conjunctions. Using OR-conjunctions allows us to link multiple terms and find a matching document if any of the terms exist in the document.

PhraseQuery This query type matches documents containing a particular sequence of terms, e.g. for the term sequence "information retrieval system" only documents containing this exact sequence are matched. With the *n-grams* parameter, we specify how long these term sequences should be, e.g. a value of 2 means that we form sequences of length 2. In our tests, we extract word sequences from the citation context and combine them via OR-conjunctions to form a BooleanQuery.

SpanOrQuery With this query, we combine multiple SpanNearQuery [14] instances via OR-conjunctions. A SpanNearQuery is used to match documents where terms occur within a certain distance of each other. The *slop* parameter is used to specify this distance. In our tests, we extract the terms from the citation context and set them up as SpanNearQuery instances. These are then combined to a SpanOrQuery [15].

5.2. Results

The main performance metric we use to evaluate the results gathered through our experiments is known as *mean reciprocal rank (MRR)* [79]. The reciprocal rank of a query response is defined by the reciprocal value of the rank of the first correct answer as illustrated in Table 5.2.

Table 5.2.: Reciprocal rank calculation illustrated on a set of exemplary queries and their query responses.

Query	Query response	Answer	Rank	Recipr. rank
search	engine, algorithm, term	engine	1	1
retrieval	system, model, algorithm	model	2	1/2
system	id, details, evaluation	evaluation	3	1/3

Algorithm 4: Reciprocal rank calculation

```
    Input: List of Document objects $D$, Citation object $C$
    Result: The reciprocal rank $RR$
1:  Set $RR \leftarrow 0$
2:  Set $r \leftarrow 0$
3:  Set $i \leftarrow 1$
4:  for $d$ in $D$ do
5:      Set $D_{ID} \leftarrow d.getID()$
6:      Set $C_{ID} \leftarrow C.getReferencedArticleId()$
7:      if $D_{ID} == C_{ID}$ then
8:          Set $r \leftarrow i$
9:          break
10:     end
11:     Set $i \leftarrow i + 1$
12: end
13: if $r \neq 0$ then
14:     Set $RR \leftarrow 1/r$
15: end
16: return $RR$
```

Given this definition, the MRR can be introduced as the average over all reciprocal ranks for a set of queries Q. Formally, this means:

$$MRR = \frac{1}{|Q|} \sum_{q \in Q} RR(q)$$

where $RR(q)$ refers to the reciprocal rank associated with query q and $|Q|$ is the total number of queries.

Typically, the MRR is used when we are interested in the first correct answer to a query [17]. A good example for where this makes sense are web queries where the users specify a reference to a website and are interested in the first correct answer. Another example would be question answering systems whose goal is to provide a correct answer to a question (instead of a ranked answer set). But, also our task of finding a correct source to a pending citation can be placed in this category, as we are interested in the first correct source. Since the MRR is a metric that favors results where the correct answer is highly ranked, it is well suited for these cases and is why we choose it for our evaluation.

Algorithm 5: Mean reciprocal rank calculation

```
    Input: List of experimental results $L$
    Data: IndexReader $X$
    Result: The mean reciprocal rank $MRR$
1:  Set $MRR \leftarrow 0$
2:  for $l$ in $L$ do
3:      Set $D \leftarrow l.getDocuments()$
4:      Set $C \leftarrow l.getCitation()$
5:      if $(X.getDocument(C.getReferencedArticleId())$ then
6:          Set $MRR \leftarrow MRR + RR(D, C)$
7:      end
8:  end
9:  if $L.size() > 0$ then
10:     Set $MRR \leftarrow MRR / L.size()$
11: end
12: return $MRR$
```

Recall the list L we compute with Algorithm 2. This list comprises elements for each query, containing the query response (the list of document objects D in the algorithm) and the correct answer (the citation object C in the algorithm). Every document d in D has a unique ID and is sorted according to its score. Every citation object C has a property named *referencedArticleId*. Thus, the reciprocal rank can be calculated as shown in Algorithm 4.

Calling this algorithm on each element in L and dividing by $|L|$ after summing up all the results yields the desired MRR. This procedure is also shown in Algorithm 5.

Note that not every article that a citation expects is available in our data set. In this case, the reciprocal rank calculation is simply skipped as reflected by line 5 in Algorithm 5. Overall, for 106671 citations, the corresponding article is in our data set.

Also, regarding the size of the list of documents D in the algorithms shown above: We have chosen a value of 100 for the size of this list within all of our experiments. This means, if the correct article is not within this list then it is treated as having a reciprocal rank of 0.

5.2.1. Vector Space Model

Within our experiments under the vector space model, we investigate the impact of varying the query parameters listed in Table 5.1 on the performance of the system.

PhraseQuery Our test results regarding varying the *n-grams* parameter show that a lower value generates better results as illustrated in Figure 5.1. An explanation for this behavior can be that a lower value implies smaller

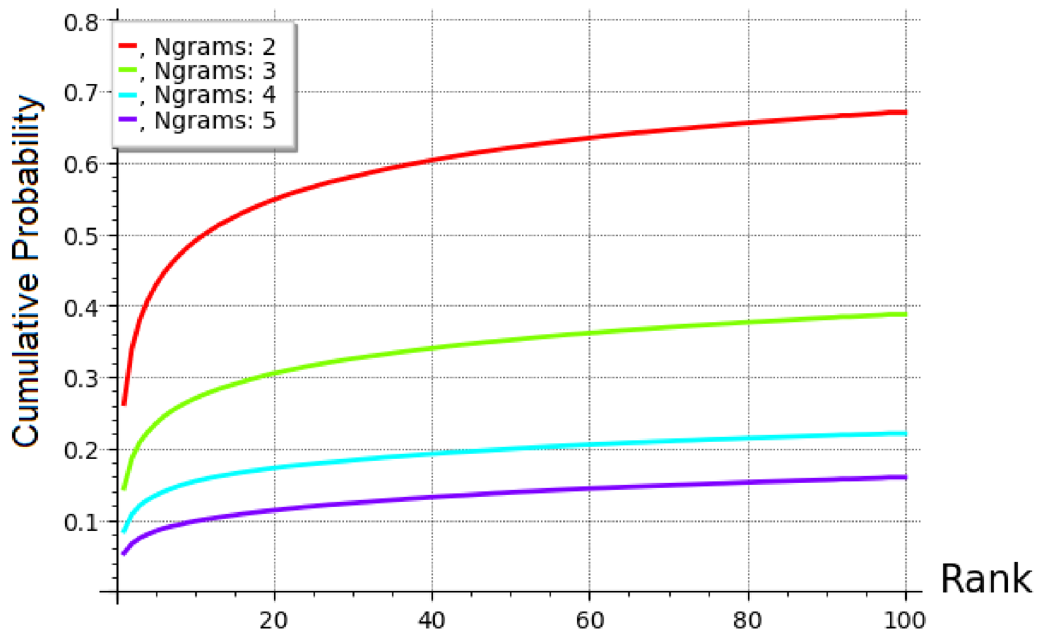


Figure 5.1.: The cumulative rank distribution for varying the *n-grams* parameter in PhraseQuery instances in the vector space model. For each rank on the x-axis, this diagram shows the cumulative probability that the expected document is found at this rank.

word groups, and this is less restrictive when it comes to finding documents that match these word groups. The respective MRRs to these test runs are shown in Table 5.3.

Table 5.3.: The MRRs for tests with PhraseQuery instances in the vector space model.

n-grams	MRR
2	0.341
3	0.188
4	0.108
5	0.069

SpanOrQuery Our test results regarding the variation of the *slop* parameter show that a higher value generates better results as illustrated in Figure 5.2. An explanation for this behavior can be that a higher value implies larger text span coverage, thus, the chance for a document to match the query is higher. The respective MRRs to these test runs are shown in Table 5.4.

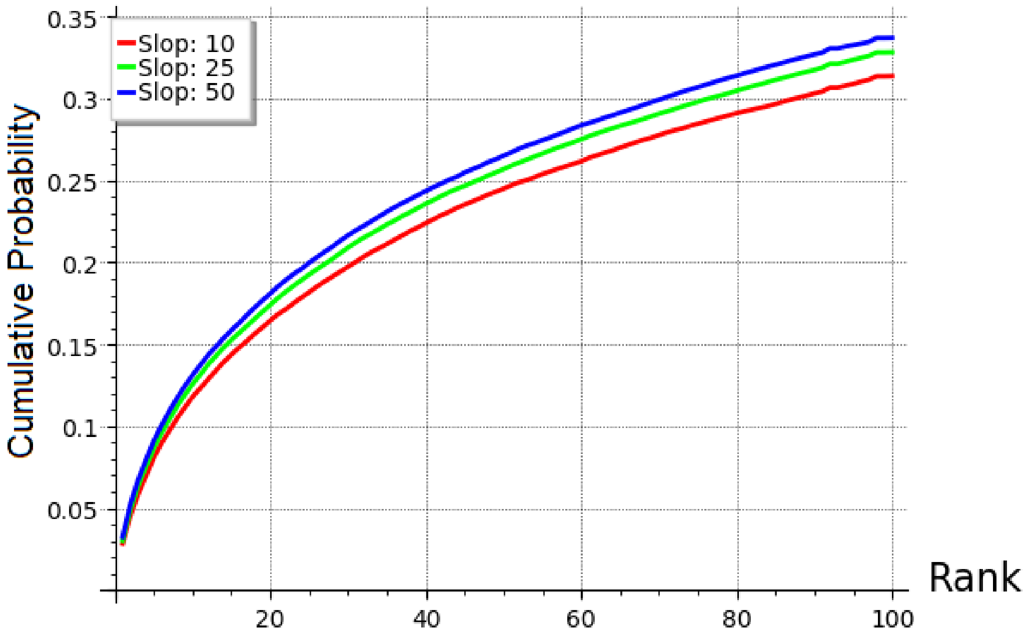


Figure 5.2.: The cumulative rank distribution for varying the *slop* parameter in SpanNearQuery instances in the vector space model. For each rank on the x-axis, this diagram shows the cumulative probability that the expected document is found at this rank.

Table 5.4.: The MRRs for tests with SpanOrQuery instances in the vector space model.

Slop	MRR
10	0.058
25	0.062
50	0.065

TermQuery Our test using TermQuery instances yields the best result as illustrated in Figure 5.3. The respective MRR is 0.416.

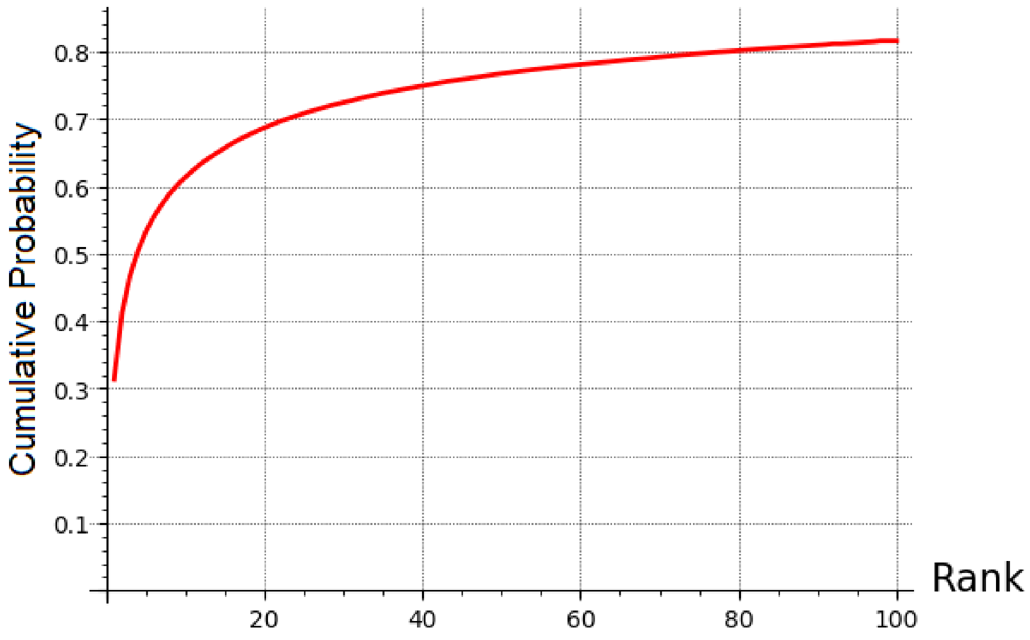


Figure 5.3.: The cumulative rank distribution for our test with TermQuery instances in the vector space model. For each rank on the x-axis, this diagram shows the cumulative probability that the expected document is found at this rank.

5.2.2. BM25

Within our experiments under the BM25 model, we investigate the impact of varying the query parameters listed in Table 5.1 on the performance of the system.

PhraseQueries Same as for the vector space model, our test results regarding the variation of the *n-grams* parameter show that a lower value generates better results as illustrated in Figure 5.4. Judging by the rank

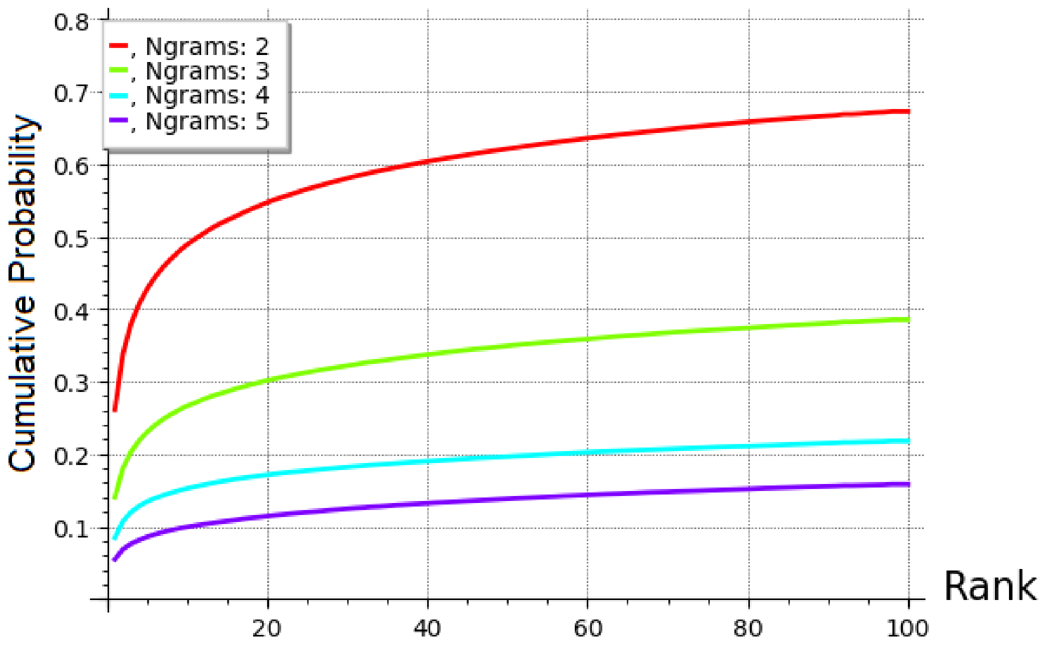


Figure 5.4.: The cumulative rank distribution for varying the *n-grams* parameter in PhraseQuery instances in the BM25 model. For each rank on the x-axis, this diagram shows the cumulative probability that the expected document is found at this rank.

distribution, performance seems very similar to the vector space model, this is also reflected by the MRRs as shown in Table 5.5.

SpanOrQuery Like in the vector space model, our test results regarding the variation of the *slop* parameter show that a higher value generates better results as illustrated in Figure 5.5. Based on the rank distribution,

Table 5.5.: The MRRs for tests with PhraseQuery instances in the BM25 model.

n-grams	MRR
2	0.340
3	0.183
4	0.108
5	0.070

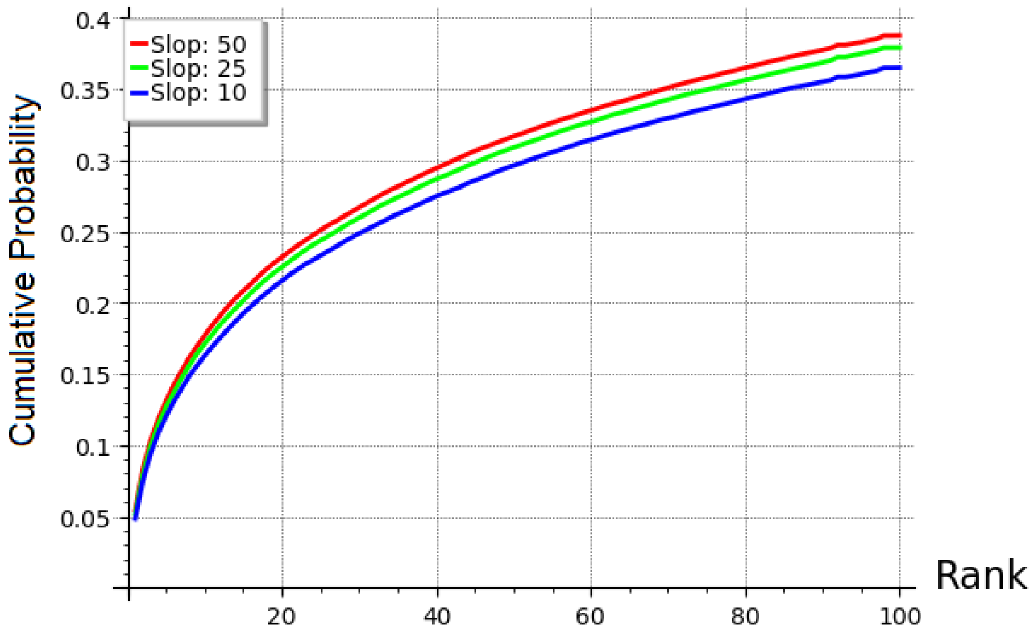


Figure 5.5.: The cumulative rank distribution for varying the *slop* parameter in SpanNearQuery instances in the BM25 model. For each rank on the x-axis, this diagram shows the cumulative probability that the expected document is found at this rank.

performance is slightly better than in the vector space model, this is also reflected by the MRRs as shown in Table 5.6.

TermQuery Our test results regarding TermQuery instances yields the result illustrated in Figure 5.6. The respective MRR is given as 0.415.

Table 5.6.: The MRRs for tests with SpanOrQuery instances in the BM25 model.

Slop	MRR
10	0.087
25	0.092
50	0.096

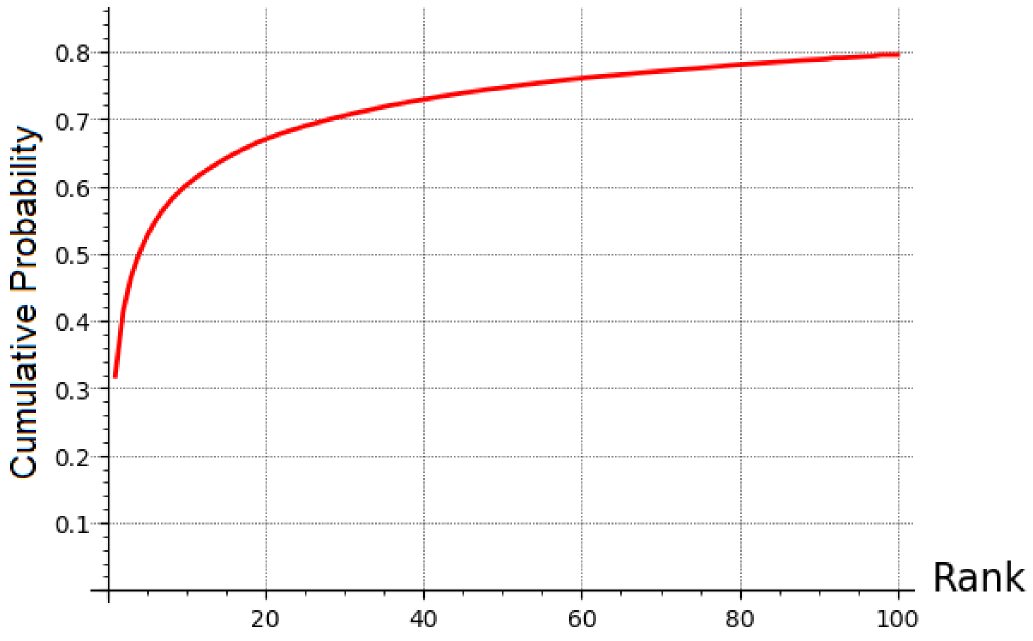


Figure 5.6.: The cumulative rank distribution for our test with TermQuery instances in the BM25 model. For each rank on the x-axis, this diagram shows the cumulative probability that the expected document is found at this rank.

5.3. Role of the Citation Context

In this section, we investigate how the size of the citation context influences the recommendation performance. In order to do so, we re-run our TermQuery approach under the vector space model from the previous section on a subset of 1000 citations, while setting a limit on the size of the citation context.

The size of the citation context is determined by how many words it comprises. In this experiment, we vary the size of the citation context from 1 to 300 words and measure the resulting MRR. The result of this experiment is illustrated in Figure 5.7.

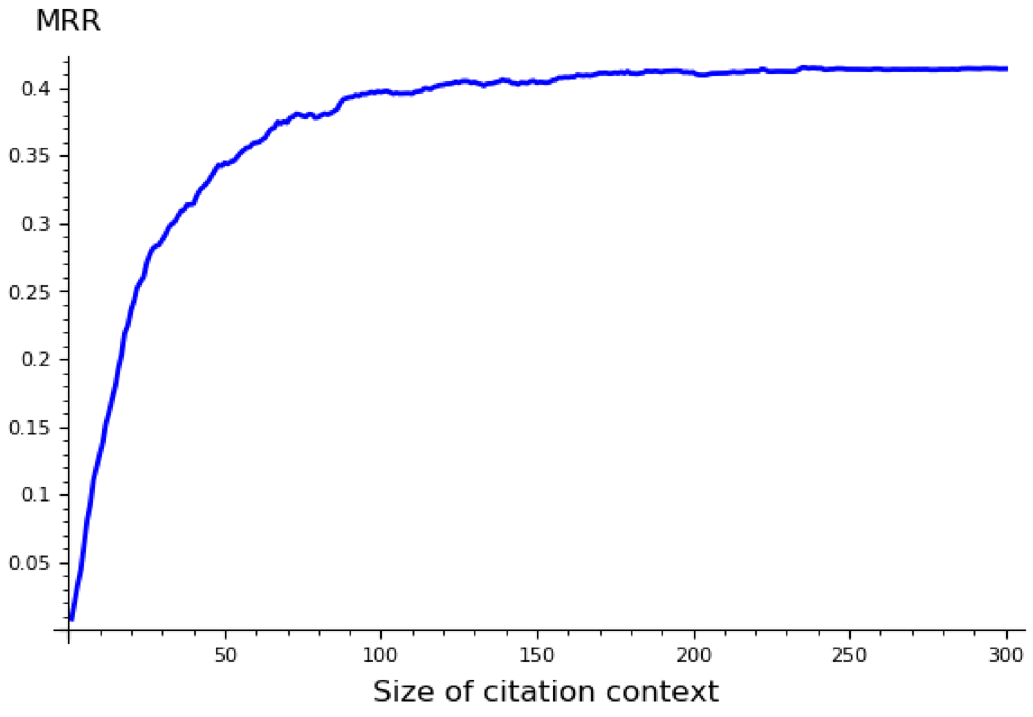


Figure 5.7.: Recommendation performance measured as MRR using the TermQuery approach under the vector space model for differently sized citation contexts.

By default, our citation context is read from the *Citations.XML* file and consists of the full text paragraph of where the citation occurs in (see description of the *Citations.XML* file in Section 3.2). Now, if the size of this citation context is larger than the set limit, we alternately remove words from left and right until the set limit is reached. In the case that the size is already smaller than the limit, no modification is performed.

Judging by the resulting MRRs, we find that the MRR drops significantly if the citation context is chosen too small (e.g. only a few words). On the other hand, after the size of the citation context reaches a certain threshold, it only induces minor alterations to the MRR.

In the results shown in Figure 5.7, the MRRs between the sizes 120 to 300 of the citation context are in the interval $[0.401, 0.415]$. In contrast, the MRRs between the sizes 1 to 75 of the citation context are in the interval $[0.008, 0.380]$. The MRRs between the sizes 76 to 119 of the citation context are in the interval $[0.378, 0.401]$.

Within the scope of our data set, a citation context of, for instance, 120 words seems to work well. The mean size of the citation context in our *Citations.XML* file is 182 words, with a standard deviation of 101 words. With this characteristics, we obtain a MRR of 0.416 as mentioned in the previous section.

5.4. Conclusion

Judging by the results of our test runs, the vector space model and BM25 model show very similar performance. In terms of generating queries, simple OR-conjuncted TermQuery instances in the form of a BooleanQuery seem to work best and OR-conjuncted SpanNearQuery instances in the form of a SpanOrQuery seem to work worst.

Looking at the cumulative rank distribution regarding our TermQuery approach, we observe that in about 60 out of 100 cases, the expected article is present within the first 10 hits. If we consider a list of 20 hits, around 68 out of 100 times, the expected article is found. Going up to 100 hits, in approximately 82 out of 100 cases, the expected article is available. In around 18 out of 100 cases, the expected article is not contained in the top 100.

Regarding the cumulative rank distribution of the best PhraseQuery approach, we observe that in about 48 out of 100 cases, the expected article is within the first 10 hits. Inspecting the first 20 hits, in roughly 55 out of 100 cases, the expected article is present. Considering a list of 100 hits, in about 67 out of 100 times, the expected article is found. In around 33 out of 100 cases, the expected article is not found within the top 100.

The cumulative rank distribution of the SpanQuery approach shows that approximately in 16 out of 100 cases, the expected article is found within the first 10 hits. If we consider a list of 20 hits, around 22 out of 100 times, the expected article is found. In a list of 100 hits, in roughly 36 out of 100 times, the expected article is found. In about 64 out of 100 cases, the expected article is not found within the top 100.

Overall, the TermQuery approach works best and shows a clear performance margin over the other approaches. The potential of this method shows itself by an almost 70% chance of finding the expected article within the top 20 retrieved documents. By introducing further techniques such as query boosting [4] and/or optimizing parameters in the ranking models themselves, this value can most certainly be improved even more.

A summary of the best MRRs amongst our query formulation strategies is shown in Table 5.7. A visual comparison is provided by Figure 5.8.

Table 5.7.: The highest MRR for each query formulation strategy.

Query type	Query parameters		Model	MRR
	n-grams	Slop		
TermQuery	-	-	VSM	0.416
PhraseQuery	2	-	VSM	0.341
SpanOrQuery	-	50	BM25	0.096

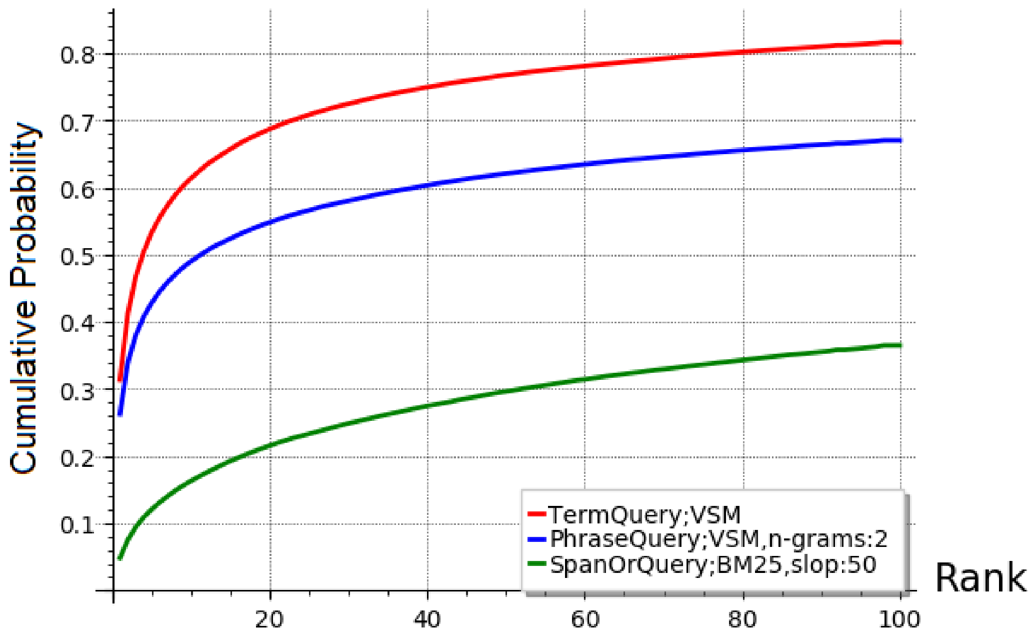


Figure 5.8.: The cumulative rank distributions of our query formulation strategies, using the parameters that worked best during our experiments. For each rank on the x-axis, this diagram shows the cumulative probability that the expected document is found at this rank.

6. Related Work

In this chapter, we discuss related work conducted on the recommendation of literature within the context of scientific writing.

He et al. [29] develop a prototype system running on the CiteSeerX [28] digital library. This system is capable of recommending citations for citation placeholders placed in a query manuscript. Within their approach, they consider the snippet of text around a placeholder (the citation context), preprocess it, and based on that, they recommend relevant documents. For the citation context, they consider a fixed size window of 50 words before and 50 words after the placeholder, as suggested by the extensive studies of Ritchie [61] on the impact of different lengths of citation contexts. After extracting the citation context, they remove all the stop words. To measure the relevance between a document and a preprocessed citation context, they use their implementation of a novel non-parametric probabilistic model. Over the course of thorough empirical evaluation, they show the effectiveness and scalability of their approach.

A set of research papers can be seen as a network: Papers reflect outgoing connections by citing other papers, and incoming connections by being cited by other papers. All these connections together form a network graph known as the *citation web*. McNee et al. [49] use the information provided by the citation web to set up a collaborative filtering (CF) framework. This framework they then use to find citations that would suit well as additional references in a target paper. After a series of experiments with different algorithms, they find that the generated recommendations are either very relevant or very novel. None of the algorithms they study, however, provides both at the same time.

Strohman et al. [75] show their model to a content-based system. The use case of their system is targeted to users who have already written several pages about a topic and then proceed to hand over the written text to the system with the goal of retrieving documents that the provided text might cite. They show that text features alone are not sufficient for this task and that additional graph-based features are required in order to retrieve high quality results. Via linear combinations of text features and citation graph features, they measure the relevance between queries and documents. After evaluating their system, they conclude that the similarity between the query and candidate documents as well as the Katz distance [39] within the retrieved set of documents are the two most significant features in this task.

Also, translation models are used to solve the problem of citation recommendation. Originally used to translate text from one language to another, Berger et el. [20] introduce the idea behind translation models to information retrieval by translating words in documents to query terms in order to overcome vocabulary gaps between queries and documents. In the work conducted by Lu et al. [44], they propose a translation model for citation recommendation. Within their experimental setup, they use the citation context around citation placeholders to retrieve a set of papers that may be referenced at the particular citation placeholder. Based on their results, they observe that building the translation model upon the abstract of a document performs better than building the model upon the full text.

Basu et al. [18] show their approach on recommending conference paper submissions to reviewers. To tackle this problem, they create a user profile for each reviewer where the purpose of these profiles is to capture and reflect the reviewers' interests. The data to create these profiles is automatically extracted from the Web. Based on the abstract of conference paper submissions and these profiles, they implement a recommendation framework to route papers to reviewers. From the evaluation of their system, they conclude that within the context of peer reviewing, they can make the recommendation process less "people intensive". Furthermore, they observe that their content-based algorithms can outperform their collaborative ones in this task.

Finding relevant scholarly literature is also a key point of the workshop on Bibliometric-enhanced Information Retrieval and Natural Language Processing for Digital Libraries (BIRNDL) [21, 23, 24]. As a part of these workshops, Computational Linguistics (CL) Scientific Summarization Shared Tasks (CL-SciSumm Shared Tasks) [33, 32, 35] take place. Having participated in the CL-SciSumm Shared Task 2017, we take a closer look at results and key insights gained by this task in the following section.

6.1. The CL-SciSumm Shared Task 2017

In this section, we discuss the CL-SciSumm Shared Task 2017 [32] and its results and key insights [34]. The CL-SciSumm Shared Task 2017 poses three problems: (1A) finding relevant text spans in a target document given a citation context in a source document, (1B) classifying the discourse facets for target documents found in (1A), and (2, optional) generating a summary for target documents by their identified text spans.

Via our own submission [1] [27] to the CL-SciSumm Shared Task 2017, we provide a solution to tasks (1A) and (1B) by translating the concepts used in this thesis to the scope of these tasks.

6.1.1. Participating Systems

Overall, nine systems [45, 53, 41, 42, 1, 27, 37, 80, 52] participated in tasks 1A and 1B. Beginning from support vector machines, over to neural network based ranking models, and up to variants of tf-idf scoring, all sorts of different approaches were employed by these systems, some of which are briefly summarized below:

The University of Mannheim [41], for example, employed a supervised learning approach to solve Task 1A. To rank the results, they considered features such as lexical similarity, entity similarity, semantic similarity, and others. To solve Task 1B, they experimented with support vector machines and trained a convolutional neural network to perform the classification task.

[1] See Appendix A for the system report to our submission

The National University of Singapor [53] solved Task 1A by using a combined scoring mechanism, where they considered TF-IDF, longest common subsequence (LCS), and semantic relations. To determine the semantic relations, they used a neural network based ranking model. For the classification in Task 1B, they also used a neural network based approach.

The Beijing University of Posts and Telecommunications [42] computed various similarity metrics such as Jaccard similarity, context similarity, and IDF similarity to solve Task 1A. By combining these similarity scores using approaches such as support vector machines, fusion method, or Jaccard Cascade, they determined the final ranking. For Task 1B, they took an approach using support vector machines.

The Nanjing University of Science and Technology [45] treated Task 1A as a classification problem. They used different classifiers such as two different support vector machines, a decision tree, and a logistic regression. By using a voting based system, they then combined the results of these classifiers. To solve Task 1B, they introduced a dictionary for each discourse facet. A reference span was labeled with a facet if its dictionary comprised any of the words present in the reference span.

6.1.2. Evaluation and Results

The evaluation for task 1A was done by comparing the sentence (text span) overlaps between sentences found by the system and a humanly created gold standard. Based on the number of overlaps, precision, recall and F_1 score [74] were calculated for each system. Furthermore, lexical overlaps using variants of the ROUGE score [43] were calculated.

For Task 1B, under the condition that the response of task 1A was as expected, the proportion of the correctly classified discourse facets serves as base for the evaluation. Same as for task 1A, precision, recall and F_1 score were used to score this task.

Regarding our own contribution, a ROUGE score of 0.108 and second best overall result at Task 1A, closely behind the ROUGE score (0.114) of the winning system, shows the effectiveness of our approach in this task. A similar situation is present at Task 1B, where we generate a ROUGE score of 0.337, placing eighth among all the 47 submitted system runs.

A complete overview comprising all the results of all the different participating systems is available in [34].

7. Conclusion

With the rapidly growing amount of information in our time, it becomes more and more difficult to conduct any form of information related search. This circumstance is known as *information overload* and makes its presence felt in scenarios where, for example, someone who is writing a scientific paper has to conduct a literature search as she does not know a correct source for a citation she wants to make.

Over the course of this work, we show the implementation of a recommender system and demonstrate how we tackle this problem via the assistance of this system.

Within a general description of this system, we discuss about its various requirements and give insight to several concepts used during its implementation. We discuss the data set that we use and show how we process and use the data within the system.

Referring to the implementation of the system, we describe important technical details and provide relevant background information to the various technologies that we make use of. Among these technical details, we are mostly concerned with different query formulation strategies and information retrieval approaches.

To measure the performance of the system, we conduct a series of experiments. We describe how these experiments are set up and specify the active parameters for each one. Eventually, we gather the outcomes of these experiments and interpret the result.

Additionally, we investigate how the size of the citation context influences the recommendation performance within our top performing experiment and analyze the result. Based on this analysis, we find that the citation context should not be chosen too small (e.g. only a few words), otherwise the recommendation performance drops significantly. We also observe that if the size of the citation context reaches a certain threshold, only minor performance changes are measurable if the size is further increased.

The measured performance of the system reflects the potential of our approach, given that the correct parameters are chosen. In particular, our query formulation strategy using TermQuery instances performs well: At our best experimental results, our algorithm has a 60% chance that the expected document to cite is found within the first 10 recommended documents. Considering a list of 20 or 100 recommended documents, this chance even increases to 68% and 82% respectively. Based on these probabilities, our system is definitely helpful as there is a good chance that the expected document is found even within a small list of only 10 recommended documents. Following this observation, we can conclude that this approach offers a solid solution to our original problem, which also provides an answer to our research question where we were interested in how reliable content-based recommendations, serving as citation candidates for a pending citation, are.

Via our participation at the CL-SciSumm Shared Task 2017, we further investigate the capabilities of the system within the context of bibliometric-enhanced information retrieval and natural language processing for digital libraries. Yielding top results among all participating systems also shows the potential of our approach.

Bibliography

[1] Ahmed AbuRa'ed et al. "LaSTUS/TALN @ CLSciSumm-17: Cross-document Sentence Matching and Scientific Text Summarization Systems." In: *BIRNDL@SIGIR (2)*. Vol. 2002. CEUR Workshop Proceedings. CEUR-WS.org, 2017, pp. 55–66 (cit. on p. 71).

[2] *Apache Lucene Documentation - The BM25Similarity Object*. https://lucene.apache.org/core/6_5_0/core/org/apache/lucene/search/similarities/BM25Similarity.html (cit. on p. 53).

[3] *Apache Lucene Documentation - The BooleanQuery Object*. https://lucene.apache.org/core/6_5_0/core/org/apache/lucene/search/BooleanQuery.html (cit. on p. 54).

[4] *Apache Lucene Documentation - The BoostQuery Object*. https://lucene.apache.org/core/6_5_0/core/org/apache/lucene/search/BoostQuery.html (cit. on p. 66).

[5] *Apache Lucene Documentation - The ClassicSimilarity Object*. https://lucene.apache.org/core/6_5_0/core/org/apache/lucene/search/similarities/ClassicSimilarity.html (cit. on p. 53).

[6] *Apache Lucene Documentation - The Document Object*. https://lucene.apache.org/core/6_5_0/core/org/apache/lucene/document/Document.html (cit. on pp. 39, 45).

[7] *Apache Lucene Documentation - The EnglishAnalyzer Object*. https://lucene.apache.org/core/6_5_0/analyzers-common/org/apache/lucene/analysis/en/EnglishAnalyzer.html (cit. on p. 45).

[8] *Apache Lucene Documentation - The Field Object*. https://lucene.apache.org/core/6_5_0/core/org/apache/lucene/document/Field.html (cit. on p. 39).

[9] *Apache Lucene Documentation - The IndexSearcher Object*. https://lucene.apache.org/core/6_5_0/core/org/apache/lucene/search/IndexSearcher.html (cit. on p. 49).

[10] *Apache Lucene Documentation - The IndexWriter Object*. https://lucene.apache.org/core/6_5_0/core/org/apache/lucene/index/IndexWriter.html (cit. on p. 46).

[11] *Apache Lucene Documentation - The PhraseQuery Object*. https://lucene.apache.org/core/6_5_0/core/org/apache/lucene/search/PhraseQuery.html (cit. on p. 53).

[12] *Apache Lucene Documentation - The Query Object*. https://lucene.apache.org/core/6_5_0/core/org/apache/lucene/search/Query.html (cit. on p. 48).

[13] *Apache Lucene Documentation - The QueryParser Object*. https://lucene.apache.org/core/6_5_0/queryparser/org/apache/lucene/queryparser/classic/QueryParser.html (cit. on p. 48).

[14] *Apache Lucene Documentation - The SpanNearQuery Object*. https://lucene.apache.org/core/6_5_0/core/org/apache/lucene/search/spans/SpanNearQuery.html (cit. on p. 54).

[15] *Apache Lucene Documentation - The SpanOrQuery Object*. https://lucene.apache.org/core/6_5_0/core/org/apache/lucene/search/spans/SpanOrQuery.html (cit. on pp. 53, 54).

[16] *Apache Lucene Documentation - The TermQuery Object*. https://lucene.apache.org/core/6_5_0/core/org/apache/lucene/search/TermQuery.html (cit. on p. 53).

[17] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval*. 2nd. USA: Addison-Wesley Publishing Company, 2008. ISBN: 9780321416919 (cit. on pp. 12, 13, 15, 20, 23, 26, 27, 55).

[18] Chumki Basu et al. "Technical paper recommendation: A study in combining multiple information sources." In: *Journal of Artificial Intelligence Research* 14 (2001), pp. 231–252 (cit. on p. 70).

[19] Marta Vega Bayo. "Reference Recommendation for Scientific Articles." MA thesis. Graz: Graz University of Technology, 2016 (cit. on p. 37).

[20] Adam Berger and John Lafferty. "Information Retrieval As Statistical Translation." In: *Proceedings of the 22Nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '99. Berkeley, California, USA: ACM, 1999, pp. 222–229. ISBN: 1-58113-096-1. DOI: 10.1145/312624.312681. URL: http://doi.acm.org/10.1145/312624.312681 (cit. on p. 70).

[21] Guillaume Cabanac et al. "Joint Workshop on Bibliometric-enhanced Information Retrieval and Natural Language Processing for Digital Libraries (BIRNDL 2016)." In: *JCDL*. ACM, 2016, pp. 299–300 (cit. on p. 70).

[22] Stefano Ceri et al. *Web information retrieval*. Springer Science & Business Media, 2013 (cit. on pp. 28, 31).

[23] Muthu Kumar Chandrasekaran, Kokil Jaidka, and Philipp Mayr. "Joint Workshop on Bibliometric-enhanced Information Retrieval and Natural Language Processing for Digital Libraries (BIRNDL 2017)." In: *SIGIR*. ACM, 2017, pp. 1421–1422 (cit. on p. 70).

[24] Muthu Kumar Chandrasekaran, Kokil Jaidka, and Philipp Mayr. "Joint Workshop on Bibliometric-enhanced Information Retrieval and Natural Language Processing for Digital Libraries (BIRNDL 2018)." In: *SIGIR*. ACM, 2018, pp. 1415–1418 (cit. on p. 70).

[25] Lee R Dice. "Measures of the amount of ecologic association between species." In: *Ecology* 26.3 (1945), pp. 297–302 (cit. on p. 19).

[26] Michael D Ekstrand, John T Riedl, and Joseph A Konstan. "Collaborative filtering recommender systems." In: *Foundations and Trends in Human-Computer Interaction* 4.2 (2011), pp. 81–173 (cit. on p. 8).

[27] Thomas Felber and Roman Kern. "Graz University of Technology at CL-SciSumm 2017: Query Generation Strategies." In: *BIRNDL@SIGIR (2)*. Vol. 2002. CEUR Workshop Proceedings. CEUR-WS.org, 2017, pp. 67–72 (cit. on p. 71).

[28] C. Lee Giles, Kurt D. Bollacker, and Steve Lawrence. "Citeseer: an automatic citation indexing system." In: *INTERNATIONAL CONFERENCE ON DIGITAL LIBRARIES*. ACM Press, 1998, pp. 89–98 (cit. on p. 69).

[29] Qi He et al. “Context-aware citation recommendation.” In: *Proceedings of the 19th international conference on World wide web*. ACM. 2010, pp. 421–430 (cit. on p. 69).

[30] Will Hill et al. “Recommending and evaluating choices in a virtual community of use.” In: *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM Press/Addison-Wesley Publishing Co. 1995, pp. 194–201 (cit. on p. 8).

[31] Paul Jaccard. *Etude comparative de la distribution florale dans une portion des Alpes et du Jura*. Impr. Corbaz, 1901 (cit. on p. 19).

[32] Kokil Jaidka, Muthu Kumar Chandrasekaran, and Min-Yen Kan, eds. *Proceedings of the Computational Linguistics Scientific Summarization Shared Task (CL-SciSumm 2017) organized as a part of the 2nd Joint Workshop on Bibliometric-enhanced Information Retrieval and Natural Language Processing for Digital Libraries (BIRNDL 2017) and co-located with the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2017), Tokyo, Japan, August 11, 2017*. Vol. 2002. CEUR Workshop Proceedings. CEUR-WS.org, 2017 (cit. on p. 71).

[33] Kokil Jaidka et al. “Overview of the CL-SciSumm 2016 Shared Task.” In: *BIRNDL@JCDL*. Vol. 1610. CEUR Workshop Proceedings. CEUR-WS.org, 2016, pp. 93–102 (cit. on p. 71).

[34] Kokil Jaidka et al. “The CL-SciSumm Shared Task 2017: Results and Key Insights.” In: *BIRNDL@SIGIR (2)*. Vol. 2002. CEUR Workshop Proceedings. CEUR-WS.org, 2017, pp. 1–15 (cit. on pp. 71, 73).

[35] Kokil Jaidka et al. “The CL-SciSumm Shared Task 2018: Results and Key Insights.” In: *BIRNDL@SIGIR*. Vol. 2132. CEUR Workshop Proceedings. CEUR-WS.org, 2018, pp. 74–83 (cit. on p. 71).

[36] *Journal Article Tag Suite (JATS)*. `https://jats.nlm.nih.gov/index.html` (cit. on p. 36).

[37] Samaneh Karimi et al. “University of Houston @ CL-SciSumm 2017: Positional language Models, Structural Correspondence Learning and Textual Entailment.” In: *BIRNDL@SIGIR (2)*. Vol. 2002. CEUR Workshop Proceedings. CEUR-WS.org, 2017, pp. 73–85 (cit. on p. 71).

[38] George Karypis. “Evaluation of Item-Based Top-N Recommendation Algorithms.” In: *Proceedings of the Tenth International Conference on Information and Knowledge Management*. CIKM ’01. Atlanta, Georgia, USA: ACM, 2001, pp. 247–254. ISBN: 1-58113-436-3. DOI: 10.1145/502585.502627. URL: http://doi.acm.org/10.1145/502585.502627 (cit. on p. 9).

[39] Leo Katz. “A new status index derived from sociometric analysis.” In: *Psychometrika* 18.1 (1953), pp. 39–43 (cit. on p. 70).

[40] Frederick Wilfrid Lancaster and Emily Gallup. *Information retrieval on-line*. Tech. rep. 1973 (cit. on p. 15).

[41] Anne Lauscher, Goran Glavas, and Kai Eckert. “University of Mannheim @ CLSciSumm-17: Citation-Based Summarization of Scientific Articles Using Semantic Textual Similarity.” In: *BIRNDL@SIGIR (2)*. Vol. 2002. CEUR Workshop Proceedings. CEUR-WS.org, 2017, pp. 33–42 (cit. on p. 71).

[42] Lei Li et al. “CIST@CLSciSumm-17: Multiple Features Based Citation Linkage, Classification and Summarization.” In: *BIRNDL@SIGIR (2)*. Vol. 2002. CEUR Workshop Proceedings. CEUR-WS.org, 2017, pp. 43–54 (cit. on pp. 71, 72).

[43] Chin-Yew Lin. “Rouge: A package for automatic evaluation of summaries.” In: *Text Summarization Branches Out* (2004) (cit. on p. 72).

[44] Yang Lu et al. “Recommending Citations with Translation Model.” In: *Proceedings of the 20th ACM International Conference on Information and Knowledge Management*. CIKM ’11. Glasgow, Scotland, UK: ACM, 2011, pp. 2017–2020. ISBN: 978-1-4503-0717-8. DOI: 10.1145/2063576.2063879. URL: http://doi.acm.org/10.1145/2063576.2063879 (cit. on p. 70).

[45] Shutian Ma et al. “NJUST @ CLSciSumm-17.” In: *BIRNDL@SIGIR (2)*. Vol. 2002. CEUR Workshop Proceedings. CEUR-WS.org, 2017, pp. 16–25 (cit. on pp. 71, 72).

[46] Tariq Mahmood and Francesco Ricci. “Improving Recommender Systems with Adaptive Conversational Strategies.” In: *Proceedings of the 20th ACM Conference on Hypertext and Hypermedia*. HT ’09. Torino, Italy: ACM, 2009, pp. 73–82. ISBN: 978-1-60558-486-7. DOI: 10.1145/1557914.

1557930. URL: http://doi.acm.org/10.1145/1557914.1557930 (cit. on p. 5).

[47] Christopher D Manning, Prabhakar Raghavan, Hinrich Schütze, et al. *Introduction to information retrieval*. Vol. 1. 1. Cambridge university press Cambridge, 2008 (cit. on pp. 22, 32).

[48] *MariaDB Foundation*. https://mariadb.org/ (cit. on p. 43).

[49] Sean M. McNee et al. "On the Recommending of Citations for Research Papers." In: *Proceedings of the 2002 ACM Conference on Computer Supported Cooperative Work*. CSCW '02. New Orleans, Louisiana, USA: ACM, 2002, pp. 116–125. ISBN: 1-58113-560-2. DOI: 10.1145/587078.587096. URL: http://doi.acm.org/10.1145/587078.587096 (cit. on p. 69).

[50] Dunja Mladenic. "Text-Learning and Related Intelligent Agents: A Survey." In: *IEEE Intelligent Systems* 14.4 (July 1999), pp. 44–54. ISSN: 1541-1672. DOI: 10.1109/5254.784084. URL: http://dx.doi.org/10.1109/5254.784084 (cit. on p. 6).

[51] Martin F Porter. "An algorithm for suffix stripping." In: *Program* 14.3 (1980), pp. 130–137 (cit. on pp. 33, 45).

[52] Aniket Pramanick et al. "SciSumm 2017: Employing Word Vectors for Identifying, Classifying and Summarizing Scientific Documents." In: *BIRNDL@SIGIR (2)*. Vol. 2002. CEUR Workshop Proceedings. CEUR-WS.org, 2017, pp. 94–98 (cit. on p. 71).

[53] Animesh Prasad. "WING-NUS at CL-SciSumm 2017: Learning from Syntactic and Semantic Similarity for Citation Contextualization." In: *BIRNDL@SIGIR (2)*. Vol. 2002. CEUR Workshop Proceedings. CEUR-WS.org, 2017, pp. 26–32 (cit. on pp. 71, 72).

[54] *PubMed Central*. https://www.ncbi.nlm.nih.gov/pmc/ (cit. on p. 36).

[55] *PubMed Central FTP Service*. https://www.ncbi.nlm.nih.gov/pmc/tools/ftp/ (cit. on p. 36).

[56] *PubMed Central Open Access Subset*. https://www.ncbi.nlm.nih.gov/pmc/tools/openftlist/ (cit. on p. 36).

[57] *Qt Jambi*. http://qtjambi.org/ (cit. on p. 43).

[58] Paul Resnick and Hal R. Varian. "Recommender Systems." In: *Commun. ACM* 40.3 (Mar. 1997), pp. 56–58. ISSN: 0001-0782. DOI: 10.1145/245108.245121. URL: http://doi.acm.org/10.1145/245108.245121 (cit. on p. 5).

[59] Paul Resnick et al. "GroupLens: an open architecture for collaborative filtering of netnews." In: *Proceedings of the 1994 ACM conference on Computer supported cooperative work*. ACM. 1994, pp. 175–186 (cit. on p. 8).

[60] Francesco Ricci et al. *Recommender Systems Handbook*. 1st. New York, NY, USA: Springer-Verlag New York, Inc., 2010. ISBN: 0387858199, 9780387858197 (cit. on pp. 5, 6, 8).

[61] Anna Ritchie. *Citation context analysis for information retrieval*. Tech. rep. University of Cambridge, Computer Laboratory, 2009 (cit. on p. 69).

[62] Stephen E Robertson et al. "Okapi at TREC." In: *The First Text REtrieval Conference (TREC-1)*. Ed. by Donna K Harman. US Department of Commerce, National Institute of Standards and Technology, 1993, pp. 21–30 (cit. on p. 25).

[63] Stephen E Robertson et al. "Okapi at TREC-2." In: *The Second Text REtrieval Conference (TREC-2)*. Ed. by Donna K Harman. US Department of Commerce, National Institute of Standards and Technology, 1994, pp. 21–34 (cit. on p. 25).

[64] Stephen E Robertson et al. "Okapi at TREC-3." In: *The Thrid Text REtrieval Conference (TREC-3)*. Ed. by Donna K Harman. US Department of Commerce, National Institute of Standards and Technology, 1995, pp. 109–128 (cit. on pp. 25, 27).

[65] Stephen E Robertson and K Sparck Jones. "Relevance weighting of search terms." In: *Journal of the American Society for Information science* 27.3 (1976), pp. 129–146 (cit. on pp. 22, 24).

[66] Stephen E Robertson and Steve Walker. "Some simple effective approximations to the 2-poisson model for probabilistic weighted retrieval." In: *Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*. Springer-Verlag New York, Inc. 1994, pp. 232–241 (cit. on p. 27).

[67] G. Salton. *The SMART Retrieval System—Experiments in Automatic Document Processing*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1971 (cit. on p. 22).

[68] G. SALTON and C.S. YANG. "ON THE SPECIFICATION OF TERM VALUES IN AUTOMATIC INDEXING." In: *Journal of Documentation* 29.4 (1973), pp. 351–372. DOI: `10.1108/eb026562`. eprint: `https://doi.org/10.1108/eb026562`. URL: `https://doi.org/10.1108/eb026562` (cit. on p. 20).

[69] Gerard Salton. "Automatic information organization and retrieval." In: (1968) (cit. on p. 17).

[70] Gerard Salton and Christopher Buckley. "Term-weighting approaches in automatic text retrieval." In: *Information Processing & Management* 24.5 (1988), pp. 513–523. ISSN: 0306-4573. DOI: `https://doi.org/10.1016/0306-4573(88)90021-0`. URL: `http://www.sciencedirect.com/science/article/pii/0306457388900210` (cit. on p. 20).

[71] Gerard Salton, Anita Wong, and Chung-Shu Yang. "A vector space model for automatic indexing." In: *Communications of the ACM* 18.11 (1975), pp. 613–620 (cit. on p. 17).

[72] Badrul Sarwar et al. "Item-based collaborative filtering recommendation algorithms." In: *Proceedings of the 10th international conference on World Wide Web*. ACM. 2001, pp. 285–295 (cit. on p. 9).

[73] Upendra Shardanand and Pattie Maes. "Social information filtering: algorithms for automating "word of mouth"." In: *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM Press/Addison-Wesley Publishing Co. 1995, pp. 210–217 (cit. on p. 8).

[74] W.M. Shaw, Robert Burgin, and Patrick Howell. "Performance standards and evaluations in IR test collections: Cluster-based retrieval models." In: *Information Processing & Management* 33.1 (1997), pp. 1–14. ISSN: 0306-4573. DOI: `https://doi.org/10.1016/S0306-4573(96)00043-X`. URL: `http://www.sciencedirect.com/science/article/pii/S030645739600043X` (cit. on p. 72).

[75] Trevor Strohman, W Bruce Croft, and David Jensen. “Recommending citations for academic papers.” In: *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*. ACM. 2007, pp. 705–706 (cit. on p. 69).

[76] *The Apache Lucene project*. `https://lucene.apache.org/` (cit. on p. 37).

[77] *The JAXB Project*. `https://jaxb.java.net/` (cit. on p. 46).

[78] *The Qt Framework*. `https://www.qt.io/` (cit. on p. 43).

[79] Ellen M Voorhees et al. “The TREC-8 Question Answering Track Report.” In: *Trec*. Vol. 99. 1999, pp. 77–82 (cit. on p. 54).

[80] Dongxu Zhang and Sujian Li. “PKU @ CLSciSumm-17: Citation Contextualization.” In: *BIRNDL@SIGIR (2)*. Vol. 2002. CEUR Workshop Proceedings. CEUR-WS.org, 2017, pp. 86–93 (cit. on p. 71).

[81] George Kingsley Zipf. “Selected studies of the principle of relative frequency in language.” In: (1932) (cit. on p. 21).

Appendix

Appendix A.

Contribution to the CL-SciSumm 2017 Shared Task

Graz University of Technology at CL-SciSumm 2017: Query Generation Strategies

Thomas Felber and Roman Kern

Institute of Interactive Systems and Data Science
Graz University of Technology
Know-Center GmbH
Inffeldgasse 13, 8010 Graz, Austria
felber@student.tugraz.at,rkern@know-center.at

Abstract. In this report we present our contribution to the 3rd Computational Linguistics Scientific Document Summarization Shared Task (CL-SciSumm 2017), which poses the challenge of identifying the spans of text in a reference paper (RP) that most accurately reflect a citation (i.e. citance) from another document to the RP. In our approach, we address this challenge by applying techniques from the field of information retrieval. Therefore we create a separate index for every RP and then transform each citance to a RP into a query. This query is subsequently used to retrieve the most relevant spans of text from the RP. Different ranking models and query generation strategies have been employed to alter which spans of text are retrieved from the index. Furthermore we implemented a k-nn classification based on our search infrastructure for assigning the cited text span to pre-defined classes.

Keywords: Information Retrieval, Query Generation, Ranking Models

1 Introduction

The focus of the CL-SciSumm 2017 Shared Task is on automatic paper summarization in the Computational Linguistics (CL) domain. It is organized as part of the 2nd Joint Workshop on Bibliometric-enhanced Information Retrieval and Natural Language Processing for Digital Libraries (BIRNDL 2017)[1][2], held at the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval[2] in Tokyo, Japan. It is a follow-up on the CL-SciSumm 2016 Shared Task at BIRNDL 2016 [1] which was conducted in the course of the Joint Conference on Digital Libraries (JCDL '16) in Newark, New Jersey.

This Shared Task is divided into multiple smaller tasks which pose the following problems:

- **Task 1A:** Identify the spans of text (cited text spans) in a reference paper (RP) that most accurately reflect a citation (i.e. citance) to the RP made from another document.

[1] http://wing.comp.nus.edu.sg/ birndl-sigir2017/
[2] http://sigir.org/sigir2017/

- **Task 1B:** Classify each identified cited text span according to a predefined set of facets. The elements of that set are: *Implication*, *Method*, *Aim*, *Results*, and *Hypothesis*.
- **Task 2 (optional bonus task):** Generate a structured summary of the RP from the identified cited text spans of the RP, where the length of the summary must not exceed 250 words.

The data set provided for these tasks comprises a training set and a test set consisting of 30 and 10 RPs respectively. Each RP is associated with a set of citing papers (CP) which all contain citations to the RP. In each CP, the text spans (citances) have been identified that pertain to a particular citation to the RP.

To tackle the problem in Task 1A, we followed an information retrieval (IR) approach. For every RP, we created an index holding all the spans of text of that RP. A citance to a RP is transformed into a query and performed on the index associated with the RP to retrieve the most relevant spans of text.

For Task 1B, we followed a k-NN classification approach. Each identified cited text span is compared against all different cited text spans in the training set. Among the top five most similar cited text spans a majority vote is used to determine the facet.

2 Task 1A: Identification of Text Spans in the RP

In this section we provide a closer look at our approach to Task 1A. We will describe how the indices to the RPs are created as well as how the citances are turned into queries and subsequently used to identify relevant spans of text in the RP.

2.1 Index Creation

In order to create an index to a RP, which holds all the different spans of text of the RP, we used the Apache Lucene text search engine library[3] which features Java-based indexing and searching technology.

Taking advantage of the library's indexing technology, we created an index for every RP and added all spans of text of the RP to the index. In this scenario a single span of text can be imagined as a separate text document that is being added to a conventional index. Before we added anything to the index, however, we performed two additional preprocessing steps on every span of text. At first, all stop words contained in a span of text were removed. The idea behind this is to tune the performance of the index (fewer terms in the index) and to obtain more relevant search results since stop words only carry little distinguishing potential [6].

To decide which words qualify as stop words and which do not, we used Apache Lucene's integrated list of stop words for the English language. As a

[3] http://lucene.apache.org/

second preprocessing step, we stripped down all suffixes of all words contained in a span of text in order to normalize them. This was achieved by applying Porter's stemming algorithm [3]. After the preprocessing on a span of text was completed, we moved on and added it to the index.

2.2 Query Generation

After all the indices for the RPs were in place, we transformed each citance to a RP into a query and ran it on the index associated with the RP in order to retrieve the most relevant spans of text of the RP for that particular citance. Since all indices were constructed with Apache Lucene, we also resorted to Apache Lucene functionality to generate the queries. There exists a broad range of different query types in Apache Lucene, however, after we conducted various experiments on the training set, we found that using Apache Lucene's *TermQuery* generated the best results.

To turn a citance into a query, we first applied two preprocessing steps to the citance. The preprocessing steps applied, are analogous to the ones described in section 2, that is, stop words were removed from the citance and porter stemming was performed. As a next step we extracted all words from the citance and created a *TermQuery* for every word. This means, each *TermQuery* corresponds to a single word in the citance. After that, we created an Apache Lucene *BooleanQuery* by OR-conjuncting all *TermQueries*. This resulting *BooleanQuery* was then used to query the index associated with the RP that the citance refers to.

As a result of the query, we obtained a set of top ranked spans of text of the RP. The elements of that set are ordered according to a score, however, it depends on the ranking and retrieval model used by the index what elements are in the set and what score they are given. From all spans of text that are retrieved this way, we considered the top two as most accurately reflecting the corresponding citance. This is because considering the top two yielded the best results during experiments on the training set.

2.3 Ranking

In section 2.2 we mentioned that it depends on the ranking and retrieval model that is used by the index which elements are retrieved by a query and how they are ranked. For the sake of the CL-SciSumm 2017 Shared Task, we submitted a system run using a simple vector space model (VSM) [5] based method and the popular BM25 model [4].

Vector Space Model The term frequency and inverse document frequency (TF-IDF) weighting scheme used by Apache Lucene within the scope of the VSM is as follows:
For the term frequency, which correlates to a term t in a document d, the formula

$$tf_{t,d} = \sqrt{f_{t,d}} \tag{1}$$

is used, where $f_{t,d}$ denotes the number of times the term t occurs in document d.

For the inverse document frequency, which correlates to the number of documents in which the term t appears, the formula

$$idf_t = 1 + \log \frac{N}{n_t + 1} \tag{2}$$

is used, where N is the total number of documents in the index and n_t is the number of documents containing the term t.

The score of a document d to a query q is calculated based on the cosine similarity and is defined as

$$sim(d, q) = \frac{V(d) \cdot V(q)}{|V(d)||V(q)|} \tag{3}$$

where $V(d) \cdot V(q)$ is the dot product of the weighted vectors, and $|V(d)|$ and $|V(q)|$ are their euclidean norms.

BM25 The term frequency factors used in Apache Lucene within the scope of BM25 ranking are defined as

$$\mathcal{B}_{t,d} = \frac{(k_1 + 1) f_{t,d}}{k_1 \left[(1 - b) + b \frac{|d|}{i_{boost}^2 |d|_{avg}} \right] + f_{t,d}} \tag{4}$$

where $f_{t,d}$ denotes the number of times the term t occurs in document d, $|d|$ is the length of the document d in words, $|d|_{avg}$ is the average document length, i_{boost} is an index-time boosting factor, and k_1 and b are parameters.

The ranking equation used in the BM25 model can then be written as

$$sim(d, q) = \sum_{t[q,d]} \mathcal{B}_{t,d} \times \log \frac{N - n_t + 0.5}{n_t + 0.5}. \tag{5}$$

The values we used for the parameters k_1 and b are 1.2 and 0.75 respectively.

3 Task 1B: Identification of the Discourse Facet

The discourse facet takes one of the following values: *Implication*, *Method*, *Aim*, *Results*, and *Hypothesis*. To classify the spans of text, which were identified in Task 1A, we took the following approach: At first we created an index which we filled with all available cited text spans of the training set plus their corresponding discourse facets. To classify which discourse facet a span of text belongs to, we then transformed the span of text into a query, analogous to the way described in 2.2, and then ran the query on the index. After that, we conducted a majority vote on the top five retrieved results to determine the discourse facet to use.

4 Evaluation

Overall we submitted two system runs for Task 1. One of the system runs was conducted using the BM25 ranking model and the other using a simple vector space model (VSM). See section 2.3 for the parameters that we used for these models.

The system performance for Task 1a was determined by measuring the sentence id overlaps between the sentences identified by the system and the gold standard sentences created by human annotators. Based on that, precision, recall and F_1 score were calculated for each system run.

The performance of Task 1b was measured by the proportion of the correctly classified discourse facets by the system, contingent on the expected response of Task 1a. The metrics used here are also precision, recall and F_1 score.

The official evaluation results of our submitted system runs for Task 1a and Task 1b are shown in Table 1 and Table 2 respectively:

Table 1: The Task 1a evaluation results for our system runs using the vector space model (VSM) and BM25.

Ranking Model	**Precision**	**Recall**	**F_1 score**
VSM	0.085	0.138	0.105
BM25	0.107	0.181	0.135

Table 2: The Task 1b evaluation results for our system runs using the vector space model (VSM) and BM25.

Ranking Model	**Precision**	**Recall**	**F_1 score**
VSM	0.917	0.158	0.269
BM25	0.938	0.205	0.337

Judging by the official evaluation results we find that our proposed approaches yield excellent results. Especially our BM25 approach seems to work very well for both Task 1a and Task 1b: An F_1 score of 0.135 at Task 1a achieves the third highest result among all system runs, not far behind the winning system, which has an F_1 score of 0.146. An F_1 score of 0.337 at Task 1b is the eighth highest result among all system runs, with the winning system having an F_1 score of 0.408. Overall 47 system runs have been submitted to Task 1. The mean F_1 score at Task 1a among all system runs is 0.088 and at Task 1b 0.208.

5 Conclusion

In this report we described the approaches we followed to tackle the problems posed in Task 1A and Task 1B of the CL-SciSumm 2017 Shared Task. In preliminary test we found out that using a combination of stop word removal and stemming in combination with a disjunction query strategy work best. The official evaluation results confirmed the success of our approach. We were able to reuse the indexing infrastructure for a classification task, namely assigning categories to the cited text spans. In future work we plan to make use of our infrastructure and investigate methods to enhance the process by integrating more sources of evidence. In particular, additional context information like author or venue specific information might prove beneficial.

Acknowledgements

The Know-Center is funded within the Austrian COMET Program – Competence Centers for Excellent Technologies – under the auspices of the Austrian Federal Ministry of Transport, Innovation and Technology, the Austrian Federal Ministry of Economy, Family and Youth and by the State of Styria. COMET is managed by the Austrian Research Promotion Agency FFG.

References

1. Cabanac, G., Chandrasekaran, M.K., Frommholz, I., Jaidka, K., Kan, M.Y., Mayr, P., Wolfram, D.: Joint workshop on bibliometric-enhanced information retrieval and natural language processing for digital libraries (birndl 2016). In: Proceedings of the 16th ACM/IEEE-CS on Joint Conference on Digital Libraries. pp. 299–300. ACM (2016)
2. Jaidka, K., Chandrasekaran, M.K., Rustagi, S., Kan, M.Y.: Overview of the cl-scisumm 2017 shared task. In: In Proceedings of the Joint Workshop on Bibliometric-enhanced Information Retrieval and Natural Language Processing for Digital Libraries (BIRNDL 2017), Tokyo, Japan, CEUR. (2017)
3. Porter, M.F.: An algorithm for suffix stripping. Program 14(3), 130–137 (1980)
4. Robertson, S.E., Walker, S., Jones, S., Hancock-Beaulieu, M.M., Gatford, M., et al.: Okapi at trec-3. Nist Special Publication Sp 109, 109 (1995)
5. Salton, G., Wong, A., Yang, C.S.: A vector space model for automatic indexing. Commun. ACM 18(11), 613–620 (Nov 1975), http://doi.acm.org/10.1145/361219.361220
6. Silva, C., Ribeiro, B.: The importance of stop word removal on recall values in text categorization. In: Neural Networks, 2003. Proceedings of the International Joint Conference on. vol. 3, pp. 1661–1666. IEEE (2003)