

Datenbanken und Informationssysteme

Information Retrieval: Konzepte und Beispiele

Burkhardt Renz

Fachbereich MNI
TH Mittelhessen

Sommersemester 2020

Übersicht

- Konzepte des Information Retrieval
- Architektur von Apache Lucene
- Beispiel mit Apache Lucene

Information Retrieval

Information Retrieval befasst sich mit

- der *Repräsentation* von,
- dem *Suchen* in,
- der *Manipulation* von

großen Sammlungen elektronischer Texte und nicht-textueller Dokumente.

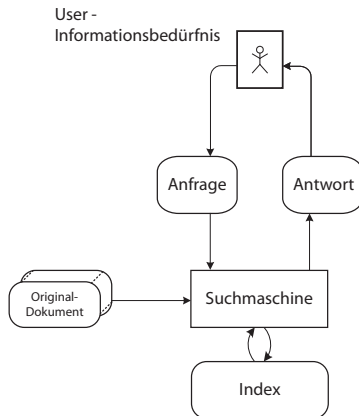
Beispiele

- Suchmaschinen im Internet
- Digitale Bibliothekssysteme
- Experten-Informationssysteme
- Informationssysteme in großen Unternehmen
- Desktop-Suchsysteme

Suchmaschinen im Internet

- Einfache, intuitive Benutzerschnittstelle für die Formulierung einer Suche
- Effizienz: schnelle Antwort
Basis: Caching und Replikation, Web Crawler
- Effektivität: akkurate Antwort
Basis: Algorithmen für Rangfolgen nach Relevanz für die Suchergebnisse

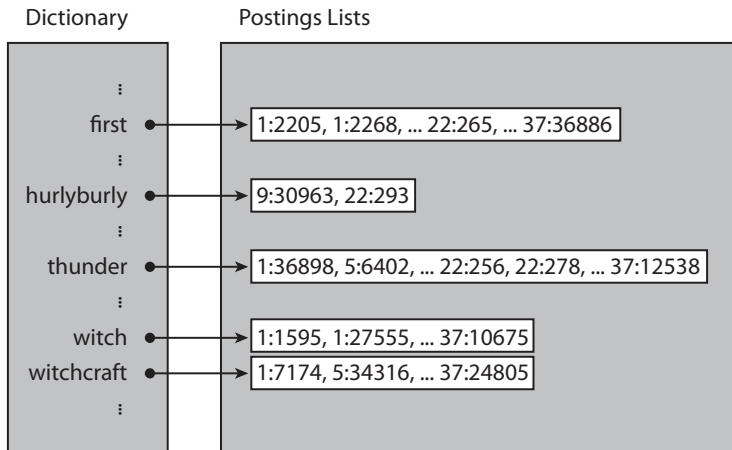
Komponenten eines IR-Systems



Konzept des invertierten Index

- Zerlegung der Originaldokumente in Tokens
- *Dictionary* = Liste aller Terme im Vokabular (alphabetisch sortiert)
- *Postings List* = Liste aller Positionen, an denen ein Term vorkommt (nach Position sortiert)
- Position = Paar aus *DocId* und *Offset*

Beispiel: Shakespeares Dramen



Quelle: Büttcher et al.

Weitere Informationen in einem invertierten Index

- N_t *document frequency*
= Zahl der Dokumente, die Term t enthalten
- $tf_{t,d}$ *term frequency*
= Anzahl von Term t in Dokument d
- l_d *document length*
= Zahl der Tokens in Dokument d
- l_{avg} *average length*
= Durchschnittliche Zahl der Tokens pro Dokument
- N = Zahl der Dokumente in der Sammlung

Tf-idf-Maß

- $tf_{t,d}$ *term frequency*
= Häufigkeit, in der Term t in Dokument d vorkommt.
- idf_t *inverse document frequency*
= Seltenheit des Terms in *allen* Dokumenten

$$idf_t = \log \frac{N}{N_t}$$

- Gewicht des Terms t im Dokument d :

$$tf_{t,d} \times idf_t$$

Arten von Indexen

- *Docid Index:*
Postings List eines Terms enthält nur die Docid d
→ geeignet für Boolesche Suche und einfaches Ranking
- *Frequency Index:*
Postings List eines Terms enthält Docid und Wert der term frequency $(d, f_{t,d})$
→ geeignet für effektive Ranking-Verfahren, aber nicht für Phrasensuche
- *Positional Index:*
Postings List eines Term enthält Docid, term frequency und Positionen im Dokument $(d, f_{t,d}, < p_1, p_2, \dots, p_{f_{t,d}} >)$
→ geeignet für alle Arten der Suche, auch Phrasensuche, sowie diverse Ranking-Verfahren

Verfahren für Suche und Ranking – Übersicht

- *Vector space model* – Vektormodell
- *Proximity ranking*
- Boolesche Suche – Terme kombiniert mit AND, OR und NOT
- Boolesche Suche mit anschließendem Ranking

Vector space model – Idee

- Jeder Term im Dictionary ist die Dimension eines n -dimensionalen Vektorraums
- Ein Dokument ist ein Vektor in diesem Vektorraum:

$$\vec{d}_i = (w_{i,1}, w_{i,2} \dots w_{i,t})$$

mit Gewichten $w_{i,j}$ für die Bedeutung des Terms j in Dokument i

- Eine Suche (Query) ist auch ein Vektor in diesem Vektorraum:

$$\vec{q} = (w_1, w_2 \dots w_t)$$

- Die „Ähnlichkeit“ ist der Winkel zwischen den beiden Vektoren.

Berechnung des Winkels

- Gegeben zwei Vektoren \vec{x} und \vec{y} in einem n -dimensionalen Vektorraum gilt:

$$\vec{x} \cdot \vec{y} = |\vec{x}| \cdot |\vec{y}| \cdot \cos\theta$$

wobei $\vec{x} \cdot \vec{y}$ das Skalarprodukt der Vektoren ist und $|\vec{x}|$ bzw $|\vec{y}|$ die Länge der Vektoren bezeichnet.

- Winkel:

$$\cos\theta = \frac{\sum_{i=1}^t x_i y_i}{\left(\sqrt{\sum_{i=1}^t x_i^2} \right) \left(\sqrt{\sum_{i=1}^t y_i^2} \right)}$$

- Also:

$$\text{sim}(\vec{d}, \vec{q}) = \frac{\vec{d} \cdot \vec{q}}{|\vec{d}| \cdot |\vec{q}|}$$

Proximity ranking – Idee

- Zu einem Termvektor $\vec{q} = (t_1, t_2 \dots t_n)$ definiert man die Überdeckung (*cover*) in einem Dokument als das kleinste Intervall $[u, v]$, das alle Terme enthält.
- Bilden der Rangfolge zu einem Termvektor einer Anfrage:
 - ① Je kürzer die Überdeckung, desto relevanter das Dokument
 - ② Je mehr Überdeckungen im Dokument, desto relevanter das Dokument

Boolesche Suche – Idee

- Terme in der Anfrage werden durch logische Operatoren verbunden:
and, or und not
- Oft wird die Boolesche Suche mit dem Bilden einer Rangfolge im zweiten Schritt verbunden.
Lucene z.B. führt (bei entsprechender Anfrage) Boolesche Suche durch und bildet hinterher mit dem Vektormodell eine Rangfolge (*Scoring*).

Token und Terme

- Voraussetzung ist die Zerlegung der Dokumente in Tokens
- Linguistische Aufbereitung des Tokens – Je nach Sprache unterschiedlich
- Komplexe Angelegenheit, hier nur einige Andeutungen

Groß-/Kleinschreibung und Satzzeichen

- I.B.M. = IBM, U.S. = US, aber C++ = C?
- I'll = ill? oder = i ll? oder = I will?
- Normalisierung durch Kleinschreibung, aber U.S. = US = us?
- Zitate?
„I'll be back“ aus *Terminator*
„I will be back“

Stammformen

- gehen, ging, gegangen → Stammform
- *Porter stemmer* nach Martin Porter
reduziert Worte, aber keine linguistische Stammform
- *Stemmer* gibt es heute für viele Sprachen, z.B. *Snowball stemmer* für europäische Sprachen (auch von Martin Porter)

Funktions- und Stoppworte

- *Funktionsworte* modifizieren andere Worte oder geben grammatikalische Beziehungen an:
z.B. Präpositionen, Artikel, Pronomen, Konjunktionen u.ä.
- Bei der Suche helfen sie in der Regel nicht, man blendet sie aus: *stop words*
- aber:
„to be or not to be that is the question“
→ „question“

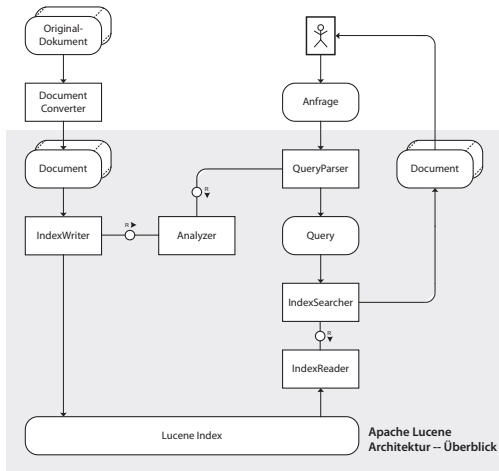
k-gram-Index – ein alternativer Ansatz

- k-gram = Sequenz von k Zeichen, inklusive Leerzeichen
- Die Terme des Dictionary sind alle k-Gramme
- Beispiel: „To be or not to be that is the question“
- 5-Gramme:
„_to_b“, „to_be“, „o_be_“, „_be_o“, „be_or“, „e_or_“,
„_or_n“, „or_no“. . .
- k-gram-Indexe geben oft sehr akkurate Suchergebnisse, gerade bei europäischen oder CJK-Sprachen
- k-Gramme können zur Spracherkennung verwendet werden

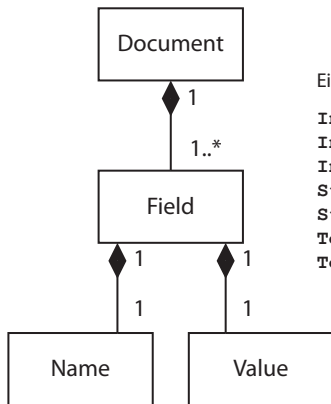
Übersicht

- Konzepte des Information Retrieval
- Architektur von Apache Lucene
- Beispiel mit Apache Lucene

Architektur von Lucene



Lucene Document



Eigenschaften von Feldern

`Index.ANALYZED`

`Index.NOT_ANALYZED`

`Index.NO`

`Store.YES`

`Store.NO`

`TermVector.YES`

`TermVector.WITH_POSITIONS`

Beispiel zweier Documents

```
author = Stefan Büttcher, Charles, L.A. Clarke, Gordon V.  
title = Information Retrieval: Implementing and Evaluating  
Engines  
path = c:\ebooks\eb_Büttcher...  
contents= ....
```

```
author = Christopher D. Manning, Prabhakar Raghavan, Hinrich  
title = Introduction to Information Retrieval  
path = c:\ebooks\eb_Manning...  
contents= ....
```

Analyzer und IndexWriter

- IndexWriter verwendet Analyzer als Strategie für die Token-Ermittlung
- Analyzer:
 - WhitespaceAnalyzer
 - SimpleAnalyzer
 - StopAnalyzer
 - KeywordAnalyzer
 - StandardAnalyzer

Lucene Index

- Segmente = Teilindexe, die gemischt werden können
- Segment enthält:
 - Feldnamen
 - Gespeicherte Feldinhalte
 - Term Dictionary
 - Daten zur Term Frequency
 - Termvektoren
 - ...

Query und QueryParser

- Feld bzw. Default: „title:Information“
- TermQuery: „lucene“
- BooleanQuery: „java and lucene“ oder „+java +lucene“
- PhraseQuery: „title:“Information Retrieval““
- PrefixQuery: „Info*“
- ...

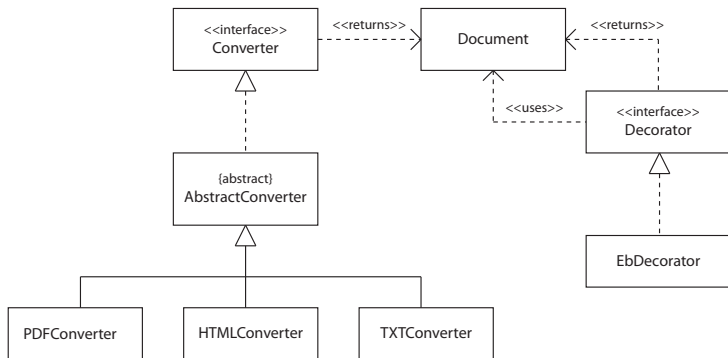
Übersicht

- Konzepte des Information Retrieval
- Architektur von Apache Lucene
- Beispiel mit Apache Lucene

Demonstration Desktopsuche

- Suche nach „Information Retrieval“
- Suche nach „+Information -Retrieval“
- Suche nach „title:“Information Retrieval““

Document Converter



Indexierung – Codestruktur

```
//create index writer
writer = new IndexWriter( FSDirectory.open(idxDDir),
    new StandardAnalyzer(Version.LUCENE_CURRENT, StopWords.getStopWords()),
    true,  IndexWriter.MaxFieldLength.UNLIMITED);

// index documents in docDir
indexDocs(docDir, writer);

// optimize
writer.optimize();
writer.close();
```


Indexierung – Codestruktur, 2

```
//foreach file in docDir
Converter c = null;
try {
    if ( filename.endsWith(".pdf") ){
        c = new PDFConverter();
    } else // .., html, htm, txt etc

//convert and decorate
Decorator d = new EbDecorator();
Document doc = d.decorate( c.convert( fileOrDir) );
writer.addDocument( doc );
```




Suche – Codestruktur

```
// process query statement
Query query = null;
Analyzer analyzer = new StandardAnalyzer( Version.LUCENE_CURRENT,
    StopWords.getStopWords() );
QueryParser parser = new QueryParser( Version.LUCENE_CURRENT,
    "contents", analyzer );
parser.setDefaultOperator( QueryParser.AND_OPERATOR );
try {
    query = parser.parse( qry );
} catch (ParseException e) {
    ...
}
```

Suche – Codestruktur, 2

```
// search index
IndexReader reader = null;
try {
    reader = IndexReader.open(FSDirectory.open(idxDir), true);
} catch (IOException e) {
    ...
}
Searcher searcher = new IndexSearcher(reader);
TopDocs topDocs = null;
try {
    topDocs = searcher.search( query, null, 100 );
} catch (IOException e) {
    ...
}
```

Literatur

-  Stefan Büttcher, Charles L.A. Clarke, Gordon V. Cormack
Information Retrieval: Implementing and Evaluating Search Engines
MIT Press, 2010
-  Christopher D. Manning, Prabhakar Raghavan, Hinrich Schütze
Introduction to Information Retrieval
Cambridge University Press, 2008
-  Michael McCandless, Erik Hatcher, Otis Gospodnetić
Lucene in Action 2nd Edition
Manning, 2010