

Konzeption und Realisierung eines Systems zur Informations-  
suche in einem Dokumentenarchiv basierend auf Textinhalt  
und Metadaten

Conception and Realization of an Information Retrieval 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 Konzeption und Realisierung eines Information-Retrieval-Systems, 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 zu seinem Informationsbedürfnis passende Dokumente zurückzuliefern 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 einen Überblick über die Thematik zu geben. Es folgt eine Vorstellung der beiden klassischen Information Retrieval Modelle boolesches Retrieval und Vektorraummodell inklusive Erläuterung der Funktionsweisen. 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 zum Einsatz kamen und inwieweit eine Modifizierung zur Anpassung auf die vorliegende Problemstellung erfolgte. Neben der internen Funktionsweise wird 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 zukünftige Verbesserungsmöglichkeiten.

This paper discusses the conception and realization of an information retrieval system that searches efficiently in semistructured data containing free text and metadata. 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 retrieve documents satisfying the user's information need and to present the results clearly.

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 an overview on the topic. The chapter about information retrieval is followed by a presentation of two classical information retrieval models 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 and if they have been modified in any way to fit this problem. Aside from the internal 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 system improvements.

---

# Inhaltsverzeichnis

<b>1</b>	<b>Einleitung und Problemstellung</b>	<b>1</b>
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	3
1.3.5	Verknüpfung mit UND/ODER	3
1.3.6	Benutzeroberfläche	3
<b>2</b>	<b>Information Retrieval</b>	<b>4</b>
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
<b>3</b>	<b>Grundbegriffe</b>	<b>7</b>
3.1	Anfrage	7
3.2	Indexierung	7
3.2.1	Term und Vokabular	7
3.2.2	Stoppwörter	8
<b>4</b>	<b>Boolesches Retrieval</b>	<b>9</b>
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-Dokument Inzidenz Matrix	11

---

4.3.2	Invertierte Liste .....	12
<b>5</b>	<b>Das Vektorraummodell .....</b>	<b>16</b>
5.1	Funktionsprinzip .....	16
5.2	Vektor und Vektorraum .....	16
5.3	Definition Vektorraummodell .....	18
5.4	Gewichte .....	18
5.4.1	Termhäufigkeit .....	19
5.4.2	Dokumenthäufigkeit .....	19
5.4.3	Invertierte Dokumenthäufigkeit .....	19
5.4.4	TF-IDF-Gewichtung .....	20
5.5	Anfragen .....	20
5.6	Ähnlichkeitsfunktion .....	21
5.6.1	Euklidischer Abstand .....	21
5.6.2	Cosinus-Maß .....	21
<b>6</b>	<b>Bewertung eines Information-Retrieval-Systems .....</b>	<b>24</b>
6.1	Problem Relevanz .....	24
6.2	Precision und Recall .....	25
6.2.1	Precision .....	25
6.2.2	Recall .....	25
6.2.3	Veranschaulichung .....	25
6.3	Zwei Evaluierungsmaße .....	26
6.4	Durchführung .....	26
<b>7</b>	<b>Implementierung .....</b>	<b>28</b>
7.1	Teilweise strukturierte Dokumente .....	28
7.2	Initialisierungsschritte .....	29
7.2.1	Erstellen des Dokument-Dictionaries .....	29
7.2.2	Verarbeiten der Dokumente .....	30
7.3	Suche .....	33
7.3.1	Metadatensuche .....	33
7.3.2	Freitextsuche .....	34
7.3.3	Erstellen des Query-Vektors .....	34
7.4	Verrechnung der Suchergebnisse .....	35
<b>8</b>	<b>Die Benutzeroberfläche .....</b>	<b>36</b>
8.1	Anforderungen .....	36
8.2	Grundaufbau der Oberfläche .....	36
8.3	Verzeichnisauswahl .....	37
8.4	Suchanfrage .....	38
8.4.1	Zusammenstellung der Teilanfrage .....	38
8.4.2	Beispiel .....	39
8.5	Gesamtanfrage .....	39
8.6	Ergebnis .....	40

---

<b>9 Zusammenfassung und Ausblick</b> .....	42
9.1 Zusammenfassung .....	42
9.2 Ausblick: Wünschenswerte Erweiterungen .....	43
<b>Literaturverzeichnis</b> .....	44
<b>Erklärung der Kandidatin / des Kandidaten</b> .....	46

---

## Abbildungsverzeichnis

4.1	Term-Dokument Inzidenz Matrix. Die Zeilen enthalten die Terme, die Spalten die <i>docIDs</i> der Dokumente. Alle Einträge mit einer 0 sind leere Einträge (Eigene Abbildung).....	11
4.2	Invertierte Listen zu drei Beispieltermen. Jede invertierte Liste enthält die Dokumentindizes oder auch <i>docIDs</i> der Dokumente, in denen der jeweilige Term vorkommt (Eigene Abbildung).....	13
5.1	Vektorraum mit den Termen $T1$ und $T2$ als Achsen, drei Dokumentvektoren zu den Dokumenten $d_i$ und einem Anfragevektor zur Anfrage $q$ . Das Cosinus-Maß liefert als ähnlichstes Dokument $d_2$ (Eigene Abbildung, basierend auf [SB10], S.55). ....	22
7.1	Eintrag im Dokument-Dictionary. Die Struct-Slots Datum und Absender können, falls nicht existent, auch leer bleiben. Im rot markierten Bereich wird später der Dokumentvektor eingetragen (Eigene Abbildung). ....	30
7.2	Modifizierte invertierte Liste zur Realisierung der Metadatenuche. Statt der <i>docIDs</i> , wie es bei invertierten Listen üblich ist, werden Structs der Form (docID, Typ, Inhalt) gespeichert(Eigene Abbildung).....	31
7.3	Struktur für einen Eintrag im Term-Dictionary. Der Termname bildet den Hash-Key, Index und idf sind in einem Struct zusammengefasst. Der Index gibt die Position des Terms im Dokumentvektor an (Eigene Abbildung).....	32
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 ignoriert. Alle Indizes ungleich 3 und 123 sind als mit 0 gewichtet zu interpretieren (Eigene Abbildung). ....	35
8.1	Gliederung der Benutzeroberfläche. Teil <i>a</i> beinhaltet die Verzeichnisauswahl, Teil <i>b</i> die gesamte Suchanfrage und Teil <i>c</i> die Erstellung der Teilanfrage (Eigene Abbildung). ....	37

---

8.2	Verzeichnisauswahl (Eigene Abbildung) . . . . .	38
8.3	Attributnamen auswählen und intern verknüpfen, hier mit dem <i>OR</i> -Operator (eigene Abbildung). . . . .	39
8.4	Freitextauswahl. Da nur ein Suchbereich ausgewählt ist, bleibt der selektierte <i>OR</i> -Operator ohne Wirkung (eigene Abbildung). . . . .	40
8.5	Anzeige der Gesamtanfrage auf dem oberen Display, das sich in Bereich b befindet (eigene Abbildung). . . . .	40
8.6	Pop-Up Fenster zur Anzeige der Resultate (eigene Abbildung). . . . .	41
8.7	Scrollbare Anzeige, um beliebig viele Treffer anzeigen zu können (eigene Abbildung). . . . .	41



---

## Tabellenverzeichnis

6.1	Kontingenztafel ([CDM08], S.143) .....	25
-----	--	----

# Einleitung und Problemstellung

## 1.1 Einleitung

Nahezu jeder nutzt heutzutage täglich E-Mail-Dienste und kennt das Phänomen, dass der Posteingang sich unter der Flut eintreffender Nachrichten stetig füllt, bis der Ordner so voll ist, dass jede Übersicht verloren geht.

Sobald eine bestimmte E-Mail darin wiedergefunden werden soll, beispielsweise weil sie eine bestimmte Kontaktadresse enthält, wird dies zu einem Problem: Wie war nochmal der Absender? Längst vergessen. Das genaue Datum? Leider ist nur noch der Monat bekannt. Eine manuelle Suche ist hier oft aus Zeitgründen unmöglich.

Ein Information-Retrieval-System, mit dem bestimmte Suchkriterien eingegeben und beliebig miteinander kombiniert werden können, löst dieses Problem. Eine mögliche Suchanfrage könnte *Freitext = Kontaktadresse AND Datum = Juni* lauten, woraufhin das System bei erfolgreicher Suche eine Reihe passender Resultate aufsteigend geordnet nach Übereinstimmung mit der Suchanfrage liefert, sodass der Anwender sich nicht selbst durch hunderte von Mails durcharbeiten muss.

Besonderheit ist das beliebige logische Verknüpfen: Der Anwender kann entscheiden, ob er nur Resultate akzeptiert, auf die beides zutrifft, wie es im obigen Beispiel der Fall ist, oder ob es bereits reicht, wenn eines der Kriterien erfüllt ist. Zudem können die Suchkriterien auch negiert werden, um bestimmte Dokumente auszuschließen. Da die logischen Ausdrücke beliebig tief geschachtelt werden können, erlaubt dies 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. wünschenswert: Es 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 in einem Dokumentenarchiv, welches semistrukturierte Dokumente enthält.

Im Englischen wird ein solches System *information retrieval system* genannt, der deutsche Begriff hierfür lautet Information-Retrieval-System oder kurz IR-System. Semistrukturiert oder teilweise strukturiert bedeutet, dass die Dokumente sowohl gewöhnlichen Freitext als auch Metadaten enthalten. Metadaten oder auch Metainformationen sind Daten, die andere Daten beschreiben, d.h. sie enthalten Informationen über das eigentliche Dokument ([met16]). Im Falle von E-Mails sind Datum, Absender, Betreff etc. typische Metadaten.

Der Nutzer soll spezifizieren können, in welchen Metadaten er suchen möchte, zudem soll die Freitextsuche auswählbar sein. Die gewählten Suchkriterien sollen beliebig mit den logischen Operatoren *AND* (engl. und), *OR* (engl. oder), *NOT* (engl. nicht) verknüpfbar sein und die sich ergebenden logischen Ausdrücke sollen beliebig tief geschachtelt werden können.

Hauptanwendungszweck des Systems sind E-Mail-Archive wie Posteingang und Postausgang, allerdings soll das System so flexibel sein, dass es auch auf andere Archive mit teilweise strukturierten Dokumenten 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 Information-Retrieval-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

Für das dynamische Einlesen der Metadaten 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.
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 Bereichen muss, unter Berücksichtigung des jeweiligen Datentyps der Inhalte, gesucht werden.

Im Gegensatz zur Freitextsuche muss hier zu jedem Dokument vor der Suche zunächst geprüft werden, ob das entsprechende Schlüsselwort, z.B. der Name „Datum“, 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, welches bei komplexeren Anfragen auch teilweise passende Ergebnisse liefert und die Resultate hinsichtlich des Übereinstimmungsgrads mit der Nutzeranfrage bewerten und entsprechend ordnen kann.

### 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. Sind in der Metadatensuche mehrere Metadaten als Suchfelder ausgewählt, müssen die Teilergebnisse zu jedem Suchfeld zu einem Gesamtergebnis für die Metadatensuche zusammengefasst werden, wobei die Art des Zusammenfassens vom selektierten logischen Operator abhängt.
2. Resultate der Metadatensuche und der Freitextsuche müssen miteinander zu einem Gesamtergebnis für die aktuelle Anfrage verknüpft werden. Auch hier bedingt der logische Operator die Art der Verknüpfung.
3. Stellt der Nutzer mehrere Teilanfragen, müssen die Ergebnisse der einzelnen Anfragen logisch verknüpft werden. Die Art der Verknüpfung mehrerer Teilanfragen muss ebenfalls einstellbar sein.

### 1.3.6 Benutzeroberfläche

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

1. Auswählbares Suchverzeichnis, d.h. das Dokumentenarchiv in welchem die Suche erfolgen soll, muss selektiert werden können
2. Übersichtliche Anzeige und Auswahlmöglichkeit aller möglichen Suchfelder, d.h. die Namen aller im Archiv vorkommenden Metadaten sowie der Freitext
3. Eingabemöglichkeit für den Suchtext
4. Selektierbare logische Operatoren (*AND*, *OR*, *NOT*) zur internen Verknüpfung einer Anfrage sowie zur externen Verknüpfung mehrerer Teilanfragen
5. Benutzerfreundlichkeit, d.h. die Oberfläche muss so verständlich und intuitiv bedienbar sein wie möglich

## 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 ([Aca12]). Hierbei wird explizit von *Rückgewinnung* gesprochen, da keine neuen Informationen erzeugt werden, sondern auf bereits existierende zugegriffen wird. Bevor auf die genaue Bedeutung eingegangen wird, erfolgt zunächst die hierfür notwendige Klärung des im Ausdruck enthaltenen Teilbegriffs „Information“.

#### 2.1.1 Information

Die Bedeutung des Begriffs Information ist sehr weit gefasst, was eine einheitliche Definition unmöglich macht. Er stammt von dem Lateinischen Wort *informare*, was sich mit Gestalt geben übersetzen lässt und im Übertragenen Sinne so viel wie jemanden durch Unterweisung bilden heißt.

Dies betont den Aspekt, dass eine Information stets einen Empfänger besitzt, welcher „gebildet“ wird. Dieser kann eine Person, aber auch ein geeignetes, nach außen wirksames System sein. Erst das Aufnehmen und korrekte Interpretieren durch einen Empfänger macht aus Daten als Informationsträgern tatsächlich Informationen. Die Informationen müssen deshalb auf irgendeine Weise dargestellt werden, beispielsweise durch alphabetische Zeichen. Zudem muss es hierfür einen geeigneten Träger, z.B. ein Textdokument, geben.

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

Nach dem der Teilbegriff Information vorgestellt wurde, wird in diesem Abschnitt auf die Bedeutung von Information Retrieval eingegangen. 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

Zum Besseren Verständnis hilft eine Abgrenzung zur Datenbanksuche, bei der im Gegensatz zu Definition 2.1 die Daten strukturiert in Form von Werttupeln bekannten Datentyps vorliegen. Ein wesentliches Unterscheidungsmerkmal ist, dass bei der Datenbanksuche nicht mit vagen Anfragen umgegangen werden kann: 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 kann hingegen solche Anfragen mit unklarer Bedeutung verarbeiten.

## 2.2 Beispiel Websuche

An dieser Stelle soll ein Beispiel zur Veranschaulichung gegeben werden. Nahezu jeder benutzt im Alltag Web-Suchmaschinen. Websuche stellt einen typischen Fall von Information Retrieval dar, was durch die Anwendung der Definition deutlich wird: Hier sollen Freitext beinhaltende, d.h. unstrukturierte Dokumente (z.B. im HTML- oder pdf-Format) innerhalb des World Wide Webs aufgefunden werden, um das Informationsbedürfnis des Internetnutzers zu befriedigen ([Fer03], S.6). Relevante Suchergebnisse sind demnach Dokumente, welche die gesuchte Information beinhalten. Diese lässt sich, wie oben beschrieben, in drei Teile zerlegen, wobei der semantische Teil die Herausforderung für das Information-Retrieval-System darstellt.

Hierzu ein spezifisches Beispiel: Möchte der Nutzer demnächst seinen Urlaub in Kreta verbringen, könnte seine Suchanfrage „*Hotel günstig Kreta*“ lauten. Der pragmatische Teil besteht demnach darin, den Urlaub zu planen. Der syntaktische Teil ist ebenfalls leicht zu bestimmen: Die gesuchten Begriffe oder hierzu verwandte Wörter müssen in den Dokumenten auftauchen. Als schwierig gestaltet sich

hingegen der semantische Teil: Die inhaltliche Bedeutung der Resultate muss mit der ursprünglichen Intention des Nutzers übereinstimmen. Diese ist allerdings vage formuliert: Der Begriff „günstig“ ist nicht genau definiert. Nur ein Teil der Hotels, welche in der Ergebnisliste erscheinen, werden mit den Ansprüchen des Nutzers übereinstimmen, vielleicht auch gar keine. Ein gutes Information-Retrieval-System zeichnet sich durch einen möglichst hohen Anteil relevanter Resultate unter allen zurückgelieferten Dokumenten aus.

Häufig passiert es, dass zwar der syntaktische Teil erfüllt ist, d.h. die Suchbegriffe tauchen im Dokument auf, allerdings stimmt der Kontext nicht mit dem Informationsbedürfnis des Nutzers überein. Dieses Problem tritt bei der Datenbanksuche, wo es keinerlei Interpretationsfreiraum gibt, gar nicht erst auf.

## 2.3 Bezug zur Problemstellung

Dieser Abschnitt soll erklären, inwiefern es sich bei der gegebenen Problemstellung um ein Information-Retrieval-Problem handelt. Die Aufgabe besteht kurz gefasst darin, nach vom Nutzer auswählbaren, logisch verknüpften Kriterien innerhalb eines Dokumentenarchivs zu suchen (siehe Abschnitt 1.2). Damit 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 strukturierte 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 dennoch ein Information-Retrieval-Problem vor, da der Begriff auch die Suche in teilweise strukturierten oder semistrukturierten (engl. semistructured) Dokumenten einschließt ([CDM08], S.1-2). Hierbei sei angemerkt, dass selbst die Metadaten nicht vollkommen strukturiert sind: Der Datentyp des Inhalts ist offen gelassen und es gibt keinerlei Vorgaben, welche Metadaten in den Dokumenten auftreten müssen.

In den folgenden Kapiteln wird beschrieben, auf welche Weise ein Information-Retrieval-System konzipiert und realisiert werden kann, welches die gegebene Problemstellung löst. Hierzu werden zunächst die hierfür benötigten Kenntnisse über die beiden klassischen Information-Retrieval-Verfahren boolesches Retrieval (siehe Kapitel 4) und Vektorraummodell (siehe Kapitel 5) vermittelt.

## 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.

### 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 syntaktisch korrekt 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-Systemen verarbeitet werden können, müssen diese mit einem eindeutigen Index, der *docID* (kurz für *document identification*), versehen werden, sodass schnell darauf zugegriffen werden kann. Bei der *docID* handelt es sich meist um einen ganzzahligen Wert ([CDM08], S.7). Zudem müssen die Dokumentinhalte indexiert werden: Hierzu werden die Texte in einzelne, mit einem eindeutigen Index versehene Einheiten zerlegt, sodass auch hierauf effiziente Zugriffe erfolgen können. 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 indexierten Einheiten, in welche die Dokumente zerlegt werden, sind unter dem Begriff Terme 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 sie sich neben dem Wortstamm auch auf ihre grammatikalische Grundform reduzieren. Die Reduktion der Wörter auf Wortstamm bzw. Grundform wird 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 Terme eines Archivs wird als Vokabular bezeichnet ([CDM08], S.6).

### 3.2.2 Stoppwörter

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 als Ergebnis infrage, sollte die Suchanfrage Stoppwörter beinhalten, was die Suche in solchen Fällen erheblich beschleunigt.

## 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 auf das Zutreffen einer bestimmten Bedingung. Somit erfolgt lediglich die Unterteilung in passende Dokumente und jene, welche diese Bedingung nicht erfüllen. Eine darüber hinausgehende Bewertung der Ergebnisse findet nicht statt, was zu einer ungeordneten Ergebnismenge führt, die in keine Rangfolge gebracht werden kann ([Fer03], S.33). Das fehlende Ranking ist ein häufiger Kritikpunkt des Verfahrens.

### 4.2 Funktionsprinzip

Boolesches Retrieval basiert auf Mengenoperationen, weshalb den Dokumenten Mengen zugeordnet werden, die jeweils durch bestimmte Attribute charakterisiert sind. Wenn von Dokumenten gesprochen wird, ist hiermit die Einheit gemeint, auf der das Retrieval stattfindet. Es kann sich hierbei um kleine Textmemos, aber auch um ganze Buchkapitel handeln ([CDM08], S.4).

#### 4.2.1 Attribut

Ein 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.

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

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  $D_{t,t_1}$  für eine Anfrage  $(t, t_1)$  kann demnach wie in Formel 4.2 charakterisiert werden.

$$D_{t,t_1} = \{d \in D \mid t(d) = t_1\} \quad (4.2)$$

#### Verknüpfung

Beim logischen Verknüpfen mehrerer elementarer boolescher Anfragen 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*.  $(t, t_1) \text{ AND } (s, s_1)$  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.

$$D_{t,t_1} \cap D_{s,s_1} \quad (4.3)$$

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.

$$D_{t,t_1} \cup D_{s,s_1} \quad (4.4)$$

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.

$$D \setminus D_{t,t_1} \quad (4.5)$$

Da bei jeder Mengenoperation als Ergebnis wieder 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 klassische Implementierungsansätze für boolesches Retrieval vorgestellt, mit denen sich die soeben beschriebenen Operationen realisieren lassen.

### 4.3.1 Term-Dokument Inzidenz Matrix

Eine mögliche Implementierung des booleschen Retrieval stellt die Umsetzung mittels einer Term-Dokument Inzidenz Matrix dar. In den Zeilen einer solchen Matrix werden die Terme eingetragen und in den Spalten die Dokumente bzw. deren *docIDs*. Tritt Term  $t$  in Dokument  $d$  auf, so lautet der Eintrag für  $(t, d)$  in der Matrix 1. Alle Einträge für nicht vorkommende Terme sind hingegen mit einer 0 versehen. Abbildung 4.1 zeigt ein Beispiel, wobei zu beachten ist dass die tatsächliche Anzahl an Termen und Dokumenten in einer Sammlung weitaus größer ausfällt. Eine Term-Dokument Inzidenz Matrix besitzt den gravierenden Nachteil, dass sie unnötig Speicherplatz verbraucht, da sehr viele Einträge der Matrix eine 0 enthalten. Gerade bei sehr großen Sammlungen bzw. Dokumenten ist dieser Implementierungsansatz aufgrund des limitierten Speicherplatzes 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 Zeilen enthalten die Terme, die Spalten die *docIDs* der Dokumente. Alle Einträge mit einer 0 sind leere Einträge (Eigene Abbildung).

### Verarbeitung einer Anfrage mittels Matrix

Um eine Anfrage wie *Kontaktadresse AND Seminar AND Termin* mithilfe einer Matrix zu verarbeiten, werden einfach die entsprechenden Zeilen entnommen und bitweise logisch verknüpft, was für die obige Anfrage und die Matrix aus Abbildung 4.1 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 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 (engl. inverted lists) verwendet ([Fer03], S.36). Der Name basiert auf den darin gespeicherten invertierten Indizes, die deshalb als invertiert bezeichnet werden, weil sie vom Term zurück auf die Position, 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, d.h. deren Indizes oder *docIDs*. Hierbei ist anzumerken, dass hierfür tatsächlich jede geeignete Speicherstruktur verwendet werden kann, denn trotz des Namens muss es sich nicht um Listen handeln. Abbildung 4.2 zeigt ein Beispiel einer solchen invertierten Liste. Manche Implementierungen beinhalten neben der *docID* noch zusätzliche Informationen wie die genaue Wortposition im Dokument.

Das Verfahren ermöglicht sehr schnelle Zugriffe, ist allerdings speicherintensiv ([Fer03], S.36). Invertierte Listen stellen jedoch eine deutliche Verbesserung im Vergleich zu Term-Dokument Inzidenz Matrizen dar, da sie keine leeren Einträge enthalten.

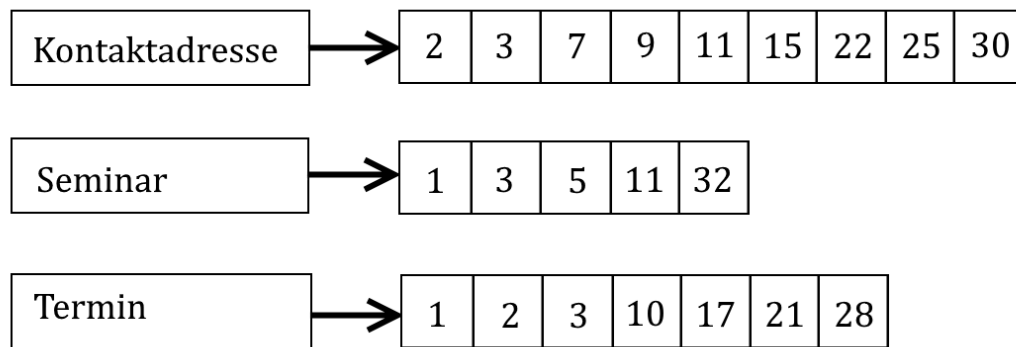
#### Verarbeitung einer Anfrage mittels invertierter Liste

Es stellt sich die Frage, wie 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. Ist nach dem Vorkommen eines bestimmten Wortes gefragt, entspricht das Attribut  $t$  dem Term des gesuchten Wortes, da Term und Wort nicht zwangsläufig äquivalent sind (siehe Abschnitt 3.2.1). 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 Termindex kann auf den Term schnell zugegriffen werden, vorausgesetzt dieser ist



**Abb. 4.2.** Invertierte Listen zu drei Beispieltermen. Jede invertierte Liste enthält die Dokumentindizes oder auch *docIDs* der Dokumente, in denen der jeweilige Term vorkommt (Eigene Abbildung).

im Vokabular enthalten. Trifft dies zu, kann die zugehörige invertierte Liste als Resultat ausgegeben werden, da für alle darin enthaltenen Dokumente  $t_1 = \text{true}$  gilt.

#### AND-Verknüpfungen

Wie sieht nun die Verarbeitung aus, wenn mehrere elementare Anfragen miteinander verknüpft werden? Um dies zu beantworten, wird zunächst der *AND*-Operator betrachtet. Eine zusammengesetzte Anfrage liegt dann in der Form  $(t, t_1) \text{ AND } (s, s_1)$  vor, wie etwa in dem Beispiel *Kontaktadresse AND Seminar*, bei dem  $t = \text{Kontaktadresse}, t_1 = \text{true}$  sowie  $s = \text{Seminar}, s_1 = \text{true}$  gelten. Dies bedeutet, dass alle Dokumente gesucht sind, in denen beide Terme auftauchen.

Hierzu wird der Durchschnitt aus den Ergebnismengen beider elementarer Anfragen gebildet, d.h. der Durchschnitt aus beiden invertierten Listen. Betrachtet man die Abbildung 4.2, so ist der Durchschnitt für  $\text{Kontaktadresse} \cap \text{Seminar}$  gleich der Ergebnisliste  $\{3, 11\}$ .

#### OR-Verknüpfungen

Lautet die Anfrage hingegen  $(t, t_1) \text{ OR } (s, s_1)$  bzw. *Kontaktadresse OR Seminar*, so sind alle Dokumente gesucht, in denen entweder  $t_1, s_1$  oder auch beide Terme vorkommen. Die zu verwendende Mengenoperation ist deshalb die Vereinigung  $D_{t,t_1} \cup D_{s,s_1}$  bzw.  $\text{Kontaktadresse} \cup \text{Seminar}$ , was bedeutet dass die invertierten Listen, die als Ergebnis für die beiden elementaren booleschen Anfragen zurückgeliefert werden, vereinigt werden. Bezogen auf Abbildung 4.2 lautet die Ergebnismenge für das Beispiel *Kontaktadresse OR Seminar*  $\{1, 2, 3, 5, 7, 9, 11, 15, 22, 25, 30, 32\}$ . Bei beiden Listenoperationen werden die Indizes in den Ergebnislisten sortiert und Duplikate entfernt ([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 jedoch sehr viel Laufzeit kosten, wenn das Archiv viele Dokumente enthält: Es muss über die gesamte Sammlung iteriert und für jedes Dokument geprüft werden, ob es in der invertierten Liste für den Term *Seminar* auftaucht. Der alleinstehende *NOT*-Operator ist aufgrund dessen so ineffizient, dass er bei den meisten booleschen Information-Retrieval-Systemen nur im Zusammenhang mit einem binären Operator zugelassen ist. Da die Kombination *OR NOT* keinen Sinn macht, wenn ein Term ausgeschlossen werden soll, ist dieser Operator in der Regel ausschließlich *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). Ergebnismenge wäre demnach  $\{2, 7, 9, 15, 22, 25, 30\}$ , d.h. die Dokumente 3 und 11 wurden ausgeschlossen.

### *Komplex geschachtelte Ausdrücke*

Da sowohl Vereinigung als auch Durchschnitt eine neue Liste 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. So wird das nachfolgende *AND* auf die Ergebnisliste des vorherigen Durchschnitts angewandt, was der folgenden Klammerung entspricht:

*(Seminar AND Termin) AND Kontaktadresse*

Auf diese Weise werden die Listen, über die iteriert wird, 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 bietet, da bei der Vereinigung zweier Listen ohnehin über alle Elemente iteriert werden muss. Die Verarbeitungsreihenfolge entspricht der folgenden Klammerung: *(Kontaktadresse OR Seminar) OR Termin*

Ist die Anfrage hingegen gemischt, wie etwa in dem Beispiel *(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 zusammengehörige Wörter verarbeiten. Über die interne Verarbeitung hat der Nutzer jedoch keinerlei Einblick: Das Information-Retrieval-System kann so realisiert sein, dass es die aus den Wörtern der Anfrage isolierten Terme mit *OR* verknüpft, es kann diese jedoch genauso gut mit *AND* verbinden ([Hen08], S.171).



## Das Vektorraummodell

Dieses Kapitel stellt mit dem Vektorraummodell ein weiteres klassisches Information-Retrieval-Verfahren vor.

### 5.1 Funktionsprinzip

Der erste Schritt besteht wie beim booleschen Retrieval in der Indexierung der Dokumente und Terme, denn die Ermittlung des Vokabulars der Sammlung ist Ausgangsgrundlage für das weitere Vorgehen.

Wie der Name bereits nahelegt, basiert das Funktionsprinzip auf Vektoren. Die Grundidee besteht darin, sowohl für die Suchanfrage als auch für jedes Dokument einen reellen Vektor, d.h. einen Vektor bestehend aus reellen Zahlen, zu erstellen und anschließend zu ermitteln, zu welchem Dokumentvektor bzw. zu welchen Dokumentvektoren der Anfragevektor oder Query-Vektor die größte Ähnlichkeit besitzt. Die Länge eines Vektors entspricht hierbei der Anzahl Terme im Vokabular, da im Vektor zu jedem Term dessen Gewicht eingetragen wird. Die Berechnung und Funktion des Gewichts werden in Abschnitt 5.4 erklärt.

Im Gegensatz zum booleschen Retrieval können die Resultate des Vektorraummodells basierend auf den Ähnlichkeitswerten in eine Rangfolge gebracht werden, d.h. ein Ranking ist bei diesem Verfahren problemlos möglich ([Fer03], S.62-63).

### 5.2 Vektor und Vektorraum

Da das Funktionsprinzip des Modells auf Vektoren basiert, werden in diesem Abschnitt 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). In dieser Definition werden die Vektoroperationen Addition und skalare Multiplikation als bekannt vorausgesetzt, darum seien sie zuvor kurz vorgestellt. Die Definitionen 5.1 und 5.2 stammen beide aus [Jän84], S.18.

**Definition 5.1.** (*Addition*)

Sind  $(x_1, \dots, x_n)$  und  $(y_1, \dots, y_n)$   $n$ -Tupel reeller Zahlen, so werde deren Summe durch

$$(x_1, \dots, x_n) + (y_1, \dots, y_n) = (x_1 + y_1, \dots, x_n + y_n)$$

erklärt.

**Definition 5.2.** (*Skalare Multiplikation*)

Ist  $\lambda \in \mathbb{R}$  und  $(x_1, \dots, x_n) \in \mathbb{R}^n$ , so erklären wir  $\lambda(x_1, \dots, x_n) = (\lambda x_1, \dots, \lambda x_n) \in \mathbb{R}^n$ .

Ein Vektorraum kann mithilfe der soeben gezeigten Addition und skalaren Multiplikation wie in Definition 5.3 ([Jän84], S.22) angegeben definiert werden.

**Definition 5.3.** (*Vektorraum*)

Ein Tripel  $(V, +, \cdot)$ , bestehend aus einer Menge  $V$ , einer Abbildung (genannt *Addition*)

$$+ : V \times V \rightarrow V$$

$$(x, y) \mapsto x + y$$

und einer Abbildung (genannt *skalare Multiplikation*)

$$\cdot : \mathbb{R} \times V \rightarrow V$$

$$(\lambda, x) \mapsto \lambda x$$

heißt reeller Vektorraum, wenn für die Abbildungen  $+$  und  $\cdot$  die folgenden acht Axiome gelten:

$$1. (x + y) + z = x + (y + z) \quad \forall x, y, z \in V$$

$$2. x + y = y + x \quad \forall x, y \in V$$

$$3. \text{ Es gibt ein Element } 0 \in V \text{ (genannt „Null“ oder „Nullvektor“) mit } \\ x + 0 = x \quad \forall x \in V$$

$$4. \text{ Zu jedem } x \in V \text{ gibt es ein Element } -x \in V \text{ mit } x + (-x) = 0$$

$$5. \lambda(\mu x) = (\lambda\mu)x \quad \forall \lambda, \mu \in \mathbb{R}, x \in V$$

$$6. 1x = x \quad \forall x \in V$$

$$7. \lambda(x + y) = \lambda x + \lambda y \quad \forall \lambda \in \mathbb{R}, x, y \in V$$

$$8. (\lambda + \mu)x = \lambda x + \mu x \quad \forall \lambda, \mu \in \mathbb{R}, x \in V$$

Nachdem bekannt ist, was ein Vektorraum ist, kann nun auf Vektoren eingegangen werden. Ein Vektor  $\vec{v} \in V$  ist ein Element des Vektorraums  $(V, +, \cdot)$ , wenn Addition und skalare Multiplikation die Axiome 1. - 8. aus Definition 5.3 erfüllen. Zwei Vektoren, die unterschiedlich viele Elemente enthalten, z.B.  $\vec{v}_1 = (1, 2)$  und

$\vec{v}_2 = (1, 2, 3)$  liegen deshalb nicht im selben Vektorraum, weil sie sich weder addieren noch multiplizieren lassen. Alle Vektoren im Vektorraummodell, d.h. alle Anfrage- und Dokumentvektoren, liegen hingegen im selben Vektorraum, da sie auf demselben Vokabular aufbauen und demnach die gleiche Länge besitzen, welche der Anzahl Terme im Vokabular entspricht.

### 5.3 Definition Vektorraummodell

Das bereits grob vorgestellte, auf Ähnlichkeiten von Vektoren basierende Funktionsprinzip, lässt sich mathematisch mithilfe von Attributen beschreiben. Attribute stellen im Vektorraummodell eine Abbildung der Dokumentenmenge  $D$  auf die reellen Zahlen  $\mathbb{R}$  dar, weshalb der Wertebereich der Attribute im Gegensatz zum booleschen Retrieval auf die reellen Zahlen beschränkt ist. Das Vektorraummodell lässt sich mittels Attributen wie folgt definieren ([Fer03], S.63):

**Definition 5.4.** (*Vektorraummodell mit Attributen*)

*Sei  $D = d_1, \dots, d_m$  eine Menge von Dokumenten oder Objekten und  $A = A_1, \dots, A_n$  eine Menge von Attributen  $A_j : D \rightarrow \mathbb{R}$  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  $w_i = (w_{i,1}, \dots, w_{i,n}) \in \mathbb{R}^n$  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  $s : \mathbb{R}^n \times \mathbb{R}^n \rightarrow \mathbb{R}$  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 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 Termmenge oder Vokabular  $T$  auffassen.

### 5.4 Gewichte

In diesem Abschnitt wird erklärt, wie Terme gewichtet werden und wozu diese geschieht.

Bei Gewichten handelt es sich, wie bereits beschrieben, um reelle Zahlenwerte. Sie geben 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. Deshalb wird diese pro Vorkommen von  $t$  in  $d$  um eins erhöht ([CDM08], S.71).

Es erscheint intuitiv als 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. Diese Gewichtungsmethode erlaubt eine viel genauere Differenzierung als eine simple Unterscheidung zwischen true und false, d.h. zwischen Vorkommen und Fehlen eines Terms, wie es beim booleschen Retrieval der Fall ist.

### 5.4.2 Dokumenthäufigkeit

Die Termhäufigkeit alleine stellt allerdings keine besonders zuverlässige Gewichtungsmethode dar: Eine Bewertung, die ausschließlich von der Termhäufigkeit ausgeht, lässt außer Acht, dass nicht alle Terme gleich wichtig sind. Taucht ein Term beispielsweise in jedem Dokument auf, kann dieser nicht besonders aussagekräftig sein. Deshalb 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

Um das Gewicht entsprechend der Dokumenthäufigkeit zu verringern, wird als reduzierender Faktor die sogenannte invertierte oder inverse Dokumenthäufigkeit (engl. inverse document frequency, kurz IDF) verwendet. Die invertierte Dokumenthäufigkeit  $idf_t$  des Terms  $t$  berechnet sich wie in Formel 5.1 gezeigt ([Fer03], S.68).

$$idf_t = \frac{1}{df_t} \quad (5.1)$$

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 ([CDM08], S.108). In der Regel beträgt die Basis des Logarithmus 10, dies spielt aber letztendlich für das korrekte Ranking der Resultate keine Rolle ([CDM08], S.109).

$$idf_t = \log \frac{N}{df_t} \quad (5.2)$$

#### 5.4.4 TF-IDF-Gewichtung

Die vollständige Gewichtungsmethode besteht darin, die Termhäufigkeit mit der invertierten Dokumenthäufigkeit zu multiplizieren. Alle Formeln dieses Typs werden als TF-IDF Gewichtung (engl. tf-idf weighting) bezeichnet ([Fer03], S.71, [CDM08], S.109). Das Gewicht für Term  $t$  in Dokument  $d$  berechnet sich somit wie in Formel 5.3 gezeigt ([CDM08], S.109).

$$tf - idf_{t,d} = tf_{t,d} \times idf_t \quad (5.3)$$

Verwendet man für die invertierte Dokumenthäufigkeit den unmodifizierten Wert aus Formel 5.1, so ergibt sich hieraus die Berechnung 5.4.

$$tf - idf_{t,d} = \frac{tf_{t,d}}{df_t} \quad (5.4)$$

Für die modifizierte Formel 5.2 lautet die TF-IDF-Gewichtung wie in Formel 5.5 angegeben.

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

Sei  $T$  die Termmenge der Sammlung oder 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  $w_i = (tf - idf_{1,i}, \dots, tf - idf_{n,i})$  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 in der Anfrage enthaltenen Term ein anderer Ähnlichkeitswert erzielt wird, werden die Ähnlichkeitswerte 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, weshalb hierfür eine geeignete Ähnlichkeitsfunktion benötigt wird. Es bietet sich an, bekannte Distanzfunktionen für Vektoren zu verwenden, da die Ähnlichkeit umso größer ist, je kleiner die Distanz ausfällt.

### 5.6.1 Euklidischer Abstand

Eine typische Distanzfunktion für Vektoren ist der euklidische Abstand, welcher in Formel 5.6 gezeigt ist ([CDM08], S.121).

$$|\vec{x} - \vec{y}| = \sqrt{\sum_{i=1}^M (x_i - y_i)^2} \quad (5.6)$$

Allerdings besitzt der euklidische Abstand den gravierenden Nachteil, dass die Länge der Vektoren eine Rolle spielt. Reiht man beispielsweise den Inhalt eines Dokumentes  $d1$  zweimal aneinander, so besitzt das entstandene Dokument  $d2$  für jeden Term die doppelte Termhäufigkeit und der Dokumentvektor  $\vec{d2}$  ist darum länger als  $\vec{d1}$ . Das resultiert darin, dass  $d2$  bei einer entsprechenden Suchanfrage einen deutlich höheren Ähnlichkeitswert erzielt als  $d1$ , obwohl sich die Dokumente inhaltlich nicht unterscheiden. 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 für das Vektorraummodell.

### 5.6.2 Cosinus-Maß

Um den Einfluss der Vektorlänge zu eliminieren, wird in der Regel stattdessen das Cosinus-Maß (engl. cosine similarity) verwendet, welches den Cosinus des zwischen den Vektoren eingeschlossenen Winkels berechnet und darum von der Vektorlänge unabhängig ist. Das Cosinus-Maß entspricht dem Skalarprodukt der normalisierten Vektoren ([CDM08], S.112). Ein normalisierter Vektor ist ein Vektor, der durch seine euklidische Länge dividiert wird und darum immer die Länge 1 besitzt. Zur Berechnung des Cosinus-Maßes werden sowohl das Skalarprodukt als auch die euklidische Länge als bekannt vorausgesetzt, darum werden beide an dieser Stelle erklärt.

#### Euklidische Länge

Sei  $\vec{x}$  ein Vektor, dann wird seine euklidische Länge wie in Formel 5.7 angegeben berechnet ([CDM08], S.111).

$$|\vec{x}| = \sqrt{\sum_{i=1}^M x_i^2} \quad (5.7)$$

## Skalarprodukt

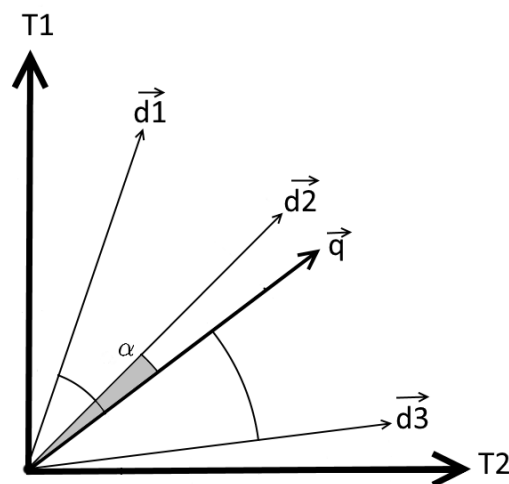
Das Skalarprodukt zweier Vektoren  $\vec{x}$  und  $\vec{y}$  wird wie in Formel 5.8 gezeigt berechnet ([CDM08], S.111).

$$\vec{x} \cdot \vec{y} = \sum_{i=1}^M x_i y_i \quad (5.8)$$

## Berechnung

Da das Cosinus-Maß wie bereits erwähnt das Skalarprodukt der normalisierten Vektoren ist, lautet die Formel wie in 5.9 angegeben ([CDM08], S.111). Hierbei bezieht sich *sim* auf das englische Wort *similarity*, was übersetzt Ähnlichkeit bedeutet.

$$\text{sim}(\vec{x}, \vec{y}) = \frac{\vec{x} \cdot \vec{y}}{|\vec{x}| \cdot |\vec{y}|} \quad (5.9)$$



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

Um das Cosinus-Maß besser nachvollziehen zu können, hilft eine grafische Veranschaulichung. Abbildung 5.1 zeigt einen Vektorraum mit zwei Termen als Achsen, wodurch der Raum sich zweidimensional darstellen lässt. Bei mehr als zwei Termen ist dies schon nicht mehr möglich, da jeder Term eine eigene Achse im Vektorraum bildet. In der Realität gibt es meist weitaus mehr Achsen als in der Abbildung, da das Vokabular tausende Terme beinhalten kann.

Der gezeigte Vektorraum beinhaltet insgesamt drei Dokumentvektoren  $\vec{d}_i$  sowie den Anfragevektor  $\vec{q}$  als Elemente, wobei alle abgebildeten Vektoren bereits normalisiert sind. Unter Verwendung des Cosinus-Maßes ergibt sich für  $\text{sim}(\vec{d}_2, \vec{q}) = \cos(\alpha)$  der höchsten Wert, da  $\alpha$  der kleinste aller Winkel zwischen einem Dokumentvektor  $\vec{d}_i$  und  $\vec{q}$  ist. Hier wird deutlich, dass ausschließlich der Winkel, nicht aber die Länge ausschlaggebend ist. Damit wird  $d_2$  als erstes Ergebnisdokument ausgegeben ([SB10], S.55-56).



## Bewertung eines Information-Retrieval-Systems

In diesem Kapitel wird erläutert, wie sich Information-Retrieval-Systeme bewerten und vergleichen lassen.

Nachdem mit dem booleschen Retrieval und dem Vektorraummodell bereits zwei klassische Verfahren vorgestellt wurden, mit denen sich Information-Retrieval-Systeme realisieren lassen, 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 hinsichtlich dessen Qualität beurteilen zu können, muss die Relevanz der zurückgelieferten Ergebnisse eingestuft werden. Hiermit eröffnet sich das Hauptproblem: Wann ist ein Dokument relevant? Allgemein formuliert lässt sich dies so beantworten: Ein Dokument ist relevant, wenn es das in der Anfrage formulierte Informationsbedürfnis des Nutzers 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  $r : D \times Q \rightarrow R$ , wobei  $D = D_1, \dots, d_m$  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.*

Da zum Bestimmen der Relation die Beurteilung von Experten erforderlich ist, lässt diese Definition 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 beiden zur Bewertung notwendigen Evaluierungsmaße Precision und Recall vorgestellt.

### 6.2.1 Precision

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

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

### 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).

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

### 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 handelt es sich hierbei um Relevanz und Rückgewinnung ([Eng14]).

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

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

Das Precision-Evaluierungsmaß lässt sich anhand Tabelle 6.1 wie folgt beschreiben:

$$Precision = \frac{tp}{tp + fp} \quad (6.3)$$

Präzision berechnet demnach, wie viele der als positiv eingestuftten Ergebnisse, d.h. inklusive der *false positives*, auch tatsächlich *true positives*, d.h. relevante Resultate, sind.

Die Beschreibung für das Recall-Evaluierungsmaß anhand Tabelle 6.1 lautet wie folgt:

$$Recall = \frac{tp}{tp + fn} \quad (6.4)$$

Für die Bestimmung des Recalls werden die *true positives*, d.h. alle vom System korrekt als positiv eingestuftten Ergebnisse, durch die Gesamtheit der relevanten Resultate dividiert. Wie sich aus der Tabelle entnehmen lässt, sind dies die *true positives* und die *false negatives*, d.h. auch jene Dokumente, die fälschlicherweise vom System als nicht relevant eingestuft wurden. Der Recall gibt deshalb kurz gesagt die Trefferquote an.

### 6.3 Zwei Evaluierungsmaße

Um ein System aussagekräftig 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% erzielen, wenn nur ein einziges Dokument gefunden wurde und dieses ein Treffer war. Vielleicht wurden hier 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 die folgenden Voraussetzungen gegeben sind ([CDM08], S.140):

- Vorgegebene Dokumentsammlung
- Feste Menge von Test-Anfragen
- Eine Relation  $r$ , die 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 Anfragenmenge zu groß, um sie im Rahmen dieser Arbeit bewältigen zu können, da Anfragen aus beliebigen Wörtern bestehen und zudem beliebig tief geschachtelt werden können. Aufgrund dieser Punkte musste auf eine Bewertung des Systems anhand von Precision und Recall in dieser Arbeit verzichtet werden.

## Implementierung

In diesem Kapitel wird beschrieben, auf welche Weise das System zur Informationssuche in einem Dokumentenarchiv basierend auf Textinhalt sowie Metadaten 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 erfordert das Lösen zweier Teilprobleme: Zum einen das Realisieren der Metadaten suche und zum anderen das Realisieren der Freitextsuche. Beides wurde mit unterschiedlichen Verfahren gelöst, da die Suchen unterschiedliche Anforderungen besitzen. Im Falle der Metadaten suche wurde sich für boolesches Retrieval entschieden, bei der Freitextsuche hingegen für das Vektorraummodell.

Bevor erklärt wird, aus welchen Gründen diese Entscheidung getroffen wurde, ist es wichtig, zunächst eine Vorstellung zu haben, wie die zu durchsuchenden Dokumente des Archivs beschaffen sind, weshalb Abbildung 7.1 ein Beispiel zeigt. Hier wird deutlich, dass jedes Dokument spezifische Eigenschaften besitzt, die jeweils durch einen Attributnamen und den zugehörigen Attributwert dargestellt sind. Beispielsweise ist „datum“ ein Attributname und „(\"Wed, 22 Jun 2017 07:47:51 +0200\")“ der zugehörige Attributwert.

Die Gesamtheit aller Attribute eines Dokuments bildet den Metadatenteil. Darauf folgt der unstrukturierte Freitextpart.

**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
9 (ABSENDER-NAME "Max_Muster")
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 Regel enthalten die Dokumente weitaus mehr Attribute als in diesem Beispiel, zudem kann sich deren Inhalt auch über mehrere Zeilen erstrecken. Zusätzlich enthalten die Dokumente oft Kommentare, die vom System als Freitext interpretiert werden.

Bei Beispiel 7.1 handelt es sich zwar um eine E-Mail, dies muss jedoch nicht der Fall sein. Die Attribute können inhaltlich vollkommen abweichend ausfallen, abhängig davon welches Archiv durchsucht wird. Vorausgesetzt, die Dokumente sind semistrukturiert, ist jede Art von Dokumentsammlung denkbar. Einzige Bedingung für die korrekte Funktionsweise des Systems ist, dass die im Archiv enthaltenen Dokumente die in 7.2 vorgegebene Struktur erfüllen.

#### Listing 7.2. Dokumentstruktur

```

1 (Attributname_1 Attributwert_1)
2 ....
3 (Attributname_n Attributwert_n)
4
5 Freitext

```

## 7.2 Initialisierungsschritte

Beim Starten des Programms werden zunächst einige Initialisierungsschritte ausgeführt, die im folgenden Abschnitt beschrieben werden.

### 7.2.1 Erstellen des Dokument-Dictionaries

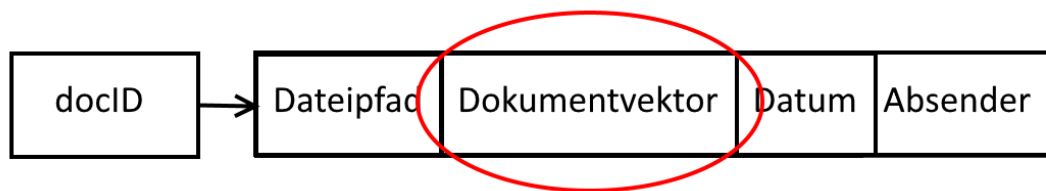
Zunächst wird das Dokument-Dictionary angelegt. Dieses wird intern durch eine Hash-Table realisiert, da dies in Lisp einem Dictionary am nächsten kommt. Zu jedem Dokument werden darin die folgenden Punkte erfasst:

- *docID*: Das Zuweisen einer einmaligen *docID* in Form eines fortlaufenden Integer-Wertes. Die *docID* bildet den Hash-Key des Dokuments.
- Dateipfad
- Dokumentvektor (zu Beginn noch nicht initialisiert).
- Datum: Optionale Information, die zur verbesserten Ergebnisanzeige dient.
- Absender: ebenfalls optionale Information, die nur der Ergebnisanzeige dient.

Dateipfad, Dokumentvektor, Datum und Absender werden in einem Struct zusammengefasst. Ein Struct ist eine Datenstruktur in Lisp, die aus selbst definierten

und mit Werten belegbaren Slots besteht und die sich darum ideal eignet, um zusammengehörige Werte kompakt und schnell abrufbar zu speichern. Das Struct bildet den Hash-Value des Dokuments.

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 bewusst so flexibel wie möglich gehalten, um auch andere Dateien verarbeiten zu können. Zur Veranschaulichung zeigt Abbildung 7.1 den Aufbau eines Eintrags im Dokument-Dictionary.



**Abb. 7.1.** Eintrag im Dokument-Dictionary. Die Struct-Slots Datum und Absender können, falls nicht existent, auch leer bleiben. Im rot markierten Bereich wird später der Dokumentvektor eingetragen (Eigene Abbildung).

### 7.2.2 Verarbeiten der Dokumente

Anschließend wird über alle Dokumente im Dokument-Dictionary iteriert, um die folgenden Schritte auszuführen:

1. Aufteilen in Metadaten und Freitext
2. Verarbeiten der Metadaten
3. Verarbeiten des Freitextes

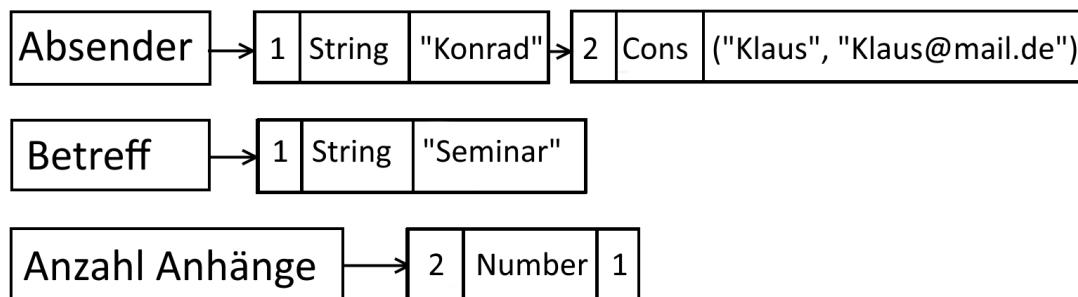
#### Verarbeiten der Metadaten

Die Metadaten werden zunächst in Attributname und Attributwert zerlegt, um sie weiterverarbeiten zu können. Für die Metadatensuche wurde boolesches Retrieval (siehe Kapitel 4) eingesetzt, da eine klare Bedingung definiert werden kann: Der gesuchte Attributname muss im Dokument auftreten, d.h. jedes Dokument wird auf den Wertebereich  $\{true, false\}$  abgebildet. Allerdings muss zusätzlich zum Auftreten noch geprüft werden, ob der Attributwert inhaltlich mit der Suchanfrage übereinstimmt bzw. ob darin Teile der Anfrage auftauchen. Um dies zu lösen, wurde auf eine modifizierte Form der invertierten Liste (siehe Abschnitt 4.3.2) zurückgegriffen. Eine Term-Dokument Inzidenz Matrix wurde hingegen von vornherein aufgrund des zu hohen Speicherbedarfs ausgeschlossen.

Zum Verarbeiten der Metadaten wird eine Hash-Table mit den Attributname als Hash-Keys erstellt, in der die folgenden Punkte, zusammengefasst in einem Struct, erfasst werden:

- *docID*: Eindeutiger Index des Dokuments, in dem das Attribut auftritt. Dieser gehört standardmäßig in die invertierte Liste.
- Typ: Dieser Eintrag gibt an, von welchem Datentyp der Attributwert ist, da dies über die Art der Suche darin entscheidet.
- Inhalt: Enthält den Attributwert. Dieser ist in der Regel so klein, dass er problemlos darin gespeichert werden kann und im Gegensatz zum Freitext keine weitere Verarbeitung erfordert. Durch das Unterlassen einer Indexierung der Attributwerte wird zudem Speicher gespart.

Abbildung 7.2 zeigt die modifizierte invertierte Liste anhand einiger Beispiel-Attribute. Hierbei ist zu beachten, dass beim Speichern der Attributnamen Groß- und Kleinschreibung keine Rolle spielt, d.h. für die Keywords „absender“ und „AB-SENDER“ wird nur ein einziger Hash-Key angelegt. Mit dem Wort Keywords wird ausgedrückt, dass beide Begriffe sich auf ein und dasselbe Attribut beziehen, auch wenn die Namen unterschiedlich ausfallen. Kommen beide Keywords innerhalb eines Dokuments vor, gibt es unter dem entsprechenden Hash-Key zwei Einträge mit der gleichen *docID*, aber unterschiedlichen Inhalten. Das Ignorieren von Groß- und Kleinschreibung wird in diesem Information Retrieval System im Allgemeinen angewendet, um möglichst fehlertolerant gegenüber abweichenden Nutzereingaben zu sein bzw. um auch Wörter, die sich nur durch Groß- und Kleinschreibung vom Suchbegriff unterscheiden, als relevante Treffer zu erkennen.



**Abb. 7.2.** Modifizierte invertierte Liste zur Realisierung der Metadaten-suche. Statt der *docIDs*, wie es bei invertierten Listen üblich ist, werden Structs der Form (docID, Typ, Inhalt) gespeichert (Eigene Abbildung).

## Verarbeiten des Freitextes

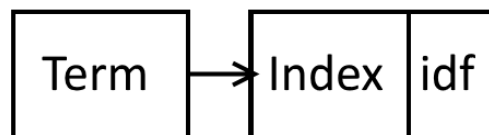
Für die Freitextsuche wurde das Vektorraummodell (siehe Kapitel 5) gewählt, da dieses Teiltreffer sowie ein Ranking der Ergebnisse ermöglicht. Grundvoraussetzung für das Verfahren ist die Bestimmung des Vokabulars. Da jedes Dokument bereits in Metadaten und Freitext aufgeteilt wurde, liegen die Freitexte der Sammlung bereits isoliert vor. Es wird über diese Liste iteriert und pro Freitext werden jeweils die folgenden Schritte ausgeführt:



- Zerlegung des Textes in Terme, wobei auf eine Lemmatisierung (siehe Abschnitt 3.2.1) verzichtet wurde, da dies den Rahmen der Arbeit sprengen würde. Zur zeilenweisen Zerlegung der Texte wurde das Package `split-sequence` verwendet ([Ion16]).
- Zu jedem Term wird geprüft, ob dieser bereits im Vokabular enthalten ist. Falls nicht, wird er hinzugefügt, vorausgesetzt es handelt sich nicht um ein deutsches oder englisches Stoppwort.

Stoppwörter (siehe Abschnitt 3.2.2) wurden bewusst aus dem Vokabular entfernt, um Speicherplatz zu sparen und die Suche zu beschleunigen, denn je kleiner die Dokument- und Anfragevektoren ausfallen, desto schneller erfolgt die Berechnung der Ähnlichkeitswerte. Die englischen Stoppwörter stammen aus [web13], die deutsche Stoppwortliste aus [Koh09].

Nachdem das Vokabular vollständig bestimmt wurde, werden die darin vorkommenden Terme indexiert. Hierfür wird zunächst ein Term-Dictionary angelegt, wobei auch hier die zugrunde liegende Datenstruktur eine Hash-Table ist. Anschließend wird über das Vokabular iteriert und pro Term ein Eintrag in der Form (Index, idf=0) angelegt. Der Index ist hierbei ein fortlaufender Integer-Wert, der den Term eindeutig identifiziert und dessen Position im Dokument- oder Anfragevektor bestimmt. Der idf-Wert muss erst noch ermittelt werden, darum wird dieser mit 0 initialisiert. Abbildung 7.3 zeigt den Aufbau eines Eintrags.



**Abb. 7.3.** Struktur für einen Eintrag im Term-Dictionary. Der Termname bildet den Hash-Key, Index und idf sind in einem Struct zusammengefasst. Der Index gibt die Position des Terms im Dokumentvektor an (Eigene Abbildung).

Anschließend muss das Term-Dictionary mit idf-Werten gefüllt werden, weshalb über alle Freitexte iteriert wird und jeweils folgende Schritte ausgeführt werden:

- Anlegen eines Dokumentvektors, realisiert mittels Hash-Table, welcher die Term-Indizes als Hash-Keys und deren noch zu bestimmende tf-idf-Gewichte als Hash-Values besitzt.
- Tritt ein Term zum ersten Mal in der Sammlung, wird der zuvor mit 0 initialisierte 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 um 1 erhöht.
- Beim ersten Auftreten im Dokument wird der entsprechende Eintrag im Dokumentvektor auf die Termhäufigkeit 1 gesetzt. Die Position im Vektor wird durch den Index des Terms (gespeichert im Term-Dictionary) vorgegeben.

- Für jedes erneute Auftreten im Dokument wird die Termhäufigkeit im Dokumentvektor um 1 erhöht.
- Ist der Freitext des aktuellen Dokuments vollständig verarbeitet, kann der bereits angelegte Eintrag an der entsprechenden Stelle im Dokument-Dictionary (rot markiert in Abbildung 7.1) mit dem hier erstellten Dokumentvektor initialisiert werden.

Bei der Datenstruktur für die Dokumentvektoren wurde sich bewusst für eine Hash-Table entschieden, um das Problem der leeren Einträge zu vermeiden und damit Speicher zu sparen. Fehlt ein Term in der Hash-Table, so wird dies bei der Berechnung des Cosinus-Maßes als Gewicht 0 interpretiert.

Noch enthalten die idf-Slots im Term-Dictionary die Dokumenthäufigkeiten statt der idf-Werte. Deshalb werden diese nach Formel 5.2 in den idf-Wert umgerechnet. Es wurde sich für die Formel mit Logarithmus entschieden, um die Werte seltener Terme abzuschwächen. Mit den idf-Werten können nun auch die tf-idf-Werte bestimmt werden, weshalb die Termhäufigkeiten in den Dokumentvektoren nach Formel 5.5 umgerechnet werden. Die Vektoren stehen jetzt fest und können zur Verrechnung mit dem Anfragevektor verwendet werden.

## 7.3 Suche

Nachdem die Initialisierungsschritte ausgeführt wurden, kann der Nutzer die Suche starten. Er sieht auf der Benutzeroberfläche, welche Attribute ihm als Bereiche, in denen gesucht werden kann, zur Verfügung stehen. Das Attribut „Freitext“ ist hierbei immer vorhanden. Metadaten- und Freitextsuche wurden intern unterschiedlich realisiert, darum werden sie getrennt erklärt.

### 7.3.1 Metadaten- und Freitextsuche

Die Metadaten- und Freitextsuche verwendet boolesches Retrieval. Lautet die Anfrage beispielsweise *Absender=„Klaus“*, wird auf die modifizierte invertierte Liste 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 Attributwert wird mit der vordefinierten Lisp-Funktion `search` durchsucht. Die Suche ist erfolgreich, wenn die gesuchte Zeichenkette an einer beliebigen Stelle darin als Substring auftaucht.
2. **Number:** Ist der Inhalt eine Zahl, wird die Suchanfrage, welche stets als String übergeben wird, wenn möglich zum Datentyp Number konvertiert. Hierbei sind auch als Wort ausgeschriebene Zahlen von null bis zwölf konvertierbar. Ist kein Konvertieren möglich, schlägt die Suche sofort fehl, da der Inhalt nicht zur Anfrage passen kann.

3. **Liste:** Eine Liste entspricht dem Datentyp *cons* in Lisp und wird rekursiv durchsucht, um alle darin enthaltenen Elemente typspezifisch zu durchsuchen. Dies können Strings, Zahlen oder Unterlisten sein.

Liegt ein Treffer vor, wird die *docID* des Dokuments der Ergebnisliste hinzugefügt. Hierbei kann es sein, dass pro Teilanfrage mehrere Metadatensuchen durchgeführt werden, da der Nutzer die Anfrage für verschiedene Attribute gleichzeitig stellen kann. Dann hat jedes Attribut 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. Hierzu werden die in Lisp vordefinierten Funktionen *intersection* und *union* verwendet.

Nun liegt das finale Ergebnis für die Metadatensuche 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. Hierzu wurde die vordefinierte Funktion *set-difference* verwendet.

### 7.3.2 Freitextsuche

#### 7.3.3 Erstellen des Query-Vektors

Die Freitextsuche verwendet das Vektorraummodell, darum muss eine Anfrage erst in einen Vektor umgewandelt werden. Wie beim Dokumentvektor wird auch hier eine Hash-Table als Datenstruktur verwendet. Für jeden Term wird dessen Index im Term-Dictionary abgefragt. Vorausgesetzt, der Term existiert im Vokabular, wird dieser Index zum Hash-Key und die Termhäufigkeit in der Anfrage zum Wert. Ausnahme sind Stoppwörter, die bei der Suche ignoriert werden. Der Nutzer erhält in diesem Fall eine Warnmeldung. Anschließend wird die Termhäufigkeit mit dem idf-Wert multipliziert, 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

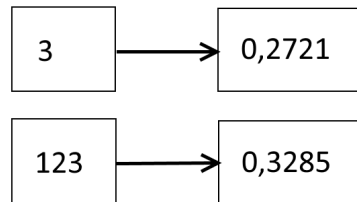
Zum Bestimmen der Suchergebnisse für die Freitextsuche wird über alle Dokumente iteriert, um folgende Schritte auszuführen:

- Zugriff auf den Dokumentvektor
- Berechnung des Cosinus-Maßes (siehe Formel 5.9) zur Bestimmung der Ähnlichkeit zwischen Dokument- und Query-Vektor
- Hinzufügen des Ähnlichkeitswertes (Score) inklusive *docID* zur Ergebnisliste
- Sortieren der Liste, damit sich die ähnlichsten Dokumente vorne befinden

Anfrage: "Kontaktadresse Seminar Mai"



Query-Vektor:



**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 ignoriert. Alle Indizes ungleich 3 und 123 sind als mit 0 gewichtet zu interpretieren (Eigene Abbildung).

## 7.4 Verrechnung der Suchergebnisse

Die Resultate beider Suchverfahren müssen miteinander kombiniert werden, wobei der Score problematisch ist, da die Freitextergebnisse einen besitzen, die Metadaten-Resultate jedoch nicht. Gelöst wurde dies wie folgt:

- Jedes Dokument in der Metadaten-Ergebnisliste erhält den Score 1.
- Ist *AND* ausgewählt, wird der Durchschnitt der Ergebnislisten beider Suchverfahren gebildet und Metadaten-Score sowie Freitextscore werden addiert.
- Ist *OR* ausgewählt, wird die Vereinigung beider Suchen gebildet und Metadaten-Score sowie Freitextscore der Dokumente, die in beiden Ergebnislisten vorkommen, werden addiert.

Da die Anfrage beliebig tief geschachtelt vorliegen kann, ist es möglich, dass sich der Score eines Dokuments mit weiteren Teilanfragen erhöht: Das Gesamtergebnis wird mit jeder neuen Teilanfrage auf dieselbe Weise wie soeben beschrieben verrechnet, d.h. je nach selektiertem Operator (*AND* oder *OR*) wird der Durchschnitt oder die Vereinigung aus Gesamt- und Teilanfrage mit entsprechender Addition der Scores durchgeführt. Demnach erhält ein Dokument, dass für 5 Teilanfragen einen Treffer in der Metadatensuche 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 Scores kann der Nutzer in etwa abschätzen, wie wichtig ein Dokument für seine Anfrage war und anhand des Zahlenwertes - ganzzahlig oder nicht - auch, auf welche Weise der Treffer zustande kam.

## Die Benutzeroberfläche

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

### 8.1 Anforderungen

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

1. Wählen und Festlegen des Suchverzeichnisses
2. Eingabe des Suchtextes
3. Auswählen der Freitextsuche
4. Auswählen der Attributnamen für die Metadatensuche
5. Logische Operatoren zur internen Verknüpfung mehrerer Suchkriterien
6. Logische Operatoren zur externen Verknüpfung mehrerer Teilanfragen
7. Anzeigen der Anfrage
8. Zurücksetzen der Anfrage
9. Starten der Suche
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

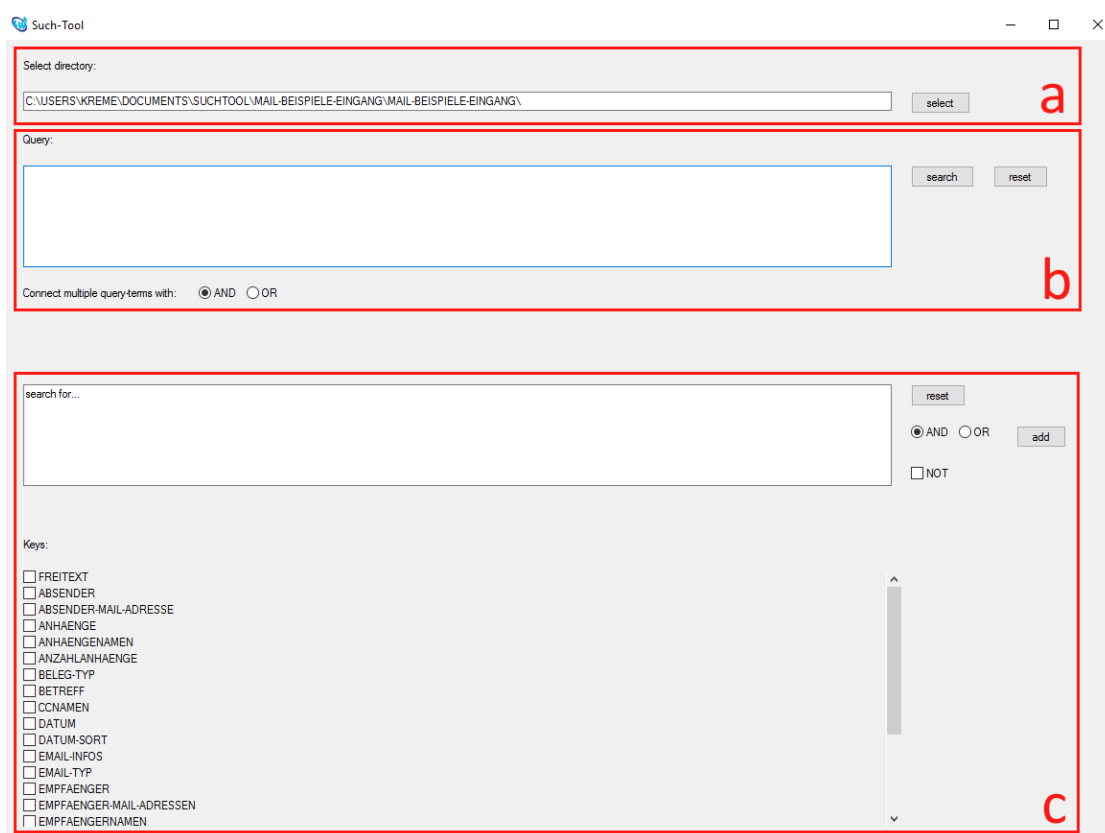
In den folgenden Abschnitt wird erläutert, auf welche Weise diese Punkte realisiert wurden.

### 8.2 Grundaufbau der Oberfläche

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

Der oberste Bereich beinhaltet die Verzeichnisauswahl, da dies der erste vom Anwender auszuführende Schritt ist. In der Mitte befindet sich die Anzeige der Gesamtanfrage, weil sie sich dort sofort im Blickfeld des Nutzers befindet. Die Erstellung der Teilanfragen wurde im unteren Teil der Oberfläche untergebracht, da sich hier die meisten Bedienungselemente befinden, weshalb jede andere Position unweigerlich Einbußen bezüglich Übersichtlichkeit zur Folge hätte.

Funktional zusammengehörige Elemente wurden stets nah beieinander angeordnet, was dem Gesetz der Nähe entspricht. Dieses 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 Erstellung der Teilanfrage (Eigene Abbildung).

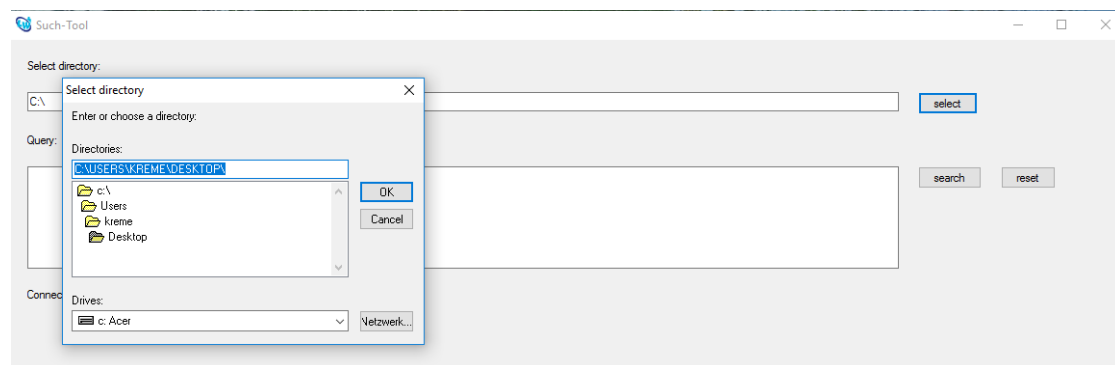
## 8.3 Verzeichnisauswahl

Die Verzeichnisauswahl (siehe Abbildung 8.1 Teil a) besteht aus einem Textfeld und dem rechts daneben befindlichen Button mit der Aufschrift „select“, welcher das Auswahlmenü öffnet. Dieses wird in einem separaten Pop-Up-Fenster angezeigt

(siehe Abbildung 8.2) und bietet 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 anschließend mit „ok“ bestätigen. Alternativ kann er den Dialog über „cancel“ abbrechen. Einmal gewählt, wird der Verzeichnispfad im Textfeld angezeigt und kann nachträglich nicht mehr geändert werden, weshalb der select-Button anschließend deaktiviert wird. Möchte der Nutzer in einem anderen Verzeichnis als bisher suchen, kann er das System einfach erneut starten.



**Abb. 8.2.** Verzeichnisauswahl (Eigene Abbildung)

## 8.4 Suchanfrage

Im unteren Teil der Benutzeroberfläche (siehe Abbildung 8.1 Teil c) befinden sich die Attributnamen. Diese wurden als auswählbare Checkboxes realisiert, welche sich auf einem vertikal scrollbaren Panel befinden. Auf diese Weise ist sichergestellt, dass eine beliebig große Anzahl von Attributnamen angezeigt werden kann. Die Freitextsuche ist über die oberste Checkbox mit der Beschriftung „Freitext“ auswählbar, womit diese wie ein zusätzlicher Attributname behandelt wird und sich damit in die Benutzeroberfläche einfügt. Dies ist gewollt, denn der Anwender soll nicht mitbekommen, dass die Implementierung der Freitextsuche intern anders realisiert wurde als die übrigen Auswahlmöglichkeiten.

### 8.4.1 Zusammenstellung der Teilanfrage

Im Textdisplay wird die Teilanfrage vom Nutzer eingegeben und lässt sich über den reset-Button bequem zurücksetzen. Die ausgewählten Checkboxes bestimmen darüber, in welchen Bereichen nach dem eingegebenen Text gesucht werden soll, wobei beliebig viele auf einmal selektierbar sind.

Im Falle mehrerer ausgewählter Bereiche lassen sich die Teilergebnisse mit *AND*

sowie *OR* verknüpfen. Der Operator lässt sich über zwei Radio-Buttons einstellen, die sich rechts neben dem Texteingabefeld befinden, wobei *AND* die Standardauswahl ist. Zusätzlich lässt sich die Anfrage durch das Auswählen der unterhalb der Radio-Buttons befindlichen Checkbox *NOT* negieren. Der Operator bezieht sich hierbei auf das Gesamtergebnis der erstellten Teilanfrage, d.h. wenn die Verknüpfung der Treffer für die selektierten Attributnamen bzw. für die Freitextsuche bereits erfolgt ist. Möchte man Bereiche einzeln negieren, müssen die Teilanfragen getrennt eingegeben werden.

Das Drücken von „add“ fügt die soeben erstellte Teilanfrage der Gesamtanfrage hinzu. Eine einzelne Teilanfrage ist somit das, was der Nutzer mit dem Betätigen des add-Buttons der Gesamtanfrage, die im oberen Display angezeigt wird, hinzufügt (siehe Abbildung 8.1 Teil b).

### 8.4.2 Beispiel

Ein anhand von Grafiken erläutertes Beispiel hilft, die einzelnen Schritte besser nachvollziehen zu können. Abbildung 8.3 zeigt, wie der Name „Dr. Claus-Peter Wirth“ im Absender oder in der Absender-Mail-Adresse gesucht werden soll. Das Betätigen des add-Buttons schließt die Teilanfrage ab, fügt sie der Gesamtanfrage hinzu und eröffnet die Möglichkeit, weitere zu stellen.

The screenshot shows a search interface. At the top, there is a large text input field containing "Dr. Claus-Peter Wirth". To the right of this field are two radio buttons labeled "AND" and "OR", with "OR" being selected. Below these is a checkbox labeled "NOT". To the right of the radio buttons is a blue button labeled "add". Below the input field, there is a section titled "Keys:" followed by a list of checkboxes: "FREITEXT" (unchecked), "ABSENDER" (checked), "ABSENDER-MAIL-ADRESSE" (checked), and "ANHAENGE" (unchecked). A vertical scrollbar is visible on the right side of the key list.

**Abb. 8.3.** Attributnamen auswählen und intern verknüpfen, hier mit dem *OR*-Operator (eigene Abbildung).

Abbildung 8.4 zeigt, wie der Suchbegriff „dfki“<sup>1</sup> im Bereich Freitext gesucht werden soll. Es erfolgt ein erneutes Hinzufügen der zweiten Teilanfrage durch Drücken von „add“.

## 8.5 Gesamtanfrage

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. Wenn stattdessen *OR*

<sup>1</sup> DFKI = Deutsches Forschungszentrum für Künstliche Intelligenz



dfki

reset

☐ AND ☒ OR ☐ NOT

add

Keys:

☒ FREITEXT

☐ ABSENDER

☐ ABSENDER-MAIL-ADRESSE

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

erwünscht ist, muss diese Auswahl vor dem Hinzufügen einer weiteren Teilanfrage erfolgen, da eine Änderung des Operators im Nachhinein nicht mehr möglich ist! Abbildung 8.5 zeigt, wie die im Display angezeigte Gesamtanfrage für das Beispiel mit dem Operator *AND* aussieht. In dieses Display lässt sich vom Nutzer nichts eingeben, um die korrekte Anzeige der intern gespeicherten Anfrage sicherzustellen.

Query:

(AND (or (= ABSENDER "Dr. Claus-Peter Wirth") (= ABSENDER-MAIL-ADRESSE "Dr. Claus-Peter Wirth")) (= FREITEXT "dfki"))

**Abb. 8.5.** Anzeige der Gesamtanfrage auf dem oberen Display, das sich in Bereich b befindet (eigene Abbildung).

## 8.6 Ergebnis

Der Nutzer kann die Gesamtanfrage über den 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, was in Abbildung 8.6 gezeigt ist. In einer tabellarischen Ansicht werden die Ranking-Position, der Score des Dokuments, wenn möglich Datum und Absender sowie der Dateipfad angezeigt. Die Tabelle ist vertikal scrollbar (siehe Abbildung 8.7), um eine beliebig große Zahl von Ergebnissen anzeigen zu können. Eine Begrenzung wurde nicht vorgenommen, da keine Resultate ausgeschlossen werden sollen. Der Nutzer kann sich an den Ranking-Positionen orientieren und selbst entscheiden, welche Resultate für ihn relevant sind und welche er aufgrund des niedrigen Scores ausschließen möchte. Die Einträge der Tabelle lassen sich per Doppelklick auswählen und werden je nach ausgewähltem Modus auf verschiedene Arten geöffnet. Dieser lässt sich über zwei oben links über der Tabelle angebrachte Radio-Buttons einstellen: „Open directory“ öffnet das Verzeichnis, zu dem das Dokument gehört, und markiert die

Datei, „Open file“ öffnet die Datei hingegen direkt. Das Drücken von „ok“ schließt das Ergebnisfenster.

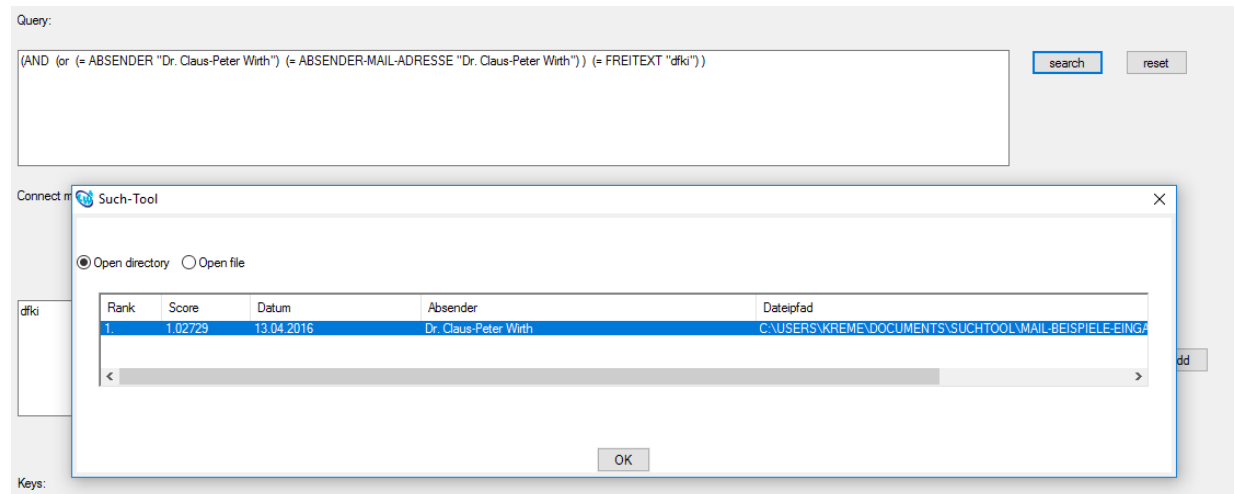


Abb. 8.6. Pop-Up Fenster zur Anzeige der Resultate (eigene Abbildung).

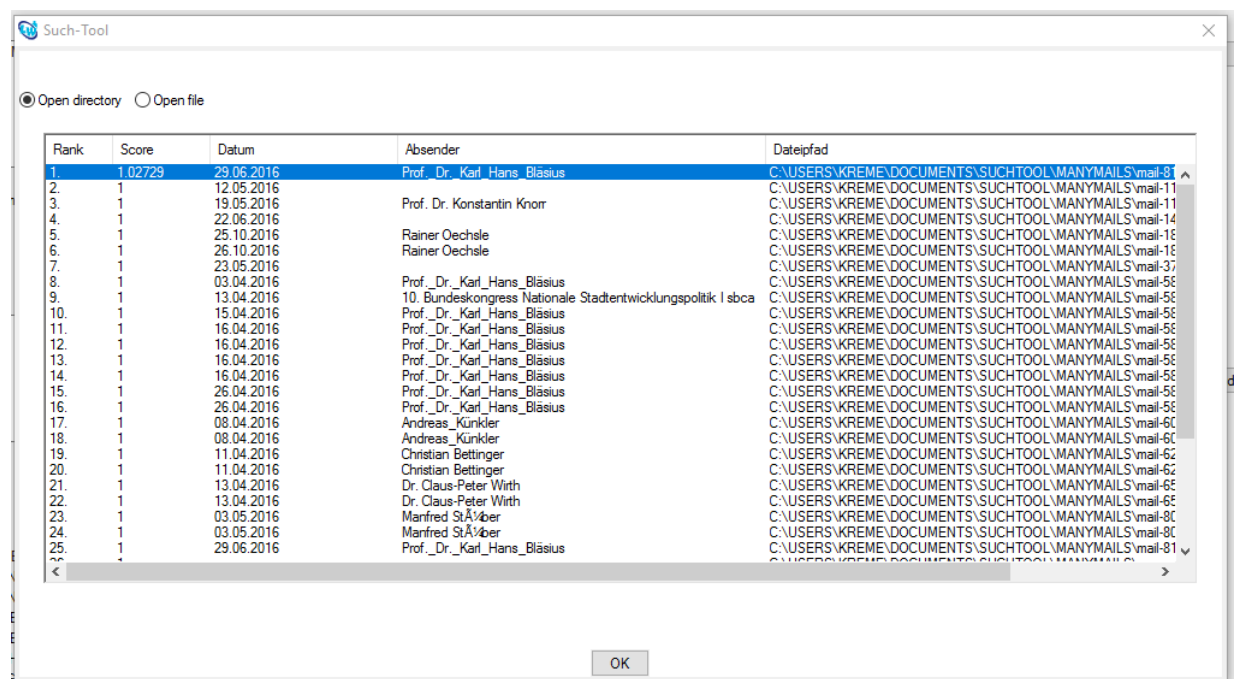


Abb. 8.7. Scrollbare Anzeige, um beliebig viele Treffer anzeigen zu können (eigene Abbildung).

## Zusammenfassung und Ausblick

In diesem Kapitel werden die wesentlichen Ergebnisse der Arbeit zusammengefasst sowie mögliche zukünftige Verbesserungen vorgestellt.

### 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 klassischen 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

Einige Erweiterungen konnten aus zeitlichen Gründen in der Implementierung nicht umgesetzt werden, sind jedoch wünschenswert.

Die Im Kapitel Grundbegriffe beschriebene Lemmatisierung (siehe Abschnitt 3.2.1) würde eine große Verbesserung hinsichtlich des Speicherbedarfs bedeuten, da weniger Terme gespeichert werden müssen. Zudem würde das Ergänzen um Lemmatisierung auch die Suche verbessern, da auch Dokumente mit zum Suchbegriff verwandten Wörtern als Treffer gewertet werden.

Weiterhin wünschenswert ist die Ergänzung um Spelling Correction, d.h. um eine Rechtschreibkorrektur die ähnliche Suchbegriffe vorschlägt, falls der Anwender sich bei der Eingabe vertippt hat. Dies würde die Anwenderfreundlichkeit des Systems erheblich steigern, insbesondere in Fällen in welchen der Anwender seinen Tippfehler nicht bemerkt und unerwartet keine Resultate erhält. Da Nutzer in der Regel Spelling Correction aus der alltäglichen Websuche gewöhnt sind, ist es wünschenswert auch dieses Information-Retrieval-System damit auszustatten.

---

## Literaturverzeichnis

- Aca12. *Academic - Universal-Lexikon*, 2012.  
[http://universal\\_lexikon.deacademic.com/253489/Informationsr%C3%BCckgewinnung](http://universal_lexikon.deacademic.com/253489/Informationsr%C3%BCckgewinnung).
- CDM08. CHRISTOPHER D. MANNING, PRABHAKAR RAGHAVAN, HINRICH SCHÜTZE: *Introduction to Information Retrieval*. Cambridge University Press, 2008.
- Eng14. ENGELHARDT, ALEXANDER: *Crashkurs Statistik - Kreuztabellen / Kontingenztafeln*, 2014.  
<http://www.crashkurs-statistik.de/kreuztabellen-kontingenztafeln/>.
- 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](https://www.uni-bamberg.de/fileadmin/uni/fakultaeten/wiai_lehrstuehle/medieninformatik/Dateien/Publikationen/2008/henrich-ir1-1.2.pdf).
- 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>.
- 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.
- Koh09. KOHLFÜRST, MICHAEL: *PromoMasters Online Marketing - Deutsche StopWords Liste*, 2009.  
<http://www.promomasters.at/blog/stop-words/>.

- 
- met16. *ITWissen.info - Metadaten*, 2016.  
<http://www.itwissen.info/Metadaten-meta-data.html>.
- 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, 2010.
- web13. *99webTools - List of English Stop words*, 2013.  
<http://99webtools.com/blog/list-of-english-stop-words/>.

**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