Maschinelle Sprachverarbeitung

Übung

Aufgabe 2: Implementierung einer Volltextsuche

Mario Sänger

mario.saenger@informatik.hu-berlin.de

Übung 2

- Aufgabenstellung: Implementierung einer Volltextsuche
  - Ermittlung relevanter Dokumente für beliebige Suchanfragen
  - Abfragesprache:
    - Keyword-/Termanfragen
    - Phrasenanfragen
    - Konjunktionen von Phrase- und Termanfragen
  - Boolean Information Retrieval
    - Dokumente sind relevant oder nicht (1/0)
    - Kein Scoring / Ranking der gefunden Dokumente!

- Wiederverwendung / Adaption der Parser aus Übung 1!

Anfragesyntax

- Durchsuchbare Felder:
  - Titel des Artikels (*title*)
  - Abstract des Artikels (*abstract*)
  - Jahr der Veröffentlichung (*year*)
    - Verwendung des Journalveröffentlichungsangabe
    - Sowohl <PubDate> als auch <MedlineDate>
    - „2005-2006“ -> „2005“ + „2006“
  - MeSH-Terme (*mesh*)
    - Verwendung der Konzeptbezeichnung (descriptor)
    - Spezialisierungen (qualifier) diesmal ignorieren!

Anfragesyntax

- Keyword-/Termanfrage: <Feld>:<Token>
  - Beispiel: title:tumor
- Phrasenanfragen: <Feld>:"<Phrase>"
  - Beispiel: abstract:"breast cancer"
- Konjunktionen: <Anfrage> AND <Anfrage>
  - Beispiel: title:protein AND abstract:"network interaction"

Preprocessing

- Trennung von Wörtern / Token
  - Leerzeichen, Punkt, Komma, Doppelpunkt, Ausrufe- und Fragezeichen => (**.,:!?**)
  - Alle anderen Zeichen bleiben Teil der Token / Wörter!

- Beispiele:
  - "Then, EtHsp90 was detected"
    - "then", "ethsp90", "was", "detected"
  - "to the 5' end of nuclear"
    - "to", "the", "5'", "end", "of", "nuclear"
  - "a 39-nucleotide spliced leader RNA"
    - "a", "39-nucleotide", "spliced", "leader", "rna"

Preprocessing

- Konvertierung aller Token / Wörter in Kleinbuchstaben
  - Sowohl für Indexaufbau als auch Term- und Phrasenanfragen
  - Suche ist case-insensitive!

- Phrasensuche: Wörter müssen aufeinander folgen!
  - Dokument: “TATA-binding protein, LtTBP, co-fractionates”
    - “tata-binding”, “protein”, “lttbp”, “co-fractionates”
  - Dokument erfüllt “protein LtTBP”
  - Dokument erfüllt nicht “binding protein”!
  - Dokument erfüllt nicht “protein co-fractionates”!

Programmschnittstelle

- Bereitstellung eines Programmskeletons
  - Implementierung der Schnittstelle PubMedSearchEngine

- void buildIndex(Path pubmedDirectory)
  - List und parst alle xml-Dateien in *pubmedDirectory*
  - Baut die benötigten Indices auf

- Set<Integer> query(String query)
  - Parst die Anfrage und führt diese aus
  - Liefert die PubMed-Ids der Dokumente, welche die Anfrage erfüllen, zurück

Service-Implementierung

- Implementierung der Schnittstelle
  - Klassenskeleton in service.PubMedSearchEngineImpl

- Rahmenbedingungen:
  - Nur neue Klassen und Quelltext hinzufügen
    - Keinen Quelltext entfernen oder ändern!
  - Keine Veränderung der Methodensignaturen
    - Sichtbarkeit, Typen der Parameter, Rückgabewerte
  - Package-Struktur und Klassenname beibehalten
  - Verwendet nur den Default-Konstruktor!
    - Keine Parameter einführen!

Programmschnittstelle

- Bereitstellung von PubMedSearchApplication.java
  - Programm zur Evaluation Eurer Lösung

- Programmparameter:
  - Pfad zum PubMed-Verzeichnis
  - Eingabedatei mit Auflistung von Anfragen
  - Eingabedatei mit Auflistung der (Goldstandard-) Ergebnisse der Anfragen

- Programm führt die Anfragen aus und vergleicht die Ergebnisse mit dem Goldstandard

Anfragen- und Ergebnisdateien

- Anfragedatei
  - Eine Anfrage je Zeile

```
title:cancer AND title:therapy AND year:2015
mesh:"AMINO acid" AND year:2010
```

- Ergebnisdatei
  - Auflistung aller PubMed-Ids je Anfrage (komma-getrennt)

```
22069572,22069574,22081600
25408521,26091825,26100670,26108715,26117979
```

Abgabedetails

- Aufgabe muss mit Java oder einer Java-VM kompatiblen Sprache gelöst werden (z.B. Scala, Kotlin, ..)
- Programm lässt sich wie folgt starten:

```
java -jar uebung2-gruppeX.jar \
<PubMed-Verzeichnis> <anfragen> <ergebnisse>
```

- Programm ....
  - Indexiert alle PubMed-Artikel aus *PubMed-Verzeichnis*
  - Führt die Anfragen aus <anfragen> aus
  - Vergleicht die Ergebnisse mit <ergebnisse>

Abgabe

- Upload eines ZIP-Archivs *uebung2-gruppeX.zip*
  - Ausführbares JAR-Archiv und Quellcode des Programms
  - Feedback zur benötigten Zeit

- Testet das Programm vor der Abgabe auf gruenau!
  - Verwendet (auch) eigene Testanfragen
  - Prüft Plausibilität des Ergebnisses mit *grep*

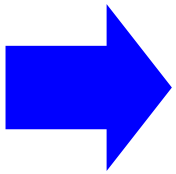
- Abgabe bis spätestens 22.11.2018, 23:59 Uhr über:
  - https://hu.berlin/ue_masprach1819_ass2

Lösungshinweise

- Verwendung von Inverted Files
  - Einfache und effektive Indexstruktur für Termsuche
  - Betrachtet Dokumente als „Bag of Words“

- Invertierte Sicht auf Dokumente
  - „Wörter kommen in Dokumenten vor“

	term1	term2	term3
Doc1	1	0	1
Doc2	1	0	0
Doc3	0	1	1
Doc4	1	0	0
Doc5	1	1	1
Doc6	1	1	0
Doc7	0	1	0
Doc8	0	1	0



	Doc1	Doc2	Doc3	Doc4	Doc5	Doc6	Doc7	Doc8
term1	1	1	0	1	1	1	0	0
term2	0	0	1	0	1	1	1	1
term3	1	0	1	0	1	0	0	0

Doc1:
Now is the time for all good men to come to the aid of their country

term	Doc
now	1
is	1
the	1
time	1
for	1
all	1
good	1
men	1
to	1
come	1
to	1
the	1
aid	1
of	1
their	1
country	1

Doc2:
It was a dark and stormy night in the country manor. The time was past midnight

term	Doc
it	2
was	2
a	2
dark	2
and	2
stormy	2
night	2
in	2
the	2
country	2
manor	2
the	2
time	2
was	2
past	2
midnight	2

Merge

term	Doc
a	2
aid	1
all	1
and	2
come	1
country	1,2
dark	2
for	1
good	1
in	2
is	1
it	2
manor	2
men	1
midnight	2
night	2
now	1
of	1
past	2
stormy	2
the	1,2
their	1
time	1,2
to	1,2
was	1,2

Lösungshinweise

- Effiziente Implementierung von Boolean-Anfragen möglich
  - Für jeden Term $term_i$, suche die Dokumentenliste doc_i welche $term_i$ enthalten
  - AND-Konjunktion als Schnittmenge: $term_i \wedge term_j : doc_i \cap doc_j$

- Beispiel:
  - title:time AND title:past AND title:the

  $= doc_{title:time} \cap doc_{title:past} \cap doc_{title:the}$
  $= \{1,2\} \cap \{2\} \cap \{1,2\}$
  $= \{2\}$

term	Doc
a	2
aid	1
all	1
and	2
come	1
country	1,2
dark	2
for	1
good	1
in	2
is	1
it	2
manor	2
men	1
midnight	2
night	2
now	1
of	1
past	2
stormy	2
the	1,2
their	1
time	1,2
to	1,2
was	1,2

Wettbewerb

- Die drei besten Teams bekommen 5/3/1 Punkte
  - Evaluation mittels eines Evaluationsdatensatzes!

- Herausforderungen:
  - Effiziente Implementierung von AND-Operator
    - Schnelle Schnittmengenbildung von der Teilergebnissen
  - Effiziente Implementierung der Phrasensuche
    - Wie indexiert man Phrasen effizient für die Suche?
    - Separater Index für Phrasen?
  - Effiziente Ausführung der Suche
    - Wie wählt man eine effiziente Strategie und Reihenfolge zur Evaluation der Teilanfragen?