SPRACHTECHNOLOGIE IN SUCHMASCHINEN
IR-GRUNDLAGEN – TERMBASIERTE SUCHE

MASTERSEMINAR SUCHMASCHINEN

COMPUTERLINGUISTIK

SOMMERSEMESTER 2025

STEFAN LANGER

STEFAN.LANGER@CIS.UNI-MUENCHEN.DE

"Big data"

IR ist eigentlich schon lange 'big data':

- Internet-Suchmaschinen indizieren Milliarden von Dokumenten
- Suchmaschinen-Software wie Solr und Elasticsearch kann ohne weiteres mehrere Millionen oder gar Milliarden von Dokumenten indizieren und verwalten, auch repliziert
- Ähnlich neueren NLP-Modellen basiert der Index (i.e. das Datenmodell) auf sehr vielen Daten
- Corporate-Suchmaschinen enthalten in ihren Indexen große Ausschnitte des Wissens eines Unternehmens und sind sehr häufig der umfassendste Informationsspeicher eines Unternehmens; sie machen die Information zugänglich

IR – Grundlagen - Termindex

Texte auf (Teil-)-Zeichenketten durchsuchen:

- Sequenziell mit Zeichenkettenabgleich (string matching) oder mit regulären Ausdrücken (z.B. Unix grep) – dies stößt bei größeren Textmengen schnell an die Grenzen des Machbaren

- Mithilfe eines termbasierten Indizes: Invertierte Dateien (inverted index)
  - Grundidee der Implementierung: Hash oder Trie von Dokumententermen (aber Indizes von großen Suchmaschinen sind noch zusätzlich optimiert) – Dense Vector

Invertierter Index

Zerlegung des Dokuments in Terme

Im einfachsten Fall: Zuordnung von Termen zu Dokument-Ids

Dokument 1

Schöne Männer gehören nach Cannes wie die Aschewolke an den isländischen Himmel

Dokument 2

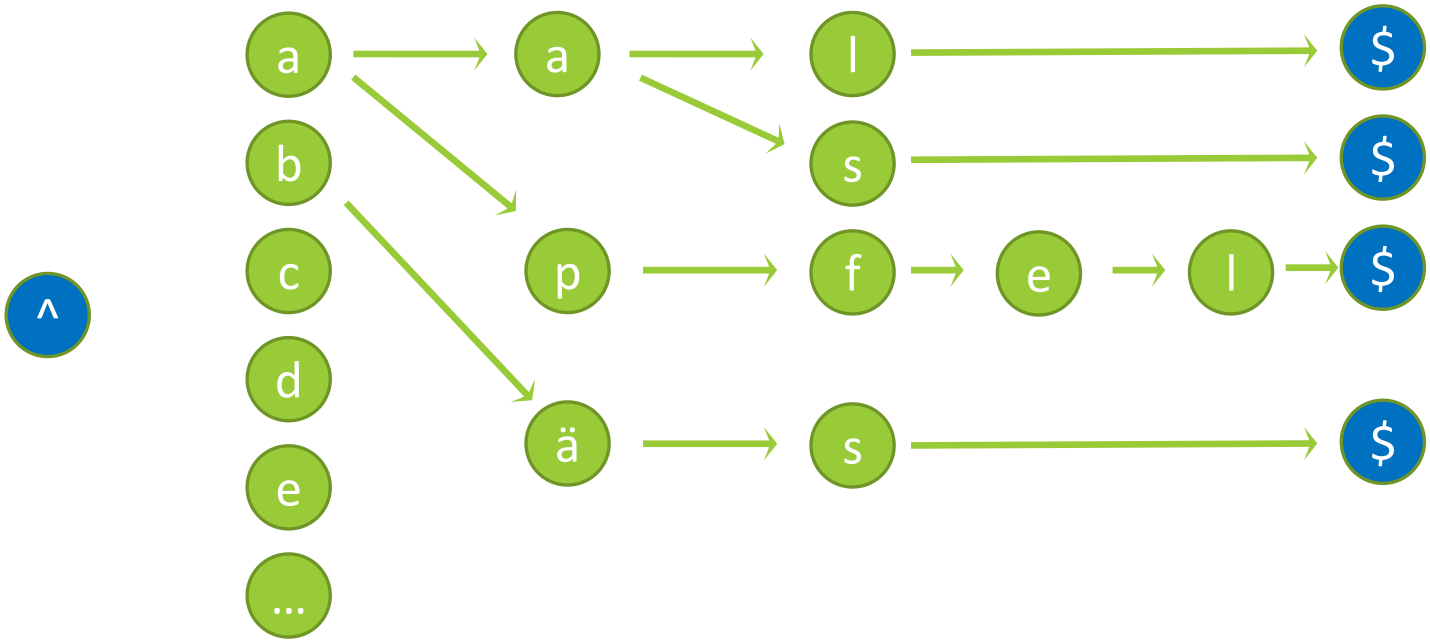
Getrocknetes Wasser, das vom Himmel fällt und Tiere ohne Flügel, die trotzdem fliegen können?

Index

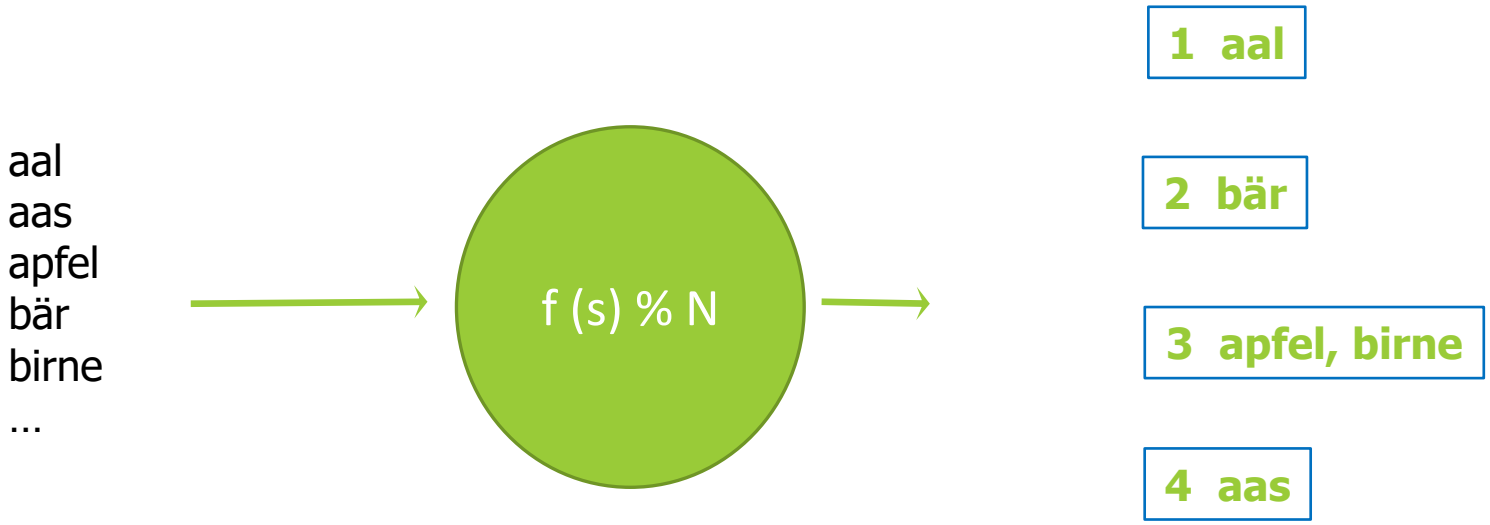
schöne.d1
männer.d1
....
himmel.d1/d2
wasser.d2
fliegen.d2

Datenstruktur: Trie

aal
aas
apfel
bär
...



Datenstruktur Hash



Hashing method
Hash function examples: md5, sha-1

Terminologie

Terminologie	Erklärung	Elasticsearch
Index	Index zum Speichern der Informationseinheiten (z.B. Tokens, Wortbigramme, Buchstaben N-Gramme)	Index, (Shard)
Feld	Teil eines Dokuments mit Label. Jedes indizierte Element gehört zu einem Feld	Field, property
Schema	Welche Felder existieren im Index, und welches Format haben sie (String, Integer, Boolean etc.)	Mapping, mapping type
Tokenisierung	Nach welchen Kriterien werden Token erzeugt	analyzer, tokenizer
Stemming	Linguistisch motivierte Trunkierung von Strings	stemming / stemmer
Facetten, Aggregationen	Unterteilung von Suchergebnissen aufgrund der Inhalte bestimmter Felder	aggregations
Filter	Teil einer Anfrage, der nicht zum Ranking verwendet wird	Filter

Beispieldokument


```
 {
    "title": "Japanische Wollmispel",
   "text": "Die Japanische Wollmispel ist eine Pflanzenart
aus der Gattung der Wollmispeln (Eriobotrya)",
   "url":
"https://de.wikipedia.org/wiki/Japanische_Wollmispel",
   "paragraph_number": "1",
   "chunk_id": 1,
   "language": "de",
   "filter": [
     "Kernobst",
     "Pflanzen"
   ],
   "keywords": [
     "Mispel",
     "Eriobotrya",
   ],
   "entry_modified": "2023-12-18 15:03:32",
   "entry_length": 593400,
   "processing_success": true,
   "processing_errors": false,
   "processing_max_tokens": 2000
 },
```


Resulting mapping (index):


```
"mappings": {
      "properties": {
        "chunk_id": { "type": "long"},
        "entry_length": { "type": "long"},
        "entry_modified": {
          "type": "date",
          "format": "yyyy/MM/dd||epoch_millis",
          "locale": ""},
        "filter": {
          "type": "text",
          "fields": {
            "keyword": {
              "type": "keyword",
              "ignore_above": 256}}},
        "processing_errors": { "type": "boolean"},
        "text": {
          "type": "text",
          "fields": {
            "keyword": {
              "type": "keyword",
              "ignore_above": 256
            .....
```

Boolsche Operatoren

UND (AND): beide Teilausdrücke müssen in einem Dokument auftreten

Das ist der Default bei allen mir bekannten Websuchmaschinen

Beispiel: *UND(“HAMLET”,”OPHELIA”)*

ODER (OR): einer der Teilausdrücke muss im Dokument auftreten

Beispiel: *ODER(“Streichholz”,”Zündholz”)*

NICHT (NOT): negiere den dahinterstehenden Ausdruck

Beispiel: *UND(“HAMLET”,NOT(”OPHELIA”))*

Kombinationen: z. B. ANDNOT = AND NOT

Termbasiertes Retrieval mit boolschen Ausdrücken

Verknüpfung von Suchtermen mit UND/ ODER/ NICHT

Verfahren:

1. Dokumenten-Ids werden für jede Teilquery ermittelt
2. Bilde Schnittmengen (UND) / Vereinigungsmengen (ODER) oder Differenzmengen (NICHT)

Effiziente Algorithmen verfügbar

Komplexität Schnittmengenbildung

Die Komplexität der Schnittmengenbildung (UND-Verknüpfung) für einen Index (Mengen nach Größe sortiert):

- Naiver Ansatz: Vergleiche alle Elemente aus Menge 1 mit allen aus Menge 2: O (n^2)

- Verbesserter naiver Ansatz: Sortiere die Elemente aus Menge 1 oder schreibe sie in einen Hash/Dictionary. Lookup von allen Elementen aus Menge 2 in Menge 1 (binäre Suche oder Hash-Lookup):O (n log n) (sortieren)

- Index-Ansatz: lege die Mengen bereits nach Dokument-ID sortiert ab. Arbeite die sortierten Listen nach Dokument-ID ab. Linear – O (N)

Suche von Termfolgen und Nähe im durchsuchten Dokument

Beispiele für Operatoren:

PHRASE(“Rot“,“Grün“)

NAH(Rot,Grün)

NAH(Rot,Grün,<ABSTAND>)

GEORDNETES_NAH(Rot,Grün)

Lucene: Abfrage von Tokens

text:dalloway → "text" : "Mrs. Dalloway would by the flowers …"

Finde alle Dokumente, die im Feld 'text' das Wort 'dalloway' enthalten. Wenn das Feld ein Textfeld ist, wird die Anfrage durch den Tokeniserer/Analyzer vorbehandelt, d.h. normalisiert.

Lucene: Boolsche Suche

Oder-Operator:

- text:"to the lighthouse" OR text:dalloway

Und-Operator:

- text:"Peter Walsh" AND text:Dalloway

Plus-Operator (der Term nach dem Plus muss auf jeden Fall auftreten):

- text:"Peter Walsh" +text:Dalloway

(logisch äquivalent zu allen Dokumenten für query *text:Dalloway*, aber anderes Ranking)

Nicht-Operator

- text:"Peter Walsh" AND NOT text:Dalloway

Boolean query syntax in elasticsearch / opensearch

Elasticsearch /opensearch **query_string** query (lucene format):

```
GET /_search
{ "query":
   { "query_string" :
     { "default_field" : "text", "query" : "this AND that OR thus" }
   }
 }
```

Equivalent elasticsearch **bool** query:

```
{ "query": {
   "bool": {
    "should": [
     {
      "bool": {
       "must": [
        { "match": { “text": "this" } },
        { "match": { “text": "that" } }
       ]
      }
     },
     {
      "match": { “text": "thus" }
     }]}}}
```

Lucene: Suche mit Platzhaltern

Wildcard ?: text:dall?way → dalloway, dallaway…

Wildcard *: text:lighthou*

Einschränkung: nicht in Phrasen, nicht als erstes Zeichen

Lucene: Unscharfe Suche

text:dalloway~

text:dalloway~1

Je höher der Wert nach der Tilde, desto unschärfer die Suche, Defaultwert ist 1, Maximalwert 2 – Levenshtein-Distanz

Einschränkungen:

- nicht in Phrasen (dort ist es Phrasen-Editierdistanz),
- nicht als erstes Zeichen

Ranking / Scoring

Boolsche Suche (ja/nein-Entscheidung) ist nicht ausreichend für eine Suche über sehr viele Dokumente

Beim Ranking geht es darum, für eine gegebene Anfrage zu ermitteln, wie gut ein Dokument zu ihr passt, um die Reihenfolge der Ergebnisse festzulegen.

Einige Parameter:

- Gewicht der Dokumententerme
  - Abhängig von der Termfrequenz im Dokument (TF)
  - Abhängig von der Termfrequenz über alle Dokumente (DF, IDF)
  - Abhängig vom Feld, in dem ein Term auftritt (Dokumentenstruktur)
- Gewicht der Anfrage-Terme in der Query
- Statischer Score des Dokuments / Felds

Rankingfunktionen

- (Okapi) BM25

$$\text{score}(D,Q) = \sum_{i=1}^{n} \text{IDF}(q_i) \cdot \frac{f(q_i,D) \cdot (k_1+1)}{f(q_i,D) + k_1 \cdot \left(1 - b + b \cdot \frac{|D|}{\text{avgdl}}\right)}$$

  - D: Dokument
  - q: query-Terme
  - k, b: Parameter (1.2, 0.75)

BM25F: + Felderinformation

Lucene scoring

Formel:

$$\text{score}(q,d) = \text{coord}(q,d) \cdot \sum\big(\text{tf}(t \text{ in } d) \cdot \text{idf}(t)^2 \cdot t.\text{getBoost}() \cdot \text{norm}(t,d) \big)$$

- **tf(t in d):** Wurzel aus der Termfrequenz in Dokument
- **idf(t):** Inversed Dokumentfrequenz - 1 + log (numDocs/(docFreq+1))
- **coord(q,d):** Bewertungsfaktor, der darauf basiert, wie viele der Anfrageterme im Dokument gefunden wurden. Standard = 1
- **t.getBoost():** Möglicher manueller Boost für term t in Anfrage q
  Bsp: *jakarta^4 apache*
- **norm(t,d):** umfasst einige(zur Indexierungszeit relevante) Boost- und Längenfaktoren:
  - Dokumenten-Boost – wird gesetzt, indem doc.setBoost() aufgerufen wird, wenn das Dokument dem Index hinzugefügt wird.
  - Feld-Boost – wird gesetzt, indem field.setBoost() aufgerufen wird, wenn das Feld einem Dokument hinzugefügt wird.
  - lengthNorm – wird beim Hinzufügen des Dokuments zum Index berechnet, abhängig von der Anzahl der Tokens dieses Felds im Dokument, sodass kürzere Felder stärker zur Bewertung beitragen.

Lucene scoring: Proximity

In der Formel fehlt:

- Termnähe (proximity) als Faktor

Mögliche Option:

~ "sloppy phrase query": text:"Leonce Lena"~100

- → max Distanz 100, sortiert nach Distanz

Lucene: Phrasensuche / Proximity

text:"to the lighthouse" → "text" : "Tomorrow, we'll go to the lighthouse"

Suche im Feld 'text' die Wortsequenz 'to the lighhouse' (wird tokenisiert/normalisiert)

Zwischen den Wörtern können im Dokument keine weiteren Token stehen.

Proximity:

text:"to the lighthouse"~1 → "text" : "Tomorrow, we'll go to the red lighthouse"

Zwischen zwei der Wörter kann ein Token stehen (~ ist die Phrasen-Editierdistanz)