Informatik
Hauptcampus

H O C H
S C H U L E
T R I E R

Konzeption und Realisierung eines Systems zur Informationssuche in einem Dokumentenarchiv basierend auf Textinhalt und Metadaten.

Conception and Realization of an Information Retrival System for a Document Archive based on Text Content and Metadata

Annika Kremer

Bachelor-Abschlussarbeit

Betreuer: Prof. Dr. Karl Hans Bläsius

Trier, Abgabedatum

Kurzfassung

Diese Arbeit befasst sich mit der Frage, auf welche Weise ein Information Retrieval System realsiert werden kann, welches ein Archiv mit semistrukturierten Dokumenten, die sowohl Freitext als auch Metadaten enthalten, effizient nach vom Nutzer festgelegten und logisch verknüpfbaren Kriterien durchsucht. Ziel der Implementierung ist es, dem Nutzer nach Abschluss des Suchvorgangs relevante Informationen zurückzuleifern und diese auf eine übersichtliche Weise zu präsentieren.
Zunächst wird die Problemstellung im Detail erläutert, damit der Leser eine genaue Vorstellung über die Anforderungen, welche die Implementierung erfüllen soll, bekommt. Anschließend wird Information Retrieval im Allgemeinen vorgestellt, um dem Leser einen Überblick über die Thematik zu verschaffen.
Es folgt eine Vorstellung der beiden klassischen Information Retrieval Modelle boolesches Retrieval und Vektorraummodell inklusive Erläuterung der Funktionsweise. Anschließend wird beschrieben, wie sich solche Modelle in Hinsicht auf deren Qualität bewerten und vergleichen lassen.
Nach dem Vermitteln der notwendigen theoretischen Kenntnisse beschreibt der Implementierungsteil, auf welche Weise und in welchen Bereichen die beiden Verfahren für die Implementierung dieser Arbeit zum Einsatz kamen und inwieweit eine Modifizierung zur Anpassung auf die vorliegende Problemstellung erfolgte. Neben der internen Funktionsweise wird in dieser Ausarbeitung auch die Benutzung der Oberfläche erläutert, um den Anwender mit der Bedienung des Systems vertraut zu machen. Es folgt eine Zusammenfassung der Arbeit mit abschließender Bewertung der Ergebnisse sowie ein Ausblick auf weitere Mögliche Verbesserungen.

This paper discusses the question of how an information retrieval system that searches efficiently in semistructured data containing free text and metadata can be realized. The search criteria for this system are determined by the user and can be freely connected with boolean operators. The aim of the implementation is to present relevant information in an clearly arranged way after finishing the search process.
First the problem is discussed in detail to give the reader a precise idea of the requirements which the implementation has to fit. Afterwards, information retrieval

in general is presented to give the reader an overview on the topic.
The chapter about information retrieval is followed by a presentation of two classical information retrieval model including their functionality called boolean retrieval and vectorspace model. Afterwards, it is discussed how the quality of such systems can be estimated and compared.
After having provided the necessary theory, the paper continues with the implementation part, which explains which models were used in which way in this implementation and if they have been modified in any way to fit this problem.
Aside from the intern functionality, the reader learns about how to operate with the system via the given user interface.
The paper concludes with a summary including a final result evaluation and a presentation of future prospects for possible system improvements.

Inhaltsverzeichnis

1 Einleitung und Problemstellung 1
1.1 Einleitung 1
1.2 Problemstellung 1
1.3 Teilprobleme 2
1.3.1 Dynamisches einlesen der Metadaten 2
1.3.2 Unterscheidung Metadaten und Freitext 2
1.3.3 Metadatensuche 2
1.3.4 Freitextsuche 2
1.3.5 Verknüpfung mit UND/ODER 3
1.3.6 Benutzeroberfläche 3

2 Information Retrieval 4
2.1 Bedeutung 4
2.1.1 Information 4
2.1.2 Information Retrieval 5
2.1.3 Unterschied zur Datenbankensuche 5
2.2 Beispiel Websuche 5
2.3 Bezug zur Problemstellung 6
2.3.1 Teilweise strukturierte Daten 6

3 Grundbegriffe 7
3.1 Anfrage 7
3.2 Indexierung 7
3.2.1 Term und Vokabular 8
3.2.2 Stopwords 8

4 Boolesches Retrieval 9
4.1 Eigenschaften des Verfahrens 9
4.2 Funktionsprinzip 9
4.2.1 Attribut 9
4.2.2 Anfragen 10
4.3 Implementierungsansätze 11
4.3.1 Term-Inzidenz-Matrix 11

4.3.2 Invertierte Liste 12

5 Das Vektorraummodell 16
5.1 Funktionsprinzip 16
5.2 Vektor und Vektorraum 16
5.3 Definition Vektorraummodell 17
5.4 Gewichte 18
5.4.1 Termhäufigkeit 18
5.4.2 Dokumenthäufigkeit 19
5.4.3 Invertierte Dokumenthäufigkeit 19
5.4.4 TF-IDF-Gewichtung 19
5.5 Anfragen 20
5.6 Ähnlichkeitsfunktion 20
5.6.1 Euklidischer Abstand 20
5.6.2 Cosinus-Maß 21

6 Bewertung eines Information Retrieval Systems 23
6.1 Problem Relevanz 23
6.2 Precision und Recall 24
6.2.1 Precisison 24
6.2.2 Recall 24
6.2.3 Veranschaulichung 24
6.3 Zwei Evaluierungsmaße 25
6.4 Durchführung 25

7 Implementierung 26
7.1 Teilweise strukturierte Dokumente 26
7.2 Vorverarbeitung 27
7.2.1 Erstellen des Dokument-Dictionaries 27
7.2.2 Verarbeiten der Dokumente 27
7.3 Die Suche 30
7.3.1 Keywordsuche 30
7.3.2 Freitextsuche 31
7.3.3 Erstellen des Query-Vektors 31
7.4 Verrechnung der Suchergebnisse 31

8 Die Benutzeroberfläche 33
8.1 Anforderungen 33
8.2 Grundaufbau der Oberfläche 33
8.3 Verzeichnisauswahl 34
8.4 Suchanfrage 35
8.4.1 Zusammenstellung der Teilanfrage 35
8.4.2 Beispiel 36
8.5 Gesamtanfrage 36
8.6 Ergebnis 37

9 Zusammenfassung und Ausblick 39
9.1 Zusammenfassung 39
9.2 Ausblick: Wünschenswerte Erweiterungen 39

Literaturverzeichnis 41

Erklärung der Kandidatin / des Kandidaten 43

Abbildungsverzeichnis

4.1 Term-Dokument-Inzidenz-Matrix. Die Zeile enthalten die Terme, die Spalten die Dokumente (hier durch deren *docID* repräsentiert). Alle Einträge mit einer 0 sind leere Einträge (Eigene Abbildung). . . 11
4.2 Invertierte Listen zu Beispieltermen. Die Zahlen sind die Dokumentindizes oder auch *docID*s, in denen der jeweilige Term vorkommt (Eigene Abbildung). 12

5.1 Vektorraum mit den Termen $T1$ und $T2$ als Achsen, drei Dokumentvektoren und dem Anfragevektor. Das Cosinus-Maß liefert als ähnlichstes Dokument d_2 (Eigene Abbildung, basierend auf [SB10], S.55). 22

7.1 Struktur für einen Eintrag im Dokument-Dictionary. Die Einträge Datum und Absender können, falls nicht existent, auch leer bleiben. Im rot markierten Bereich werden später die Dokumentvektoren eingetragen. (Eigene Abbildung). 28
7.2 Modifizierte invertierte Liste zur Realisierung der Keywordsuche mittels Structs der Form (docID, Typ, Inhalt) (Eigene Abbildung). . 29
7.3 Struktur für einen Eintrag im Term-Dictionary. Der Index gibt die Position im Dokumentvektor an (Eigene Abbildung). 29
7.4 Beispiel für Umwandlung einer Anfrage in einen Query-Vektor. „Kontaktadresse“ hat den Index 3, „Seminar“ den Index 123. Mai kommt im Archiv nicht vor, darum wird der Term nicht vermerkt. Alle Indizes ungleich 3 und 123 sind als mit 0 gewichtet zu interpretieren (Eigene Abbildung). 32

8.1 Gliederung der Benutzeroberfläche. Teil a beinhaltet die Verzeichnisauswahl, Teil b die gesamte Suchanfrage und Teil$c$ die Eingabe der Teilanfrage (Eigene Abbildung). 34
8.2 Verzeichnisauswahl (Eigene Abbildung) 35
8.3 Keywords auswählen und intern verknüpfen, hier OR (eigene Abbildung) 36
8.4 Freitextauswahl. Da nur ein Bereich ausgewählt ist, bleibt der selektierte OR-Operator ohne Wirkung (eigene Abbildung) 37

8.5 Display (eigene Abbildung) 37
8.6 Resultatfenster (eigene Abbildung) 38
8.7 Scrollbare Anzeige für viele Treffer (eigene Abbildung) 38

Tabellenverzeichnis

6.1 Kontingenztafel ([CDM08], S.143) 24

1

Einleitung und Problemstellung

1.1 Einleitung

Nahezu jeder nutzt heutzutage E-Mail-Dienste und kennt die Problematik, dass der Posteingang sich unter der Flut täglich eintreffender Nachrichten stetig füllt, bis der Ordner unübersichtlich voll ist.
Dies wird zu einem ernsthaften Problem, sobald etwas bestimmtes darin wiedergefunden werden soll.
Die benötigte Mail war wichtig, weil sie eine bestimmte Kontaktadresse enthielt, aber wie war nochmal der Absender? Längst vergessen. Das genaue Datum? Leider ist nur noch der Monat bekannt. Wer die Suche manuell bewältigen muss, ist an dieser Stelle verloren.
Abhilfe schafft ein Information Retrieval System, mit dem man bestimmte Suchkriterien eingeben und beliebig miteinander kombinieren kann.
Das System liefert idealerweise eine Reihe passender Resultate und der Anwender braucht sich nicht selbst durch hunderte von Mails durchzuarbeiten.
In diesem Fall könnte die Person beispielsweise im Freitext nach dem Wort *„Kontaktadresse“* suchen und weitere Kriterien wie *„Datum = Juni “* hinzufügen.
Besonderheit ist das beliebige Verknüpfen: Der Anwender kann sich entscheiden, ob er nur Resultate akzeptiert, auf die Beides zutrifft, oder ob es bereits reicht, wenn eines der Kriterien erfüllt ist. Dies erlaubt eine sehr individuelle, auf die Informationsbedürfnisse des Nutzers zugeschnittene Suche.
Ein solches System ist nicht nur für das Alltagsbeispiel E-Mail-Ordner, d.h. Posteingang, Postausgang etc. wertvoll, sondern lässt sich auch auf jede andere Art von Dokumentenarchiv, dessen Dokumente sowohl Freitext als auch Metadaten beinhalten, anwenden.

1.2 Problemstellung

Ziel der Arbeit ist die Konzeption und Realisierung eines Systems zur Informationssuche (engl. Information Retrieval System) in einem Dokumentenarchiv, wobei die Dokumente von teilweise strukturierter Natur sind.
Dies bedeutet, dass sie sowohl gewöhnlichen Freitext als auch Metadaten enthalten. Der Nutzer soll spezifizieren können, in welchen Metadaten er suchen möchte,

zudem soll die Freitextsuche auswählbar sein. Alle Suchanfragen sollen hierbei beliebig mit den booleschen Operatoren *AND* (engl. und) sowie *OR* (engl. oder) verknüpfbar sein.
Hauptanwendungszweck des Systems sind E-Mail-Archive wie Posteingang und Postausgang, allerdings soll das System so flexibel sein, dass es auch auf andere teilweise strukturierte Dokumentenarchive anwendbar ist.

1.3 Teilprobleme

Aus der Aufgabenstellung ergeben sich die im folgenden beschriebenen Teilprobleme.

1.3.1 Dynamisches einlesen der Metadaten

Die genauen Metadaten sind, da das System flexibel sein soll, vor dem Ausführen des Systems noch nicht bekannt. Demnach muss das IR-System die Namen der Metadaten beim Starten des Programms dynamisch einlesen und diese dem Nutzer auf der grafischen Oberfläche anzeigen.

1.3.2 Unterscheidung Metadaten und Freitext

Damit das System die Metadaten dynamisch einlesen kann, müssen die folgenden Punkte erfüllt sein:

1. Das System muss zwischen Metadaten und Freitext unterscheiden können.
2. Metadaten setzen sich aus Name und Inhalt zusammen, weshalb beides erkannt und voneinander abgegrenzt werden muss. Im folgenden werden die Namen als „Keywords “ bezeichnet.
3. Der Inhalt kann unterschiedlichen Datentyps sein, z.B. String oder Liste, weshalb dieser bestimmt werden muss.

1.3.3 Metadatensuche

Es muss erkannt werden, welche Metadaten der Nutzer ausgewählt hat und in genau diesen muss, unter Berücksichtigung des jeweiligen Datentyps der Inhalte, gesucht werden.
Im Gegensatz zur Freitextsuche muss zu jedem Dokument vor der Suche zunächst geprüft werden, ob das entsprechende Schlüsselwort überhaupt darin auftritt.

1.3.4 Freitextsuche

Bei der Freitextsuche ist die Wortzahl weitaus größer als bei der Metadatensuche. Daraus resultieren zwei Probleme:

1. Wie kann effizient in großen Wortmengen gesucht werden?
2. Wie kann die Suche bei begrenztem Speicher bewältigt werden?

Zudem stellt sich die Frage nach einem geeigneten Verfahren, das bei längeren Anfragen auch teilweise passende Ergebnisse liefert und messen kann, wie gut die erzielten Treffer zur Nutzeranfrage passen.

1.3.5 Verknüpfung mit UND/ODER

Alle Anfragen sollen beliebig mit den logischen Operatoren *AND*, *OR* sowie *NOT* verknüpfbar sein. Dies beinhaltet die folgenden Problemstellungen:

1. Keywordsuche und Freitextsuche müssen miteinander verknüpft werden.
2. Sind meherere Keywords ausgewählt, müssen die Teilergebnisse verknüpft werden.
3. Stellt der Nutzer mehrere Anfragen, müssen die Ergebnisse der einzelnen Anfragen verknüpft werden.

1.3.6 Benutzeroberfläche

Der Nutzer benötigt eine verständliche Benutzeroberfläche, die es ihm ermöglicht, seine Suchanfragen beliebig zusammenzustellen. Hierzu muss die Oberfläche folgende grundlegenden Funktionalitäten aufweisen:

1. Das Suchverzeichnis, d.h. das Dokumentenarchiv in welchem die Suche stattfindet, muss auswählbar sein.
2. Alle im Archiv auftretenden Keywords sowie die Freitextsuche müssen auswählbar sein.
3. Logische Operatoren (AND,OR,NOT) zur Verknüpfung müssen auswählbar sein.
4. Die Suchanfrage muss für den Nutzer verständlich angezeigt werden.

2

Information Retrieval

Dieses Kapitel soll dem Leser einen Überblick über die Bedeutung des Begriffs „Information Retrieval “ vermitteln.

2.1 Bedeutung

Der englische Begriff „Information Retrieval “ lässt sich mit „Informationsrückgewinnung“ ins Deutsche übersetzen. [Aca] Hierbei wird explizit von *Rück*gewinnung gesprochen, da keine neuen Informationen erzeugt werden.
Bevor erklärt wird, wofür Informationsrückgewinnung tatsächlich steht, wird zunächst auf den Teilbegriff „Information“ eingegangen.

2.1.1 Information

Die Bedeutung des Begriffs Information ist sehr weit gefasst, was eine einheitliche Definition unmöglich macht.
Er stammt aus dem Lateinischen (informare = Gestalt geben) und heißt im Übertragenen Sinne so viel wie jemanden durch Unterweisung bilden.
Dies betont den Aspekt, dass eine Information stets einen Empfänger besitzt, welcher „gebildet“ wird. Dies kann eine Person, aber auch ein geeignetes, nach außen wirksames System sein.
Daten als Träger von Informationen müssen vom Empfänger aufgenommen und korrekt interpretiert werden, damit aus den Daten tatsächlich Informationen entstehen.
Die Informationen müssen deshalb auf irgendeine Weise dargestellt werden, z.B. durch alphabetische Zeichen, außerdem muss es hierfür einen geeigneten Träger geben. Dies kann beispielsweise ein Textdokument sein.
Information lassen sich in die folgenden drei Bestandteile zerlegen:

- Syntaktischer Teil: Ist die Struktur der Information syntaktisch zulässig? Beispiel hierfür ist die Einhaltung von Rechtschreibung und Grammatik bei Texten.
- Semantischer Teil: Welchen inhaltliche Bedeutung besitzt die Information?
- Pragmatischer Teil: Welchem Zweck dient sie?

([PDVC06], S.314-315)

2.1.2 Information Retrieval

Nachdem bekannt ist, worum es sich bei Informationen handelt, wird im Folgenden beschrieben, worauf sich Information Retrieval bezieht.
Auch hier ist es problematisch, eine einheitliche Definition zu finden. Eine mögliche Erklärung lautet so:

Definition 2.1. *(Information Retrieval)*

Mit Information Retrieval, kurz IR, wird das Auffinden von in unstrukturierter Form vorliegender und ein Informationsbedürfnis befriedigender Materialien innerhalb großer Sammlungen bezeichnet.

Mit unstrukturierten Materialien sind hierbei meist Dokumente in Textform gemeint, es sind jedoch auch andere Formate möglich. Üblicherweise liegen die Sammlungen auf dem Computer gespeichert vor ([CDM08], S.1).

2.1.3 Unterschied zur Datenbankensuche

Information Retrieval unterscheidet sich stark von der Suche in Datenbanken.
In einer Datenbank liegen die Daten strukturiert in Form von Werttupeln bekannten Datentyps vor, was Definition 2.1 widerspricht.
Im Gegensatz zum Information Retrieval kann bei der Datenbankensuche nicht mit vagen Anfragen umgegangen werden. In der Datenbank kann zwar nach $(Miete < 300)$ gesucht werden, aber mit *„günstige Miete“* wäre das System überfordert: Wie ist günstig zu interpretieren? ([Fer03], S.10)
Ein Information Retrieval System muss solche Anfragen mit unklarer Bedeutung verarbeiten können.

2.2 Beispiel Websuche

Zum besseren Verständnis soll an dieser Stelle ein Beispiel zur Veranschaulichung gegeben werden.
Nahezu jeder benutzt im Alltag Web-Suchmaschinen. Websuche stellt eine Form Information Retrieval dar, da hier Freitext beinhaltende Dokumente (z.B. im HTML- oder pdf-Format) aufgefunden werden sollen, um das Informationsbedürfnis des Internetnutzers zu befriedigen. ([Fer03], S.6)
Möchte dieser zum Beispiel seinen nächsten Urlaub planen, könnte seine Suchanfrage *„Hotel günstig Kreta“* lauten.
Die gesuchten Informationen dienen also dem Zweck, den Urlaub zu planen.
Problematisch ist hierbei der semantische Teil der Informationen: Die inhaltliche Bedeutung der Resultate muss mit der ursprünglichen Intention des Nutzers

übereinstimmen.
Ein von der Suchmaschine geliefertes Resultat kann zwar zum syntaktischen Teil passen, da die korrekten Wörter darin auftauchen, allerdings in einem ganz anderen Kontext, sodass der Nutzer mit dem Dokument nichts anfangen kann.

2.3 Bezug zur Problemstellung

Die Aufgabe besteht darin, nach vom Nutzer auswählbaren, logisch verknüpften Kriterien innerhalb eines Dokumentenarchivs zu suchen (1.2).
Demnach ist die Definition 2.1 erfüllt, da hier Materialien innerhalb einer Sammlung, dem Dokumentenarchiv, aufgefunden werden sollen, um ein Informationsbedürfnis zu befriedigen.
Dieses Bedürfnis unterscheidet sich natürlich von Anfrage zu Anfrage, besteht aber allgemein gefasst darin, Dokumente wiederzufinden, z.B. eine bestimmte E-Mail.

2.3.1 Teilweise strukturierte Daten

Besonderheit der Problemstellung ist hierbei, dass die Dokumente teilweise strukturiert sind, d.h. es liegt zwar Freitext vor, aber zusätzlich sind Metadaten vorhanden.
Im Falle der Freitextsuche lässt sich aufgrund der unstrukturierten Textform eindeutig von Information Retrieval sprechen.
Anders sieht es bei den Metadaten aus, welche alle die folgende Syntax und damit Struktur besitzen:

(Name Inhalt)

Es liegt jedoch immer noch ein Information Retrieval Problem vor, da der Begriff auch die Suche in teilweise strukturierten (engl. semistructured) Dokumenten einschließt ([CDM08], S.1-2).
Wobei hierbei angemerkt sei, dass selbst die Metadaten nicht vollkommen strukturiert sind: Der Datentyp des Inhalts ist offen gelassen und es gibt keinerlei Vorgaben, welche Keywords in den Dokumenten auftreten müssen.
In den folgenden Kapiteln wird beschrieben, auf welche Weise ein Information Retrieval System konzipiert und realisiert werden kann, welches in der Lage ist, die Problemstellung 1.2 zu lösen.
Hierzu werden zunächst die hierfür benötigten Modelle boolesches Retrieval (Kapitel 4) sowie das Vector Space Model (Kapitel 5) erläutert.

3

Grundbegriffe

Unabhängig vom jeweiligen Modell gibt es einige Grundbegriffe, welche im Zusammenhang mit Information Retrieval Verfahren immer wieder auftauchen und die darum vorab vorgestellt werden sollen.

3.1 Anfrage

Eine Anfrage (engl. query) ist das, was der Anwender in den Computer eingibt, um sein Informationsbedürfnis zu befriedigen ([CDM08], S.5).
Anfragen können je nach Information Retrieval System vollkommen unterschiedlich strukturiert sein. Wichtig ist, dass der Nutzer weiß wie er seine Anfrage stellen muss, um mehr über das gewünschte Thema zu erfahren, was bei booleschen Modellen (siehe Kapitel 4) recht komplex werden kann.

3.2 Indexierung

Damit Dokumente eines Archivs von Information Retrieval Modellen verarbeitet werden können, müssen diese in einzelne Einheiten zerlegt werden, wobei jede Einheit mit einem eindeutigen Index versehen sein muss, um hinterher mit einem schnellen Zugriff abrufbar zu sein.
Zudem müssen auch die Dokumente selbst mit einem Index versehen werden, genannt *docID* (kurz für *document identification*), um auf deren enthaltende Einheiten schnell zugreifen zu können. Dabei handelt es sich meist um einen ganzzahligen Wert ([CDM08], S.7).
Dieses Vorgehen wird als Indexierung bezeichnet und ist unabdingbar, da ansonsten für jede Anfrage erneut über die gesamten Dokumentinhalte iteriert werden müsste, was ineffizient und für den Nutzer unzumutbar langsam wäre ([CDM08], S.3).

3.2.1 Term und Vokabular

Die indizierten Einheiten, in welche die Dokumente zerlegt werden, sind unter dem Begriff Term bekannt ([CDM08], S.3).
Terme sind im häufigsten Fall einfach Wörter eines Textes, dies muss jedoch nicht zwangsläufig der Fall sein.
Manche Systeme reduzieren Wörter beispielsweise auf deren Stammformen, um ähnliche Wörter zu einem einzigen Term zusammenzufassen. Alternativ lassen sich Wörter neben dem Wortstamm auch auf ihre grammatikalische Grundform reduzieren. Das Reduzieren der Wörter wird allgemein als Lemmatisierung oder Stemming bezeichnet.
Auf diese Weise müssen weniger Terme verwaltet werden, was den Speicherbedarf reduziert. Außerdem können leichter ähnliche Dokumente gefunden werden, da auch zum Suchbegriff verwandte Wörter zu einem Treffer führen ([Fer03], S.40-41).
Die Menge aller unterschiedlichen Terme eines Archivs wird als Vokabular bezeichnet ([CDM08], S.6).

3.2.2 Stopwords

Nicht jedes Wort wird bei der Indexierung zu einem Term: Handelt es sich um sehr häufig auftretende und zum Sinn des Textes wenig beitragende Wörter, wie z.B. „und“ oder „dann“, können diese wegfallen, um Speicherplatz zu sparen ([Fer03], S.37). Außerdem kommen durch ignorieren unwichtiger Terme weniger Dokumente infrage, was die Suche beschleunigt.

4

Boolesches Retrieval

Dieses Kapitel stellt das klassische Information-Retrieval-Verfahren boolesches Retrieval (engl.Boolean Retrieval) vor.

4.1 Eigenschaften des Verfahrens

Boolesches Retrieval überprüft Dokumente darauf, ob eine bestimmte Bedingung zutrifft.
Somit gibt es nur die Unterteilung in passende Dokumente und solche, welche diese Bedingung nicht erfüllen. Eine darüber hinausgehende Bewertung der Ergebnisse findet nicht statt, was zu einer ungeordneten Ergebnismenge führt ([Fer03], S.33). Das fehlende Ranking der Ergebnisse ist ein häufiger Kritikpunkt des Verfahrens.

4.2 Funktionsprinzip

Boolesches Retrieval basiert auf Mengenoperationen. Deshalb werden Dokumente Mengen zugeordnet, die jeweils durch bestimmte Attribute charakterisiert sind.
Dokument bezeichnet die Einheit, auf welcher das Retrieval stattfindet. Ein Dokument kann deshalb eine kleine Textmemo, aber auch ein ganzes Buchkapitel sein ([CDM08], S.4).

4.2.1 Attribut

Ein solches Attribut ist eine Abbildung, welche jedem Dokument einen Wert für dieses Attribut zuordnet. Die Abbildung erzeugt somit Attribut-Wert-Paare, was in Formel 4.1 gezeigt wird.

$$t : D \to T, t(d) = t_i \tag{4.1}$$

Hierbei bezeichnet t die Abbildung (d.h. das Attribut), D die Menge aller Dokumente und T den Wertebereich des Attributs t.

Der Attributwert t_i mit $t_i \in T$ und $i \in \mathbb{N}$ wird durch die Abbildung t dem Dokument $d \in D$ zugeordnet.

4.2.2 Anfragen

Elementare boolesche Anfrage

Ein Attribut-Wert-Paar wird auch als elementare boolesche Anfrage bezeichnet. Bei der elementaren booleschen Anfrage (t, t_1) werden zum Beispiel alle Dokumente gesucht, deren Attribut t den Wert t_1 annimmt.
Die Ergebnismenge D_{t,t_i} für eine Anfrage (t, t_i) kann demnach wie in Formel 4.2 charakterisiert werden.

$$D_{t,t_i} = \{d \in D | t(d) = t_i\} \tag{4.2}$$

Verknüpfung

Werden elementare Anfragen miteinander logisch verknüpft, so werden abhängig vom jeweiligen booleschen Operator bestimmte Mengenoperationen auf den Ergebnismengen der elementaren Anfragen ausgeführt.
Die möglichen booleschen Operatoren sind hierbei AND, OR und NOT.
(t, t_1) AND (s, s_1) bedeutet, dass alle Dokumente gesucht sind, bei denen sowohl $t(d) = t_1$ als auch $s(d) = s_1$ gilt. Die erforderliche Mengenoperation ist deshalb der Durchschnitt aus den beiden Ergebnismengen, was in Formel 4.3 gezeigt wird.

$$D_{t,t_1} \cap D_{s,s_1} \tag{4.3}$$

Wird hingegen der Operator OR verwendet, wird die Mengenoperation Vereinigung benötigt (siehe Formel 4.4), da alle Dokumente mit $t(d) = t_1$ oder $s(d) = s_1$ gesucht sind.

$$D_{t,t_1} \cup D_{s,s_1} \tag{4.4}$$

Außerdem kann der unäre Operator NOT verwendet werden, welcher das Komplement der Ergebnismenge erzeugt. Für die Anfrage NOT (t, t_1) muss erst die Menge aller Dokumente bestimmt werden, bei denen $t(d) = t_1$ zutrifft, um diese anschließend von der Gesamtmenge aller Dokumente abzuziehen. Dies wird in Formel 4.5 dargestellt.

$$D \setminus D_{t,t_1} \tag{4.5}$$

Da bei jeder Mengenoperation als Ergebnis neue Mengen entstehen, lassen sich hierauf erneut die oben beschriebenen Operatoren anwenden. Auf diese Weise können Anfragen beliebig tief geschachtelt werden ([Fer03], S.34).

4.3 Implementierungsansätze

Im folgenden Abschnitt werden die klassischen Implementierungsansätze vorgestellt, mit denen sich die soeben beschriebenen Operationen realisieren lassen.

4.3.1 Term-Inzidenz-Matrix

Eine mögliche Implementierung des booleschen Retrieval stellt die Umsetzung mittels einer Term-Dokument-Inzidenz-Matrix dar.
Dies bedeutet, dass die Zeilen der Matrix die Terme enthalten und die Spalten die Dokumente, was auch umgekehrt realisierbar ist. Genau betrachtet handelt es sich hier nicht um die Terme und Dokumente selbst, sondern deren Indizes.
Tritt Term t in Dokument d auf, so lautet der Eintrag für (t, d) der Matrix 1. Alle Einträge für nicht vorkommende Terme sind hingegen mit einer 0 versehen.
Abbildung 4.1 zeigt eine Beispielmatrix, wobei die tatsächliche Anzahl an Termen und Dokumenten in einer Sammlung weitaus größer ausfällt.
Eine Term-Inzidenz-Matrix verbraucht unnötig Speicherplatz, da sehr viele Einträge der Matrix eine 0 enthalten. Gerade bei sehr großen Sammlungen bzw. Dokumenten ist dies praktisch nicht realisierbar.

	1	2	3	4	5	6	7
Kontaktadresse	0	1	1	0	0	0	1
Seminar	1	0	1	0	1	0	0
Termin	1	1	1	0	0	0	0

Abb. 4.1. Term-Dokument-Inzidenz-Matrix. Die Zeile enthalten die Terme, die Spalten die Dokumente (hier durch deren *docID* repräsentiert). Alle Einträge mit einer 0 sind leere Einträge (Eigene Abbildung).

Verarbeitung einer Anfrage

Um eine Anfrage wie *Kontaktadresse AND Seminar AND Termin* mithilfe einer Matrix zu verarbeiten, werden einfach die entsprechenden Zeilen der Matrix genommen und bitweise logisch verknüpft, was für die obige Anfrage wie folgt aussieht:

0110001 AND
1010100 AND
1110000

0010000

Demnach wird das Dokument mit der *docID* 3 zurückgegeben. Analog funktioniert die *OR*-Verknüpfung:

0110001 OR
1010100 OR
1110000

1110101

Dieses Beispiel führt demnach zur Ergebnismenge $\{1, 2, 3, 5, 7\}$ ([CDM08], S.4).

4.3.2 Invertierte Liste

In der Regel werden zur Implementierung des booleschen Retrieval invertierte Listen verwendet ([Fer03], S.36).
Der Name basiert auf den darin gespeicherten invertierten Indizes. Diese werden deshalb als invertiert bezeichnet, weil sie vom Term zurück auf die Postion, in welcher der Term aufgetreten ist, schließen lassen.
In einer geeigneten Speicherstruktur, zum Beispiel einem Dictionary, werden zu jedem Term alle Dokumente gespeichert, in denen der Term auftritt.
Diese Dokumentlisten werden als invertierte Listen (engl. inverted lists) bezeichnet (siehe Abbildung Abbildung 4.2). Manche Implementierungen beinhalten neben dem Dokumentindex zusätzliche Informationen wie die genaue Wortpostion im Dokument.
Das Verfahren ermöglicht sehr schnelle Zugriffe, ist allerdings speicherintensiv ([Fer03], S.36). Im Vergleich zur Inzidenz-Matrix wird jedoch deutlich weniger Speicher benötigt, da die vielen leeren Einträge entfallen.

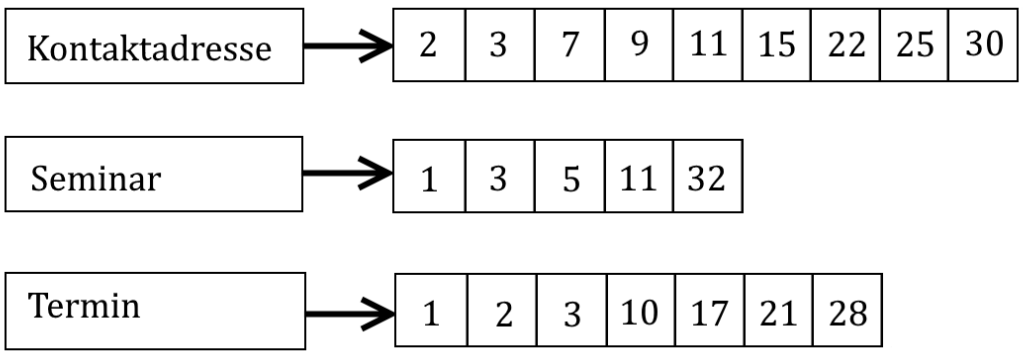

Abb. 4.2. Invertierte Listen zu Beispieltermen. Die Zahlen sind die Dokumentindizes oder auch *docID*s, in denen der jeweilige Term vorkommt (Eigene Abbildung).

Verarbeitung einer Anfrage

Hier stellt sich die Frage, wie denn nun eine boolesche Anfrage, wie in Abschnitt 4.2.2 beschrieben, mithilfe invertierter Listen umgesetzt werden kann.

Elementare Anfrage

Angenommen, es liegt eine elementare boolesche Anfrage in der Form (t, t_1) vor. In der Praxis ist meist nach dem Vorkommen eines bestimmtes Wortes gefragt. Demnach entspricht das Attribut t dem Term des gesuchten Wortes.
Da man auf dessen Vorkommen prüft, gelten für den Wertebereich $T = \{true, false\}$ und für den Attributwert $t_1 = true$.
Die Verarbeitung einer solchen elementaren Anfrage geht relativ einfach: Über den Index kann auf den Term schnell zugegriffen werden, vorausgesetzt dieser ist in der Sammlung enthalten. Trifft dies zu, kann einfach die gesamte zugehörige invertierte Liste als Resultat ausgegeben werden, da für alle enthaltenen Dokumente $t_1 = true$ gilt.

AND-Verknüpfungen

Wie sieht nun die Verarbeitung aus, wenn mehrere Anfragen miteinander verknüpft werden? Hierzu wird zunächst der AND - Operator betrachtet. Eine Anfrage liegt dann in der Form (t, t_1) AND (s, s_1) vor, wie etwa bei dem Beispiel *Kontaktadresse AND Seminar*, wobei gilt $t = Kontaktadress, t_1 = true$ sowie $s = Seminar, s_1 = true$.
Demnach werden alle Dokumente gesucht, in denen beide Terme auftauchen. Dazu wird der Durchschnitt aus den zugehörigen invertierten Listen gebildet. Betrachtet man die Abbildung 4.2, so ist der Durchschnitt für $D_{t,t_1} \cap D_{s,s_1}$ bzw. für $Kontaktadress \cap Seminar$ gleich der Ergebnisliste $\{3, 11\}$.

OR-Verknüpfungen

Lautet die Anfrage hingegen (t, t_1) OR (s, s_1) bzw. *Kontaktadresse OR Seminar*, so sind alle Dokumente gesucht, in denen entweder t_1 oder s_1 oder auch beide Terme vorkommen.
Gesucht ist also die Vereinigung $D_{t,t_1} \cup D_{s,s_1}$ bzw. $Kontaktadress \cup Seminar$. Dies ist die Vereinigung der invertierten Listen beider Terme. Im Falle des Beispiels 4.2 lautet die Ergebnisliste für die Anfrage *Kontaktadresse OR Seminar* $\{1, 2, 3, 5, 7, 9, 11, 15, 22, 25, 30, 32\}$.
Für beide Listenoperationen gilt, das die Indizes in den Ergebnislisten sortiert und Duplikate entfernt werden ([CDM08], S.11).

AND NOT-Verknüpfung

Da es sich bei *NOT* um einen unären Operator handelt, könnte dieser theoretisch alleine auftreten.
Eine Anfrage der Form *NOT Seminar* kann sehr viel Laufzeit kosten, wenn die Sammlung aus vielen Dokumenten besteht: Es muss über die gesamte Sammlung iteriert werden und für jedes Dokument geprüft werden, ob es in der invertierten Liste für den Term Seminar auftaucht. Der alleinstehende NOT-Operator ist deshalb so ineffizient, dass er bei den meisten booleschen Retrieval Systemen nur im Zusammenhang mit einem binären Operator zugelassen ist.
Da die Kombination *OR NOT* keinen Sinn macht, wenn man einen Term ausschließen möchte, ist dies in der Regel *AND*.
Hierbei wird aus den beiden Listen die Differenz gebildet. Lautet die Anfrage beispielsweise *Kontaktadresse AND NOT Seminar*, so werden aus der Ergebnisliste für Kontaktadresse alle Elemente entfernt, die in der Liste für Seminar enthalten sind ([Hen08], S.174).

Komplexe Ausdrücke

Da sowohl Vereinigung als auch Durchschnitt eine neue Ergebnisliste liefern, kann auf dieser wiederum jeder Operator angewandt werden, was eine beliebig tiefe Schachtelung erlaubt. Dieser Abschnitt erklärt, wie komplex geschachtelte Ausdrücke verarbeitet werden.

Im Falle von mehreren *AND*-Operatoren, wie etwa in der Suchanfrage *Kontaktadresse AND Seminar AND Termin*, ist es effizient, zunächst die einzelnen invertierten Listen aufsteigend nach deren Länge zu sortieren und dann von links nach rechts zu verarbeiten, indem das nachfolgende *AND* auf die Ergebnisliste des vorherigen Durchschnitts angewandt wird:
(*Seminar AND Termin*) *AND Kontaktadresse*
Auf diese Weise werden die Listen, über die iteriert werden muss, möglichst klein gehalten. Besitzt die kleinste Liste beispielsweise die Länge eins, dann kann nach der ersten Iteration bereits abgebrochen werden, da zulässige Lösungen in allen drei Listen vorkommen müssen.

Bei mehreren *OR*-Operatoren werden die Ausdrücke analog von links nach rechts verarbeitet, wobei die Sortierung nach Länge hierbei keinen Vorteil bringt, da bei der Vereinigung zweier Listen ohnehin über alle Elemente iteriert werden muss. Die Verarbeitung würde demnach in der folgenden Reihenfolge erfolgen:
(*Kontaktadresse OR Seminar*) *OR Termin*

Ist die Anfrage hingegen gemischt, wie etwa in (*Kontaktadresse OR Seminar*) *AND* (*Termin OR Seminar*), werden erst die inneren Ausdrücke ausgewertet und dann aus deren Ergebnislisten der Durchschnitt gebildet ([CDM08], S.11).

Die Verarbeitung mehrerer Wörter

Boolesches Retrieval kann auch mehrere zusammengehörende Wörter verarbeiten. Über die interne Verarbeitung besteht hierbei jedoch für den Nutzer kein Einblick: Das Information Retrieval System kann so realisiert sein, dass es die aus den Wörtern der Anfrage isolierten Terme mit *OR* veknüpft, es kann diese jedoch genauso gut mit*AND* verbinden ([Hen08], S.171).

5

Das Vektorraummodell

Dieses Kapitel stellt ein weiteres klassisches Information Retrieval Verfahren, das Vektorraummodell, vor.

5.1 Funktionsprinzip

Zunächst werden alle Dokumente in Terme zerlegt und indexiert, wie es auch beim booleschen Retrieval der Fall ist. Die Ermittlung des Vokabulars ist Grundvoraussetzung für alle weiteren Schritte.
Wie der Name bereits nahelegt, basiert das Verfahren auf Vektoren.

Die Grundidee besteht darin, für jedes Dokument sowie für die Anfrage einen reellen Vektor zu erstellen und anschließend zu ermitteln, welche Dokumentvektoren am ähnlichsten zum Anfragevektor sind.
Jeder Vektor besitzt hierbei die Länge des Vokabulars, da er die Gewichte aller Terme enthält. Die Bedeutung des Gewichts wird in Abschnitt 5.4 erklärt.
Im Gegensatz zum booleschen Retrieval können die Resultate des Vektorraummodells basierend auf dem Grad der Ähnlichkeit in eine Rangfolge gebracht werden ([Fer03], S.62-63).

5.2 Vektor und Vektorraum

Da das Funktionsprinzip des Modells auf Vektoren basiert, seien an dieser Stelle die zum Verständnis notwendigen Begriffe Vektorraum und Vektor erklärt.
Vektoren stellen Elemente eines Vektorraumes dar, darum ist es notwendig letzteres zuerst zu definieren ([Jän84], S.17).
Ein solcher Vektorraum lässt sich wie in Definition 5.1 ([Jän84], S.22) angegeben beschreiben.

Definition 5.1. *(Vektorraum)*
Ein Tripel $(V, +, \cdot)$*, bestehend aus einer Menge* V*, einer Abbildung (genannt Addition)*
$+ : V \times V \to V$
$(x, y) \to x + y$
und einer Abbildung (genannt skalare Multiplikation)
$\cdot : \mathbb{R} \times V \to V$
$(\lambda, x) \to \lambda x$
heißt reeller Vektorraum*, wenn für die Abbildungen* $+$ *und* $\cdot$ *die folgenden acht Axiome gelten:*

(1) $(x + y) + y = x + (y + z) \quad \forall x, y, z \in V$
(2) $x + y = y + x \quad \forall x, y, z \in V$
(3) *Es gibt ein Element* $0 \in V$ *(genannt „Null "oder„Nullvektor ") mit* $x + 0 = x \quad \forall x \in V$
(4) *Zu jedem* $x \in V$ *gibt es ein Element* $-x \in V$ *mit* $x + (-x) = 0$
(5) $\lambda(\mu x) = (\lambda\mu)x \quad \forall \lambda, \mu \in \mathbb{R}, x \in V$
(6) $1x = x \quad \forall x \in V$
(7) $\lambda(x + y) = \lambda x + \lambda y \quad \forall \lambda \in \mathbb{R}, x, y \in V$
(8) $(\lambda + \mu)x = \lambda x + \mu x \quad \forall \lambda, \mu \in \mathbb{R}, x \in V$

Hieraus ergibt sich die Frage, wie Addition und skalare Multiplikation bei Vektoren definiert sind, was in Definition 5.2 und 5.3 ([Jän84], S.18) gezeigt wird.

Definition 5.2. *(Addition)*
Sind $(x_1, ..., x_n)$ *und* $(y_1, ..., y_n)$ *n-Tupel reeller Zahlen, so werde deren Summe durch*
$(x_1, ..., x_n) + (y_1, ..., y_n) = (x_1 + y_1, ..., x_n + y_n)$
erklärt.

.

Definition 5.3. *(Skalare Multiplikation)*
Ist $\lambda \in \mathbb{R}$ *und* $(x_1, ..., x_n) \in \mathbb{R}^n$*, so erklären wir* $\lambda(x_1, ..., x_n) = (\lambda x_1, ..., \lambda x_n) \in \mathbb{R}^n$

.

Ein Vektor $\overrightarrow{v} \in V$ ist demnach ein Element des Vektorraums $(V, +, \cdot)$, wenn Addition und skalare Multiplikation die Axiome (1) - (8) aus Definition 5.1 erfüllen. Zwei Vektoren mit unterschiedlich vielen Elementen, z.B. $\overrightarrow{v_1} = (1, 2)$ und $\overrightarrow{v_2} = (1, 2, 3)$ liegen deshalb nicht im selben Vektorraum, weil sie sich weder addieren noch multiplizieren lassen.

5.3 Definition Vektorraummodell

Das soeben beschriebene Funktionsprinzip lässt sich mathematisch mithilfe von Attributen beschreiben.

Attribute stellen im Vektorraummodell eine Abbildung der Dokumentenmenge D auf die reellen Zahlen $\mathbb{R}$ dar. Damit ist der Wertebereich der Attribute im Gegensatz zum booleschen Retrieval eindeutig festgelegt.
Die Definition lautet somit wie folgt:

Definition 5.4. *(Vektorraummodell mit Attributen)*

Sei $D = d_1, ..., d_n$ eine Menge von Dokumenten oder Objekten und $A = A_1, ..., A_n$ eine Menge von Attributen $A_j : D \to R$ auf diesen Objekten. Die Attributwerte $A_j(d_i) =: w_{i,j}$ des Dokuments d_i lassen sich als Gewichte auffassen und zu einem Vektor $w_i = (w_{i,1}, ..., w_{i,n}) \in \mathbb{R}^n$ zusammenfassen. Dieser Vektor beschreibt das Dokument im Vektorraummodell: Er ist seine Repräsentation und wird Dokumentvektor genannt.
Eine Anfrage wird durch einen Vektor $q \in \mathbb{R}^n$ mit Attributwerten, den Anfragevektor oder Query-Vektor, dargestellt.
Eine Ähnlichkeitsfunktion $s : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ definiere für je zwei Vektoren $x, y \in \mathbb{R}^n$ einen reellen Ähnlichkeitswert $s(x, y)$.

Diese Definition ist aufgrund der Beschreibung durch Attribute sehr allgemein gehalten. Es ist deshalb nicht definiert, welche Einheiten des Dokuments gewichtet werden. Demnach sind theoretisch auch andere Dokumentformate wie etwa Bilder mit Pixeln bzw. Pixelgruppen als Attributen möglich ([Fer03], S.63).
Praktisch gesehen machen im Rahmen dieser Arbeit jedoch andere Formate als Texte keinen Sinn. Darum wird im folgenden davon ausgegangen, dass ausschließlich Terme gewichtet werden. Demnach lässt sich die Attributmenge A spezifisch auf die Problemstellung bezogen als Menge der Terme oder Vokabular T auffassen.

5.4 Gewichte

Bei Gewichten handelt es sich, wie bereits beschrieben, um reelle Zahlenwerte.
Ein Gewicht gibt die Wichtigkeit eines Terms basierend auf dessen statistischer Häufigkeit an ([CDM08], S.100).

5.4.1 Termhäufigkeit

Die Häufigkeit, mit der ein Term t in einem Dokument d auftritt, wird als Termhäufigkeit (engl. term frequency) bezeichnet. Demnach wird die Termhäufigkeit pro Vorkommen von t in d um eins erhöht ([CDM08], S.71).
Es erscheint intuitiv logisch, dass ein Text, der das gesuchte Wort mehrmals beinhaltet wichtiger sein muss als ein Dokument, in welchem der Begriff nur ein einziges mal auftaucht.
Dies Gewichtungsschema erlaubt eine viel genauere Differenzierung als eine simple Unterscheidung zwischen true und false, wie es beim booleschen Retrieval der Fall ist.

5.4.2 Dokumenthäufigkeit

Die Termhäufigkeit stellt für sich genommen schon eine mögliche Gewichtungsmethode dar, allerdings keine besonders gute: Die Bewertung alleine aufgrund der Termhäufigkeit lässt außer Acht, dass nicht alle Terme gleich wichtig sind.
Taucht ein Term beispielsweise in jedem Dokument auf, kann es nicht besonders aussagekräftig sein. Demnach ist es sinnvoll, zusätzlich zur Termhäufigkeit $tf_{t,d}$ auch die Dokumenthäufigkeit (engl. document frequency) df_t zu bestimmen.
Diese entspricht der Anzahl Dokumente in D, welche t enthalten. Um den Einfluss nicht aussagekräftiger Terme zu reduzieren, wird das Gewicht umso stärker verringert, je größer die Dokumenthäufigkeit ausfällt ([CDM08], S.108).

5.4.3 Invertierte Dokumenthäufigkeit

Zur Reduktion des Gewichts basierend auf der Dokumenthäufigkeit wird als reduzierender Faktor die Invertierte Dokumenthäufigkeit (engl. inverse document frequency) verwendet. Die invertierte Dokumenthäufigkeit idf_t des Terms t berechnet sich wie in Formel 5.1 gezeigt.

$$idf_t = \frac{1}{df_t} \tag{5.1}$$

Oftmals werden modifizierte Formen verwendet, um die großen Werte seltener Terme durch den Logarithmus wieder zu dämpfen ([Fer03], S.68-69).
Formel 5.2 zeigt ein Beispiel für eine solche modifizierte invertierte Dokumenthäufigkeit. In der Regel beträgt die Basis des Logarithmus 10, dies spielt aber letztendlich für das korrekte Ranking der Resultate keine Rolle ([CDM08], S.108-109).

$$idf_t = log\frac{N}{df_t} \tag{5.2}$$

5.4.4 TF-IDF-Gewichtung

Die vollständige Methode zur Gewichtung einzelner Terme kombiniert Termhäufigkeit und invertierte Dokumenthäufigkeit, indem erstere mit letzterer multipliziert wird.
Alle Formeln dieses Typs werden als TF-IDF Gewichtung (engl. tf-idf weighting) bezeichnet ([Fer03], S.71).
Das Gewicht für Term t in Dokument d berechnet sich somit wie in Formel 5.3 gezeigt ([CDM08], S.109).

$$tf - idf_{t,d} = tf_{t,d} \times idf_t, \tag{5.3}$$

Verwendet man für die invertierte Dokumenthäufigkeit den unmodifizierten Wert 5.1, so ergibt sich hieraus die Berechnung:

$$tf - idf_{t,d} = \frac{tf_{t,d}}{df_t} \tag{5.4}$$

Für die modifizierte Formel 5.2 lautet die TF-IDF-Gewichtung wie folgt:

$$tf - idf_{t,d} = tf_{t,d} \times (log \frac{N}{df_t}) \tag{5.5}$$

Sei T die Menge aller Terme der Sammlung bzw. das Vokabular, dann enthält der Gewichtsvektor w_i zu einem Dokument $d_i \in D$ für jeden Term $t_j \in T$ dessen Gewicht $w_{i,j} = tf - idf_{j,i}$, sodass $w_i = (tf - idf_{1,i}, ..., tf - idf_{n,i})$ gilt.

5.5 Anfragen

Beim Vektorraummodell gibt es keine booleschen Operatoren zur Verknüpfung von Termen, weshalb Anfragen in Freitextform gestellt werden. Diese Form wird auch in der Websuche verwendet und ist darum sehr bekannt.
Da die Reihenfolge von Wörtern weder bei Anfragen noch in den Dokumenten eine Rolle spielt, lassen sich Anfragen einfach als eine Menge von Wörtern bzw. als die daraus resultierende Menge von Termen betrachten.
Ein solches Modell, das lediglich die Anzahl, nicht aber die Reihenfolge von Wörtern berücksichtigt, wird auch als *bag of words model* bezeichnet.
Da für jeden Term ein anderer Ähnlichkeitswert erzielt wird, werden die Ähnlichkeitswerte aller in der Menge enthaltenen Terme addiert, sodass pro Dokument ein Gesamtwert berechnet wird ([CDM08], S.107).
Anfragetexte werden genau wie Dokumente behandelt und die Vektoren wie in Abschnitt 5.4.4 beschrieben bestimmt ([Fer03], S.82).

5.6 Ähnlichkeitsfunktion

Grundidee des Vektorraummodells ist das Ermitteln der Ähnlichkeit zwischen Vektoren. Deshalb muss hierfür eine geeignete Ähnlichkeitsfunktion gefunden werden.

5.6.1 Euklidischer Abstand

Eine typische Distanzfunktion für Vektoren ist der euklidische Abstand, bei dem die Differenz wie in Formel 5.6 gezeigt berechnet wird ([KMOB14],S.132).

$$d(\overrightarrow{x}, \overrightarrow{y}) = \sqrt{\sum_{i=1}^{M} (x_i - y_i)^2} \tag{5.6}$$

Allerdings besitzt der euklidische Abstand den gravierenden Nachteil, dass die Länge der Vektoren eine Rolle spielt.
Liegen zwei Dokumente $d1$ und $d2$ vor und $d2$ besitzt den Inhalt von $d1$ zweimal aneinandergereiht, so wird $d2$ als ähnlicher eingestuft, aus dem einzigen Grund weil die Termhäufigkeit doppelt so groß ist und der Vektor damit die doppelte Länge hat.
Das Problem, dass zwei unterschiedlich lange Dokumente, in denen die gesuchten Terme etwa gleich verteilt sind, dennoch vollkommen verschiedene Ähnlichkeitswerte erzielen macht den euklidischen Abstand zu einer ungeeigneten Ähnlichkeitsfunktion.

5.6.2 Cosinus-Maß

Um den Einfluss der Vektorlänge zu eliminieren, wird in der Regel das Cosinus-Maß verwendet.
Das Cosinus-Maß ist das Skalarprodukt der normalisierten Vektoren ([CDM08], S.112). Ein normalisierter Vektor besitzt immer die Länge 1, da Ursprungsvektor durch die euklidische Länge dividiert wird.
Zur Berechnung des Cosinus-Maßes müssen somit sowohl das Skalarprodukt als auch die Längenberechnung eines Vektors bekannt sein, darum werden beide an dieser Stelle erklärt.

Euklidische Länge

Sei $\overrightarrow{x}$ ein Vektor, dann wird seine euklidische Länge wie in Formel 5.7 angegeben berechnet.

$$|\overrightarrow{x}| = \sqrt{\sum_{i=1}^{M} x_i^2} \tag{5.7}$$

Skalarprodukt

Das Skalarprodukt zweier Vektoren $\overrightarrow{x}$ und $\overrightarrow{y}$ wird wie in Formel 5.8 gezeigt berechnet.

$$\overrightarrow{x} \cdot \overrightarrow{y} = \sum_{i=1}^{M} x_i y_i \tag{5.8}$$

Funktion

Das Cosinus-Maß multipliziert die Vektoren und dividiert sie durch deren Länge, sodass die Ähnlichkeitsfunktion von der Länge unbeeinflusst ist, was in Formel 5.9 gezeigt wird ([CDM08], S.111).

$$sim(\overrightarrow{x}, \overrightarrow{y}) = \frac{\overrightarrow{x} \cdot \overrightarrow{y}}{|\overrightarrow{x}| \cdot |\overrightarrow{y}|} \tag{5.9}$$

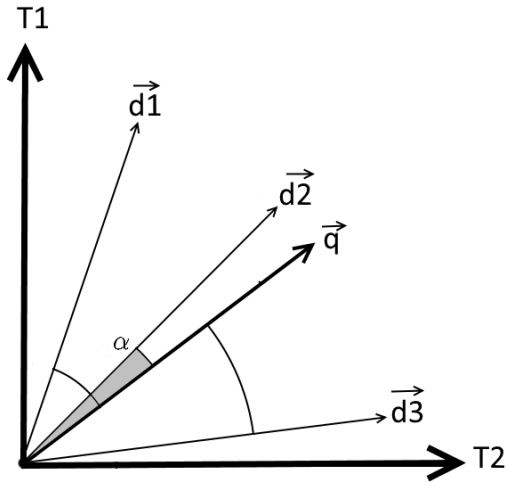

Abb. 5.1. Vektorraum mit den Termen $T1$ und $T2$ als Achsen, drei Dokumentvektoren und dem Anfragevektor. Das Cosinus-Maß liefert als ähnlichstes Dokument d_2 (Eigene Abbildung, basierend auf [SB10], S.55).

Zum besseren Verständnis des Cosinus-Maßes zeigt die Beispielabbildung 5.1 einen Vektorraum mit zwei Termen als Achsen, sodass der Raum sich zweidimensional darstellen lässt. In der Realität gibt es meist weitaus mehr Achsen, da das Vokabular tausende Terme beinhalten kann.
Der Vektorraum beinhaltet insgesamt drei Dokumentvektoren $\overrightarrow{d_i}$ sowie den Anfragevektor $\overrightarrow{q}$ als Elemente, wobei alle Vektoren normalisiert sind.
Unter Verwendung des Cosinus-Maßes ergibt sich für $sim(\overrightarrow{d_2}, \overrightarrow{q}) = cos(\alpha)$ der höchsten Wert, da α der kleinste aller Winkel zwischen einem Dokumentvektor $\overrightarrow{d_i}$ und $\overrightarrow{q}$ ist. Hier wird deutlich, dass lediglich der Winkel, nicht aber die Länge relevant ist.
Damit wird $d2$ als erstes Ergebnisdokument ausgegeben ([SB10], S.55-56).

6

Bewertung eines Information Retrieval Systems

Dieses Kapitel soll erläutern, auf welche Weise sich Information Retrieval Systeme bewerten und vergleichen lassen.

Nachdem bereits zwei klassische Verfahren zur Realisierung von Information Retrieval Systemen vorgestellt wurden, liegt es nahe, nach einer Methode zum Bewerten und Vergleichen solcher System zu suchen.
Die Bewertungsmethode muss hierbei geeignet sein, verschiedenste Systeme zu bewerten, denn diese können sich in zahlreichen Punkten wie etwa Dokumentformat, Dokumentrepräsentation, der Art und Weise, wie Anfragen formuliert und Ergebnisse präsentiert werden, unterscheiden ([Fer03], S.84).

6.1 Problem Relevanz

Um ein Information Retrieval System auf dessen Qualität hin zu beurteilen, muss die Relevanz der zurückgelieferten Ergebnisse eingestuft werden können.
Hier eröffnet sich das Hauptproblem: Wann ist ein Dokument relevant? Allgemein formuliert lässt dich dies so beantworten: Ein Dokument ist relevant, wenn es das in der Anfrage formulierte Informationsbedürfnis befriedigt ([Fer03], S.85).
Der Begriff Relevanz lässt sich zudem mathematisch wie in Definition 6.1 ([Fer03], S.86) angegeben definieren.

Definition 6.1. *(Relevanz)*
Die Relevanz eines Dokuments für eine Anfrage ist eine Relation $r : D \times Q \to R$*, wobei* $D = D_1, ..., d_m$ *die Menge der Dokumente,* Q *die Menge der Anfragen und* R *eine Menge von Wahrheitswerten, im Allgemeinen die Menge* $0, 1$*, ist. (Im Folgenden wird* $R = \{0, 1\}$ *angenommen, wenn nichts anderes gesagt wird.)*
Die Relation r *wird im Allgemeinen durch Befragen von Experten zu konkreten Anfragen und Dokumentmengen ermittelt und als Tabelle oder in Form von Listen gespeichert.*

Die Definition lässt sofort erkennen, dass Relevanz stets von der subjektiven Wahrnehmung eines Anwenders abhängig ist: Jeder Nutzer entscheidet für sich selbst, ob er ein Dokument als relevant einstuft.

6.2 Precision und Recall

In diesem Abschnitt werden die Evaluierungsmaße Precision und Recall vorgestellt.

6.2.1 Precisison

Precision, was übersetzt Präzision bedeutet, bezeichnet den Anteil relevanter Dokumente unter den zurückgelieferten Dokumenten. Formel 6.1 beschreibt die Berechnung, wobei # für die Kardinalität der Menge, d.h. die Anzahl darin enthaltener Elemente, steht.

$$Precision = \frac{\#(relevant\ \ items\ \ retrieved)}{\#(retrieved\ \ items)} \tag{6.1}$$

6.2.2 Recall

Recall kann mit Trefferquote übersetzt werden und beschreibt die Frage, wie viele der relevanten Dokumente tatsächlich vom System zurückgeliefert wurden, was in Formel 6.2 gezeigt wird ([CDM08], S.142-143).

$$Recall = \frac{\#(relevant\ \ items\ \ retrieved)}{\#(relevant\ \ items)} \tag{6.2}$$

6.2.3 Veranschaulichung

Die Bedeutung der Maße lässt sich leichter anhand der Kontingenztafel 6.1 nachvollziehen.
Kontingenztafeln sind Häufigkeitstabellen und stammen aus der Statistik. Sie beschreiben die gemeinsame Verteilung zweier Merkmale, in diesem Fall Relavanz und Rückgewinnung ([Eng14]).

Tabelle 6.1. Kontingenztafel ([CDM08], S.143)

	relevant	irrelevant
zurückgelifert	true positives (tp)	false positives (fp)
nicht zurückgeliefert	false negatives (fn)	true negatives (tn)

Precision lässt sich anhand der Tabelle wie folgt beschreiben:

$$Precision = \frac{tp}{tp + fp} \tag{6.3}$$

Präzision berechnet also, wie viele der als positiv eingestuften Ergebnisse auch tatsächlich *true postives*, also relevant, sind.

Die Beschreibung für Recall lautet wie folgt:

$$Recall = \frac{tp}{tp + fn} \tag{6.4}$$

Demnach bestimmt der Recall, wie viele aller relevanten Ergebnisse auch als positiv eingestuft wurden und nicht als *false negatives* verloren gingen.

6.3 Zwei Evaluierungsmaße

Um ein System hinreichend bewerten zu können, reicht eines der Maße nicht aus. Beispielsweise könnte ein System einen Recall von 100% erreichen, indem es einfach alle Dokumente zurückliefert.
Umgekehrt lässt sich auch eine Precision von 100% erreichen, wenn nur ein einziges Dokument gefunden wurde und dieses ein Treffer war. Vielleicht wurden allerdings eine ganze Reihe weiterer relevanter Dokumente nicht gefunden.
Deshalb werden stets beide Maße für eine qualitative Einschätzung eines Information Retrieval Systems benötigt.

6.4 Durchführung

Eine Bewertung anhand der oben aufgeführten Evaluierungsmaße kann durchgeführt werden, wenn folgende Voraussetzungen gegeben sind ([CDM08], S.140):

- Vorgegebene Dokumentsammlung
- Feste Menge von Test-Anfragen
- Eine Relation r, das jedem Anfragen-Dokument Paar einen Wert $\in \{0, 1\}$ für relevant bzw. irrelevant zuordnet.

Leider sind für diese Arbeit jedoch die oben gelisteten Voraussetzungen nicht gegeben.
Die Erzeugung einer repräsentativen Test-Dokumentsammlung ist für die Problemstellung nicht möglich, da die Art des Dokumentarchivs offen gehalten wurde.
Auch mit Beschränkung auf den Anwendungsfall E-Mails wäre in jedem Fall die zu erzeugende Anfragen-Menge zu groß, um sie im Rahmen dieser Arbeit bewältigen zu können, da Anfragen aus beliebigen Wörtern bestehen können und zudem beliebig tief schachtelbar sind.
Aufgrund dieser Punkte musste auf eine Bewertung des Systems mittels Precision und Recall in dieser Arbeit verzichtet werden.

7

Implementierung

In diesem Kapitel wird beschrieben, auf welche Weise das Information Retrieval System dieser Arbeit in der Programmiersprache Lisp realisiert wurde.

7.1 Teilweise strukturierte Dokumente

Besonderheit der Problemstellung ist das Vorliegen der Dokumente in semistrukturierter Form (siehe 1.2).
Dies bedeutet, dass zwei unterschiedliche Teilprobleme zu lösen sind: Zum einen die Keywordsuche, welche sich auf die Suche in strukturierten Metadaten bezieht und zum anderen die Freitextsuche. Es liegt nahe, beides getrennt zu lösen, da die Suchen unterschiedliche Anforderungen besitzen.
Bevor erklärt wird, wie die beiden Verfahren jeweils realisiert wurden, ist es wichtig, zunächst eine Vorstellung zu haben, wie die zu durchsuchenden Dokumente des Archivs beschaffen sind, weshalb Abbildung 7.1 ein Beispiel zeigt.

Listing 7.1. Beispieldokument

```
1
2
3  (absender ("<MaxMuster@muster-mail.de>"))
4  (Betreff (" Umfrage"))
5  (datum ("Wed, 22 Jun 2017 07:47:51 +0200"))
6  (anzahlAnhaenge 0)
7  (Termin nil)
8  (ABSENDER "Max_Muster")
9
10 (ABSENDER-MAIL-ADRESSE "MaxMuster@muster-mail.de")
11 (EMPFAENGER ("doe>> John Doe"))
12 (EMPFAENGER-MAIL-ADRESSEN ("johnd@muster-mail.de"))
13 (BETREFF "Umfrage")
14 (EMAIL-TYP "sent")
15 (QUELLBOXART "SENT")
16
17 Hallo John,
18
19 ich werde dir die Umfragenformulare schnellstmöglich per Post
20 zukommen lassen.
21
22
23 Viele Grüße
24 Max Muster
```

In der Praxis enthalten die Dokumente oft weitaus mehr Keywords, deren Inhalt sich auch über mehrere Zeilen erstrecken kann, sowie Kommentare, welche vom System als Freitext interpretiert werden.
In diesem Beispiel handelt es sich zwar um eine E-Mail, die Keywords können jedoch inhaltlich vollkommen unterschiedlich ausfallen. Allen Dokumenten gemeinsam ist die grundlegende Struktur aus 7.2.

Listing 7.2. Dokumentstruktur

```
1 (Keywordname_1 Inhalt)
2 ....
3 (Keywordname_n Inhalt)
4
5 Freitext
```

7.2 Vorverarbeitung

7.2.1 Erstellen des Dokument-Dictionaries

Zunächst wird das Dokument-Dictionary[1] des Archivs erstellt:
Zu jedem Dokument werden die folgenden Punkte erfasst und in einer Hash-Table gespeichert:

- *docID*: Jedes Dokument erhält eine einmalige ID in Form eines fortlaufenden Integer-Wertes. Diese ID ist der Key für dieses Dokument in der Hash-Table.
- Dateipfad
- Dokumentvektor (zu Beginn noch nicht initialisiert). Dateipfad und Dokumentvektor bilden in Form eines Structs den Hasheintrag zur *docID*
- Datum: Optionale Information, die zur verbesserten Ergebnisanzeige dient
- Absender: ebenfalls optionale Information.

Die letzten beiden Punkte können entfallen, da nicht alle Dokumente diese Metadaten beinhalten, insbesondere falls es sich nicht um E-Mails handelt. Das System wurde flexibel gehalten, um auch andere Dateien verarbeiten zu können.
Abbildung 7.1 zeigt den Aufbau eines Hash-Eintrags für das Dokument-Dictionary.

7.2.2 Verarbeiten der Dokumente

Anschließend wird über alle Dokument-Einträge iteriert, um die folgenden Schritte auszuführen:

- Aufteilen in Keywords und Freitext
- Verarbeiten der Keywords
- Verarbeiten des Freitextes

Es stellt sich die Frage, was in den letzten beiden Punkten geschieht.

[1] Auch wenn hier semantisch von einem Dictionary gesprochen wird, ist die zugrundeliegende Datenstruktur in Lisp eine Hash-Table.

Abb. 7.1. Struktur für einen Eintrag im Dokument-Dictionary. Die Einträge Datum und Absender können, falls nicht existent, auch leer bleiben. Im rot markierten Bereich werden später die Dokumentvektoren eingetragen. (Eigene Abbildung).

Verarbeiten der Keywords

Für die Keywords bietet sich boolesches Retrieval an, da ein Keyword entweder auftreten kann oder nicht.
Allerdings muss zusätzlich noch geprüft werden, ob der Inhalt des Keywords mit der Anfrage übereinstimmt.
Um dies zu lösen, wurde sich für eine modifizierte Form invertierter Listen entschieden. Eine Term-Inzidenz-Matrix wurde hierbei sofort aufgrund des zu großen Speicherbedarfs ausgeschlossen.
Was in diesem Verarbeitungsschritt geschieht ist das Erstellen einer Hash-Table, welche die Keywordnamen als Keys besitzt und dazu, zusammengefasst in einem Struct[2], die folgenden Inhalte besitzt:

- docID: Dies ist der Index des Dokuments, welches das Keyword enthält und gehört zur standardmäßigen invertierten Liste.
- Typ: Dieser Eintrag gibt an, von welchem Typ der Inhalt ist, da dies über die Suche darin entscheidet.
- Inhalt: Enthält den Keywordinhalt. Dieser ist in der Regel so klein, dass er problemlos darin gespeichert werden kann und im Gegensatz zum Freitext keine weitere Verarbeitung erfordert. Dies würde nur unnötig Speicher belegen.

Zur Veranschaulichung der modifizierten invertierten Liste zeigt Abbildung 7.2 diese anhand einiger Beispiel-Keywords.
Das Speichern der Informationen in einem Struct bietet den Vorteil, dass auf diese über den Slot-Value gezielt zugegriffen werden kann, ohne über Listen iterieren zu müssen.

Verarbeiten des Freitextes

Für die Freitextsuche wurde das Vektorraummodell gewählt, da dieses Teiltreffer sowie ein Ranking der Ergebnisse ermöglicht.

Hierfür muss zunächst das Vokabular bestimmt werden, was wie folgt abläuft:

[2] Element der Programmiersprache Lisp, mit dem mehrere zusammengehörige Objekte zu einem zusammengefasst werden können

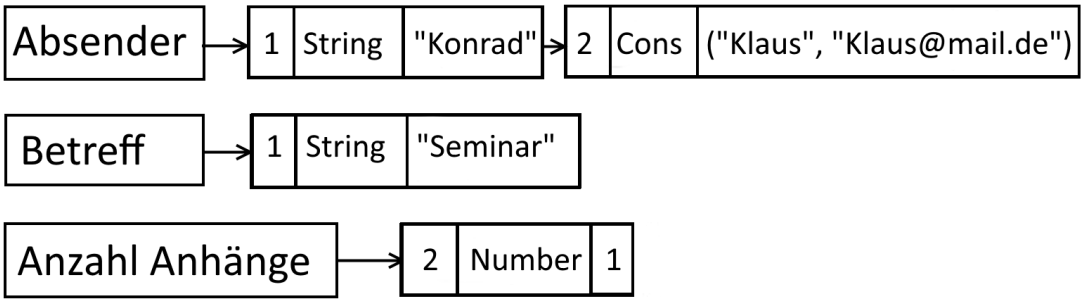

Abb. 7.2. Modifizierte invertierte Liste zur Realisierung der Keywordsuche mittels Structs der Form (docID, Typ, Inhalt) (Eigene Abbildung).

- Es wird über eine Liste aller Freitexte iteriert.
- Jeder Freitext wird in seine Terme zerlegt. Hierbei wurde auf eine Lemmatisierung verzichtet, da dies den Rahmen der Arbeit sprengen würde. Zur zeilenweise Zerlegung der Texte wurde das Package split-sequence verwenedet ([Ion16]).
- Für jeden Term wird geprüft, ob er schon bekannt ist oder nicht. Falls nicht, wird er der Vokabularliste hinzugefügt.

Nachdem alle Terme bekannt sind, werden diese indexiert:

- Es wird ein Term-Dictionary in Form einer Hash-Table angelegt.
- Für jeden Term der Vokabularliste wird ein Eintrag der Form (Index,idf=0) angelegt, wobei der Index ein fortlaufeneder Integer-Wert ist und der idf-Wert noch zu berechnen ist.

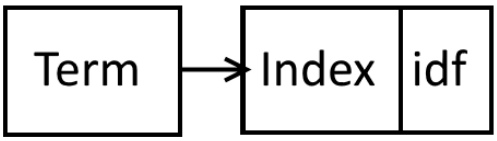

Abb. 7.3. Struktur für einen Eintrag im Term-Dictionary. Der Index gibt die Position im Dokumentvektor an (Eigene Abbildung).

Nun kann das Term-Dictionary mit Einträgen der in Abbildung 7.3 gezeigten Struktur gefüllt werden, wobei die folgenden Schritte beim Iterieren über die Freitexte ausgeführt werden:

- Anlegen eines Dokumentvektors pro Dokument, realisiert mittels Hash-Table, welche die Term-Indizes als Keys und deren tf-idf-Gewichte als Werte besitzt.
- Tritt ein Term zum ersten Mal in der Sammlung, wird der idf-Slot im Term-Dictionary auf 1 gesetzt.

- Tritt ein Term zum ersten Mal im Dokument auf und existiert bereits in der Sammlung, wird der idf-Wert im Term-Dictionary um 1 erhöht.
- Beim ersten Auftreten im Dokument wird der Eintrag im Dokumentvektor auf die Termhäufigkeit 1 gesetzt. Die Position im Vektor ist durch Zugriff auf den Index des Terms bekannt.
- Für jedes erneute Auftreten im Dokument wird die Termhäufigkeit um 1 erhöht.
- Der bereits angelegte Eintrag im Dokument-Dictionary kann an dieser Stelle mit dem Dokumentvektor initialisiert werden.

Noch enthalten die idf-Slots im Term-Dictionary die Dokumenthäufigkeiten statt der idf-Werte. Nun erfolgt deren Umrechnung in den idf-Wert nach Formel 5.2.
Anschließend können die tf-idf-Werte berechnet werden (siehe Formel 5.5) und in die Dokumentvektoren, welche bisher nur die Termhäufigkeiten enthielten, eingetragen werden.
Nun stehen die Dokumentvektoren fest und können zur Verrechnung mit dem Anfragevektor verwendet werden.
Durch das Realisieren der Vektoren als Hash-Tables gibt es keine leeren Einträge. Ein fehlender Eintrag wird als Gewicht 0 interpretiert, sodass dies kein Problem bei der Ähnlichkeitsberechnung darstellt.

7.3 Die Suche

7.3.1 Keywordsuche

Lautet die Anfrage beispielsweise *Absender* = *klaus*, wird auf das Keyword-Dictionary über den Hash-Key *Absender* zugegriffen. Anschließend wird über alle darin gespeicherten Structs iteriert, wobei zunächst der Typ des Inhalts abgefragt wird. Dieser entscheidet über die Art der Suche:

1. **String**: Der Keywordinhalt wird mit der vordefinierten Lisp-Funktion search durchsucht. Die Suche ist erfolgreich, wenn die gesuchte Zeichenkette an einer beliebigen Stelle im Keyword auftaucht.
2. **Number**: Ist der Inhalt eine Zahl, wird die Suchanfrage (die stets als String übergeben wird) wenn möglich zum Datentyp Number konvertiert. Hierbei sind ausgeschriebene Zahlen von null bis zwölf auch konvertierbar. Ist kein Konvertieren möglich, schlägt die Suche sofort fehl, da der Inhalt nicht zur Anfrage passen kann.
3. **Liste**: Eine Liste (Datentyp Cons in Lisp) wird rekursiv durchsucht, um alle darin enthaltenen Elemente typspezifisch zu durchsuchen, d.h. darin enthaltene Strings, Zahlen und Unterlisten.

Liegt ein Treffer vor, wird die*docID* als Resultat der Ergebnisliste hinzugefügt.
Hierbei kann es sein, dass pro Teilanfrage mehrere Keywordsuchen durchgeführt werden, da der Nutzer die Anfrage für verschiedene Keywords gleichzeitig stellt.
Dann hat jedes Keyword seine eigene Ergebnisliste, die gemäß booleschem Retrieval mit Mengenoperationen verrechnet werden: Ist AND ausgewählt, wird aus den

Listen der Durchschnitt gebildet, bei *OR* die Vereinigung.
Nun liegt das finale Ergebnis für die Keywordsuche der Teilanfrage vor - es sei denn, der Nutzer hat *NOT* ausgewählt. Dann wird die Differenz zwischen den Dokumenten des Archivs und dem ermittelten Ergebnis zurückgegeben.

7.3.2 Freitextsuche

7.3.3 Erstellen des Query-Vektors

Für die Freitextsuche muss die Anfrage erst in einen Vektor umgewandelt werden. Auch hier wird eine Hash-Table für das Anlegen des Query-Vektors verwendet.
Für jeden Term wird dessen Index im Term-Dictionary abgefragt. Vorausgesetzt, der Term existiert im Vokabular, wird dieser Index zum Key und die Termhäufigkeit in der Anfrage zum Wert. Anschließend wird die Termhäufigkeit durch den idf-Wert dividiert, sodass der Query-Vektor die finalen tf-idf-Gewichtungen enthält.
Damit wird die Anfrage genau wie ein Dokument behandelt, mit dem einzigen Unterschied dass bestimmte Terme eventuell nicht im Vektor eingetragen und gewichtet werden, da diese im Archiv nicht vorkommen und darum für die Suche irrelevant sind.
Abbildung 7.4 veranschaulicht das Erstellen eines Query-Vektors anhand eines Beispiels.

Finden der Resultate

Das Bestimmen der Suchergebnisse für die Freitextsuche läuft wie folgt ab:

- Es wird über alle Dokumente im Dokument-Dictionary iteriert und auf deren Dokument-Vektoren zugegriffen
- Es wird das Cosinus-Maß (siehe Formel 5.9) zur Berechnung der Ähnlichkeit zwischen Dokument- und Query-Vektor verwendet
- Das Ergebnis ist der Score, welcher gemeinsam mit der *docID* der Ergebnisliste hinzugefügt wird.
- Diese wird basierend auf dem Score sortiert, sodass sich die ähnlichsten Dokumente vorne befinden.

7.4 Verrechnung der Suchergebnisse

Die Resultate der Keywordsuche und der Freitextsuche müssen miteinander kombiniert werden, wobei der Score problematisch ist, da die Freitextergebnisse einen besitzen, die Keyword-Resultate jedoch nicht.
Das Kombinieren von beiden Suchen wurde wie folgt realisiert:

- Jedes Dokument in der Keyword-Ergebnisliste erhält den Score 1.
- Ist *AND* ausgewählt, wird der Durchschnitt beider Suchen gebildet und Keyword-Score sowie Freitextscore werden addiert.

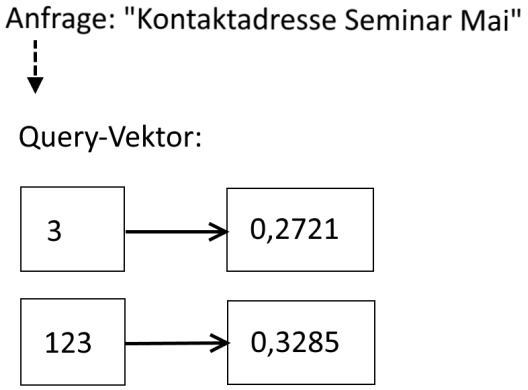

Abb. 7.4. Beispiel für Umwandlung einer Anfrage in einen Query-Vektor. „Kontaktadresse“ hat den Index 3, „Seminar“ den Index 123. Mai kommt im Archiv nicht vor, darum wird der Term nicht vermerkt. Alle Indizes ungleich 3 und 123 sind als mit 0 gewichtet zu interpretieren (Eigene Abbildung).

- Ist *OR* ausgewählt, wird die Vereinigung beider Suchen gebildet und Keyword-Score sowie Freitextscore der Dokumente, die in beiden Ergebnislisten vorkommen, werden addiert.

Da die Anfrage beliebig tief geschachtelt werden kann, ist es möglich, dass sich der Score eines Dokuments weiter erhöht:
Das Gesamtergebnis wird mit jeder neuen Teilanfrage auf dieselbe Weise verrechnet, wie es soeben beschrieben wurde.
Demnach erhält ein Dokument, dass für 5 Teilanfragen einen Treffer in der Keywordsuche lieferte, den Score 5. Handelt es sich hingegen um einen nicht ganzzahligen Wert, z.B. 5,27, kamen noch Treffer in der Freitextsuche hinzu.
Aufgrund des Rankings kann der Nutzer in etwa abschätzen, wie wichtig ein Dokument für seine Anfrage war und auch, auf welche Weise der Treffer zustande kam.

8

Die Benutzeroberfläche

Dieses Kapitel soll die Konzeption der Oberfläche begründen und deren Nutzung erläutern.

8.1 Anforderungen

Die Benutzeroberfläche muss die folgenden, aus der Problemstellung resultierenden Funktionalitäten erfüllen:

1. Wählen, Festlegen und Ändern des Suchverzeichnisses
2. Auswählen der Keywords
3. Auswählen der Freitextsuche
4. Eingabe des Suchtextes
5. Logische Operatoren zur externen Verknüpfung zur Verfügung stellen
6. Logische Operatoren zur internen Verknüpfung zur Verfügung stellen
7. Anzeigen der Anfrage
8. Starten der Suche
9. Zurücksetzen der Anfrage
10. Anzeige der Resultate

Neben den oben genannten inhaltlichen Anforderungen sind noch weitere Punkte bezüglich Anwenderfreundlichkeit zu berücksichtigen:

1. Übersichtlichkeit
2. Intuitive Bedienbarkeit, d.h. der Nutzer soll möglichst wenig nachdenken müssen

Die Realisierung der soeben beschriebenen Anforderungen wird in den folgenden Abschnitten erläutert.

8.2 Grundaufbau der Oberfläche

Die Oberfläche wurde, um dem Nutzer eine gewisse Übersichtlichkeit zu bieten, in drei Bereiche gegliedert, die in Abbildung 8.1 gezeigt sind.

Der oberste Bereich beinhaltet die Verzeichnisauswahl, da dies der erste Schritt ist, der vom Anwender ausgeführt werden muss.
In der Mitte befindet sich die Anzeige der Gesamtanfrage, weil sie sich dort sofort im Blickfeld des Nutzers befindet.
Die Editierung der Teilanfragen wurde im unteren Teil der Oberfläche untergebracht, weil sich hier die meisten Bedienungselemente befinden, weshalb jede andere Position unweigerlich Einbußen bezüglich Übersichtlichkeit zur Folge hätte.
Funktional zusammengehörige Elemente wurden hierbei stets nah beieinander angeordnet, was dem Gesetz der Nähe entspricht.
Dieses Gesetz besagt, dass Elemente, die nah zusammen liegen, vom Anwender als zusammengehörig wahrgenommen werden ([Hof17], S.17).

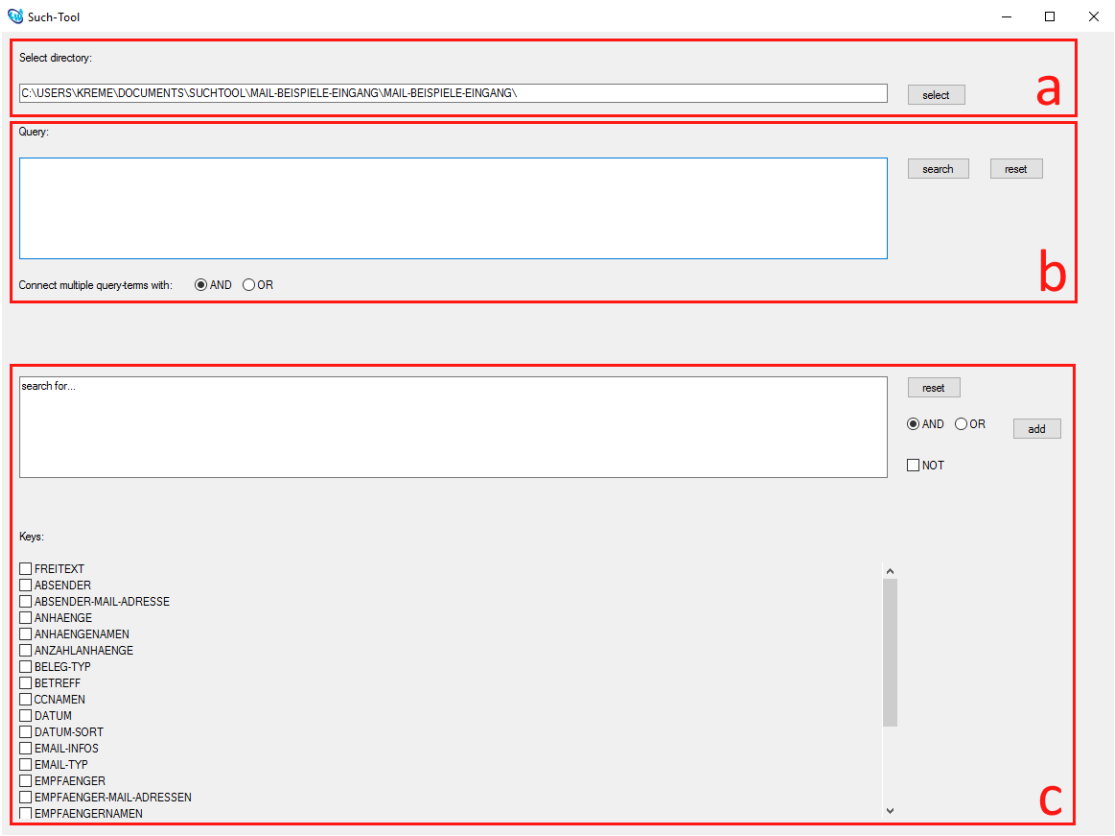

Abb. 8.1. Gliederung der Benutzeroberfläche. Teil *a* beinhaltet die Verzeichnisauswahl, Teil *b* die gesamte Suchanfrage und Teil*c* die Eingabe der Teilanfrage (Eigene Abbildung).

8.3 Verzeichnisauswahl

Die Verzeichnisauswahl (siehe Abbildung 8.1 Teil a) besteht aus einem Textfeld und einem rechts daneben befindlichen Button mit der Aufschrift „select“, welcher

das Auswahlmenü öffnet.
Dieses wird in einem seperaten Pop-Up-Fenster angezeigt (siehe Abbildung 8.2) und besitzt zwei Möglichkeiten, ein Verzeichnis zu selektieren:

1. Eintippen des Pfads in ein Textfeld
2. Auswahl des Verzeichnisses über ein Dropdown-Menü

Der Nutzer muss die Wahl mit „ok“ bestätigen bzw. kann den Dialog über „cancel“ abbrechen.
Anschließend wird der selektierte Verzeichnispfad im Textfeld angezeigt.

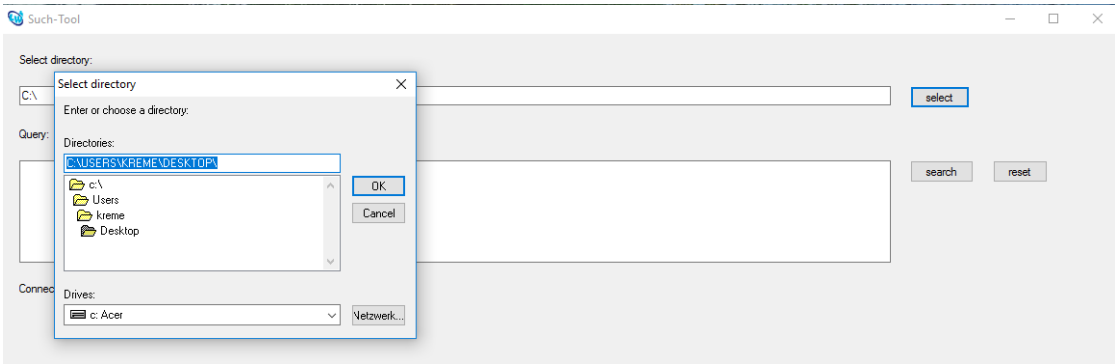

Abb. 8.2. Verzeichnisauswahl (Eigene Abbildung)

8.4 Suchanfrage

Im unteren Teil der Benutzeroberfläche (siehe Abbildung 8.1 Teil c) befinden sich die Keywords.
Diese wurden als auswählbare Checkboxen realisiert, welche sich auf einem vertikal scrollbaren Panel befinden. So ist sichergestellt, das eine beliebig große Anzahl Keywords angezeigt werden kann.
Die Freitextsuche ist über die erste Checkbox „Freitext“ auswählbar, womit die Freitextsuche wie ein zusätzliches Keyword behandelt wird, das sich in die Benutzeroberfläche einfügt. Dies ist gewollt, denn der Anwender soll nicht mitbekommen, dass das Suchen im Freitext intern anders implementiert ist als die übrigen Auswahlmöglichkeiten.

8.4.1 Zusammenstellung der Teilanfrage

Im Textdisplay wird die Anfrage vom Nutzer eingegeben und lässt sich über „reset“ bequem zurücksetzen.
Die ausgewählten Checkboxen bestimmen darüber, in welchen Bereichen nach der Anfrage gesucht werden soll, wobei beliebig viele auf einmal selektierbar sind.

Im Falle mehrerer ausgewählter Bereiche lassen sich die Teilergebnisse mit *AND* sowie *OR* verknüpfen. Beide Möglichkeiten lassen sich über zwei Radio-Buttons wählen, wobei *AND* die Standardauswahl ist.
Zusätzlich lässt sich die Anfrage durch das Auswählen der Checkbox *NOT* negieren. Der Operator bezieht sich hierbei auf das Gesamtergebnis, wenn die Verknüpfung bereits erfolgt ist. Möchte man Bereiche einzeln negieren, müssen die Anfragen getrennt eingegeben werden.
Das Drücken von „Add“ fügt die Teilanfrage der Gesamtanfrage hinzu. Sie wird dann auf dem Display im mittleren Bereich angezeigt (siehe Abbildung 8.1 Teil b).

8.4.2 Beispiel

Um die einzelnen Schritte besser nachvollziehen zu können, sei ein Beispiel anhand von Abbildungen gezeigt.
Abbildung 8.3 zeigt, wie der Begriff „Dr. Claus-Peter Wirth“ im Absender oder in der Absender-Mail-Adresse gesucht werden soll. Das Drücken von Add bestätigt die Teilanfrage und eröffnet die Möglichkeit, eine weitere zu stellen.

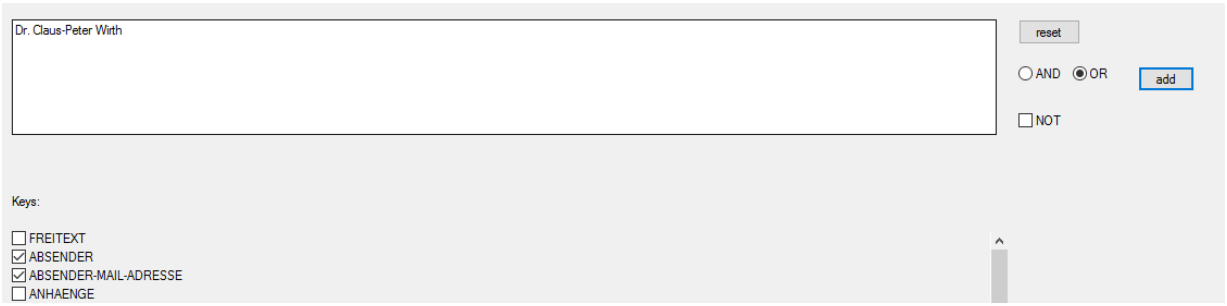

Abb. 8.3. Keywords auswählen und intern verknüpfen, hier OR (eigene Abbildung)

Abbildung 8.4 zeigt, wie der Begriff „dfki“[1] gesucht im Bereich Freitext gesucht werden soll. Es erfolgt erneutes Bestätigen durch „Add“.

8.5 Gesamtanfrage

Im vorigen Beispiel wurden zwei Teilanfragen gestellt, da zweimal Add betätigt wurde.
Das externe Verknüpfen mehrerer Teilanfragen mit *AND* bzw. *OR* wird durch zwei Radio-Buttons unter dem oberen Textdisplay (siehe Abbildung 8.1, Teil b) geregelt, wobei die Standardauswahl auch hier *AND* ist.
Ist stattdessen *OR* erwünscht, muss diese Auswahl vor dem Betätigen des Add-Buttons getroffen worden sein, da eine Änderung des Operators im Nachhinein

[1] DFKI = Deutsches Forschungszentrum für Künstliche Intelligenz

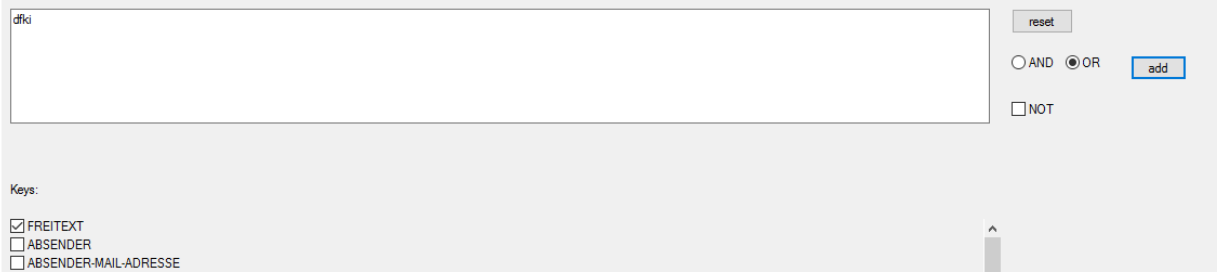

Abb. 8.4. Freitextauswahl. Da nur ein Bereich ausgewählt ist, bleibt der selektierte *OR*-Operator ohne Wirkung (eigene Abbildung)

nicht mehr möglich ist!
Abbildung 8.5 zeigt, wie die im Display angezeigte Anfrage für das Beispiel mit dem Operator *AND* aussieht.
In das obere Display lässt sich vom Nutzer nichts eingeben, um die korrekte Anzeige der intern gespeicherten Anfrage zu gewährleisten.

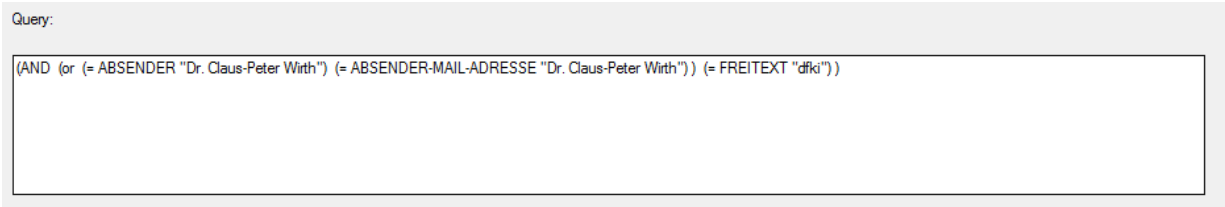

Abb. 8.5. Display (eigene Abbildung)

8.6 Ergebnis

Der Nutzer kann die Gesamtanfrage mittels reset-Button komplett zurücksetzen oder aber über den search-Button die Suche starten.
Das Drücken von „search“ öffnet ein Pop-Up-Fenster, welches die Resultate anzeigt und in Abbildung 8.6 gezeigt ist.
In einer tabellarischen Ansicht werden die Rankposition, der Score, wenn möglich Datum und Absender sowie der Dateipfad angezeigt.
Die Tabelle ist vertikal scrollbar (siehe Abbildung 8.7), falls viele Ergebnisse vorliegen. Eine Begrenzung wurde nicht vorgenommen, da keine Resultate ausgeschlossen werden sollen. Der Nutzer kann seine Wahl basierend auf der Ranking Position somit selbst treffen.
Die Einträge der Tabelle lassen sich per Doppelklick auswählen, sodass das zugehörige Verzeichnis mit der als ausgewählt markierten Datei bzw. die Datei selbst geöffnet wird. Welche dieser beiden Varianten gewünscht ist, kann der Nutzer über die oben links angebrachten Radio-Buttons „Open directory“ sowie „Open file “

bestimmen.
Das Drücken von „ok“ schließt das Ergebnis-Fenster.

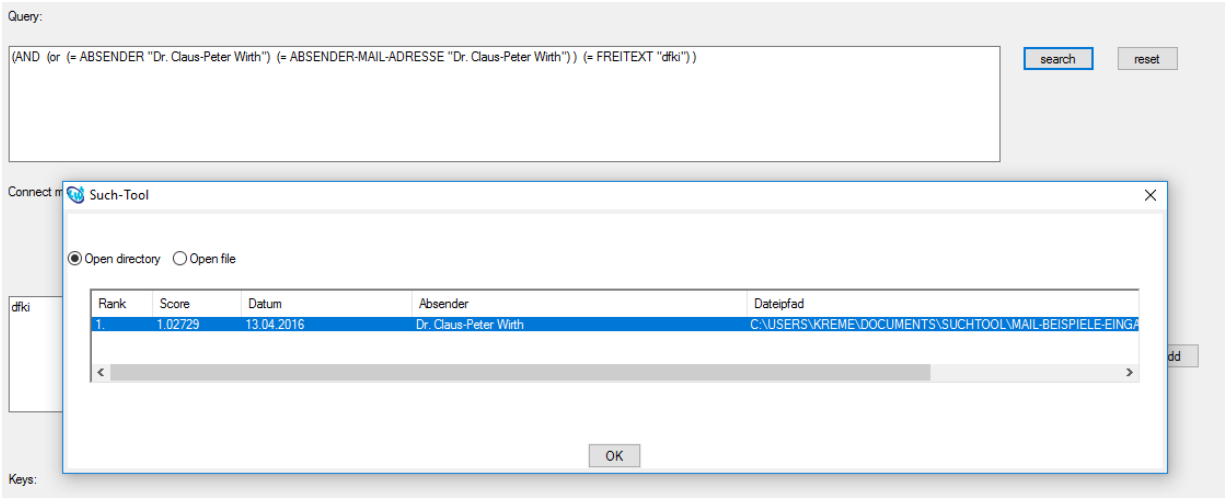

Abb. 8.6. Resultatfenster (eigene Abbildung)

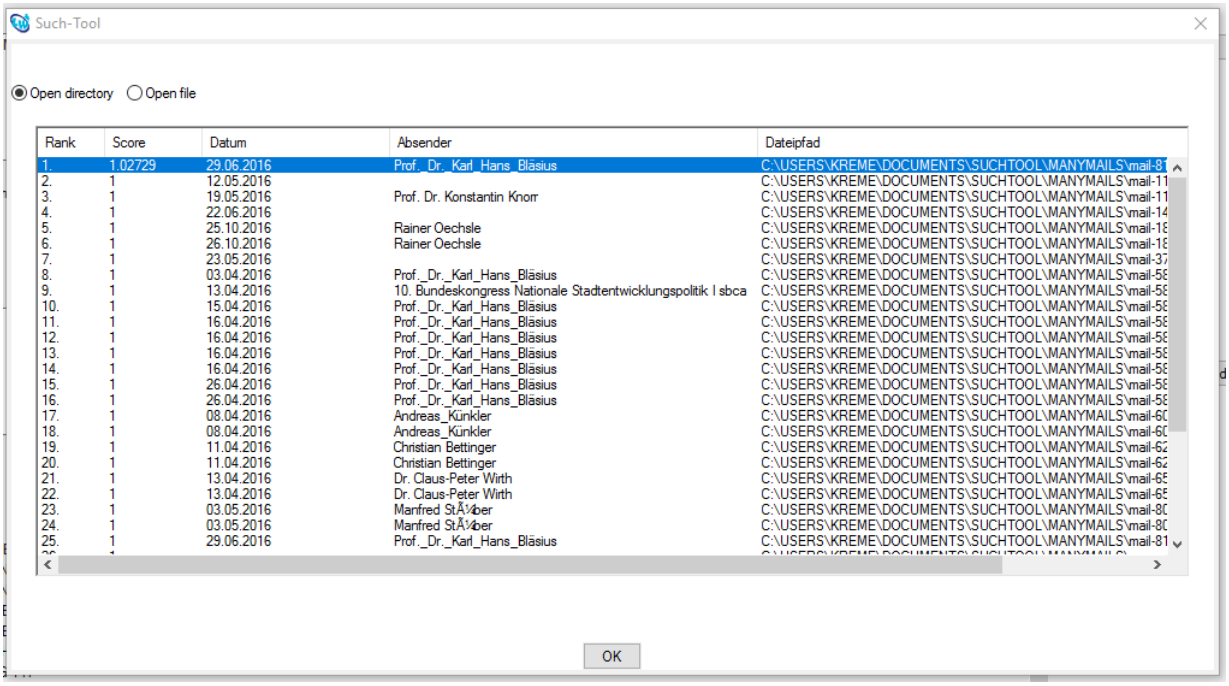

Abb. 8.7. Scrollbare Anzeige für viele Treffer (eigene Abbildung)

9

Zusammenfassung und Ausblick

9.1 Zusammenfassung

Im Kapitel Einleitung und Problemstellung wurde die Aufgabenstellung erläutert, wobei sich bereits zeigte dass diese aufgrund des semistrukturierten Aufbaus der zu durchsuchenden Dokumente eine sehr spezifische Lösung erfordern würde. Die Unterteilung in Metadaten und Freitext sowie das logische Verknüpfen der Suchbedingungen stellen die wesentlichen Herausforderungen dar. Anschließend wurden die notwendigen theoretischen Kenntnisse vermittelt. Die Bedeutung des Begriffs Information Retrieval wurde erklärt und es zeigte sich, dass diese sehr weit gefasst ist, weshalb diese Arbeit die ausgesprochen umfangreiche Thematik nur ansatzweise behandeln kann. Es wurden die beiden bekanntesten klassichen Information Retrieval Verfahren vorgestellt; das boolesche Retrieval und das Vektorraummodell, sowie Methoden, anhand derer sich solche Modelle im Hinblick auf Qualität bewerten und Vergleichen lassen. Hierbei zeigte sich, dass eine solche Bewertung im Rahmen dieser Arbeit aufgrund des Aufwands leider nicht durchführbar ist.
Im Implementierungskapitel wurde deutlich, dass die Verwendung eines der beiden klassischen Verfahren nicht ausreicht, die Kombination aus booleschen Retrieval und Vektorraummodell jedoch hervorragend passt. Das boolesche Retrieval musste allerdings geringfügig modifiziert werden, indem nicht nur das Vorkommen der Attribute in den Dokumenten vermerkt wird, sondern auch deren Inhalt inklusive Datentyp. Beim Vektorraummodell zeigte sich, dass sich Speicher sparen lässt, indem alle Elemente des Vektors, welche eine 0 enthalten, nicht eingetragen werden. Zuletzt zeigte sich, dass die Benutzeroberfläche eine nicht zu unterschätzende Herausforderung darstellt, da der Anwender sein Informationsbedürfnis auf möglichst unkomplizierte Art ausdrücken können soll. Ein Information Retrieval System muss vom Nutzer auch verstanden werden. Es wurde sich dafür entschieden, möglichst einfache UI-Elemente wie CheckBoxes zu wählen sowie die Oberfläche in drei thematische Bereiche zu gliedern, um die notwendige Übersicht zu bieten.

9.2 Ausblick: Wünschenswerte Erweiterungen

Dieser Abschnitt liefert einen Ausblick, welche Erweiterungen für die Implementierung wünschenswert wären, im Rahmen dieser Arbeit jedoch aus zeitlichen

Gründen nicht behandelt werden konnten.
Eine sehr vorteilhafte Erweiterung stellt die Lemmatisierung dar, d.h. das Zurückführen der Wörter auf deren Wortstämme, sodass die Anzahl der zu speichernden Terme deutlich reduziert wird. Gerade bei größeren Archiven ist dies unabdingbar, da der Speicherplatz limitiert ist. Außerdem erhöht dies die Anzahl möglicher Treffer, da auch zum Suchbegriff verwandte Wörter gefunden werden.
Zudem möglich wäre das Ergänzen von Spelling Correction: Vertippt der Nutzer sich, so soll nicht gleich nichts zurückgeliefert werden, sondern die Meldung, ob nicht eventuell ein anderes Wort gemeint war, inklusive einer Liste von auswählbaren Vorschlägen. Dies würde das System fehlertoleranter und nutzerfreundlicher machen.

Literaturverzeichnis

Aca. *Academic - Academic dictionaries and encyclopedias, Universal Lexikon.* http://universal_lexikon.deacademic.com/253489/Informationsr%C3%BCckgewinnung, letzter Zugriff am 05.06.2017.

CDM08. Christopher D. Manning, Prabhakar Raghavan, Hinrich Schütze: *Introduction to Information Retrieval.* Cambridge University Press, 2008.

Eng14. Engelhardt, Alexander: *Kreuztabellen / Kontingenztafeln*, 2014. http://www.crashkurs-statistik.de/kreuztabellen-kontingenztafeln/, letzter Zugriff am 03.07.2017.

Fer03. Ferber, Reginald: *Information Retrieval, Suchmodelle und Data-Mining-Verfahren für Textsammlungen und das Web.* dpunkt.verlag, 2003.

Hen08. Henrich, Andreas: *Information Retrieval 1 -Grundlagen, Modelle und Anwendungen*, 2008. https://www.uni-bamberg.de/fileadmin/uni/fakultaeten/wiai_lehrstuehle/medieninformatik/Dateien/Publikationen/2008/henrich-ir1-1.2.pdf, letzter Zugriff am 10.06.2017 .

Hof17. Hofer, Andreas: *User-Interface-Design WS 2016/17 - 2 Gestalten.* Vorlesungsskript, FH-Trier, 2017.

Ion16. Ionescu, Stelian: *split-sequence Package*, 2016. https://github.com/sharplispers/split-sequence/blob/master/split-sequence.lisp, letzter Zugriff am 23.06.2017.

Jän84. Jänich, Klaus: *Lineare Algebra - Ein Skriptum für das erste Semester.* Springer-Verlag, 1984.

KMOB14. Kweku-Muata Osei-Bryson, Ojelanki Ngwenyama: *Advances in Research Methods for Information Systems Research - Data Mining, Data Envelopment Analysis, Value Focused Thinking.* Nummer 34 in *Integrated Series in Information Systems.* Springer Verlag, 2014.

PDVC06. Prof. Dr. Volker Claus, Prof. Dr. Andreas Schwill: *Duden - Informatik A-Z, Fachlexikon für Studium, Ausbildung und Beruf.* Dudenverlag, 2006.

SB10. Stefan Büttcher, Charles L.A.Clarke, Gordon V.Cormack: *Information Retrieval - Implementing and Evaluating Search Engines.* The MIT Press - Massachusetts Institute of Technology, 2010.

A

Erklärung der Kandidatin / des Kandidaten

☐ Die Arbeit habe ich selbstständig verfasst und keine anderen als die angegebenen Quellen- und Hilfsmittel verwendet.

☐ Die Arbeit wurde als Gruppenarbeit angefertigt. Meine eigene Leistung ist ...

Diesen Teil habe ich selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet.

Namen der Mitverfasser: ...

Datum

Unterschrift der Kandidatin / des Kandidaten