

Kurs IR-Grundlagen, Praktischer Teil 5

Ziel

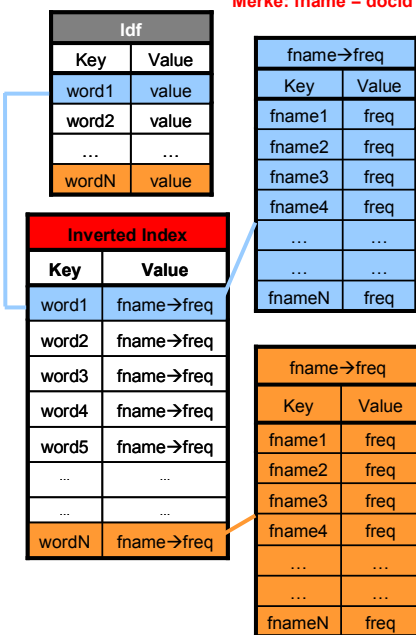
In diesem praktischen Teil geht es um die Umsetzung eines einfachen Information Retrieval Systems. Grundlage dazu bilden die Architekturfolien und der Pseudo-Code aus der Theorie. Es geht in diesem Praktikum weniger um programmiertechnische Aspekte (wie man Dateien einliest etc.), sondern um das Verständnis, wie IR-Architektur und Gewichtungsschemata in der Praxis umgesetzt werden. Dieses Verständnis ist der Schlüssel, Limitationen von IR-Systemen zu verstehen, respektive solche Systeme optimal aufzusetzen. Das in diesem praktischen Teil zu erarbeitende System verwendet das Gewichtungsschema tf.idf-Cosinus (Vektorraummodell), welches wir in Teil 2 der Theorie behandelt haben.

Architektur MiniRetrieve

Invertierter Index

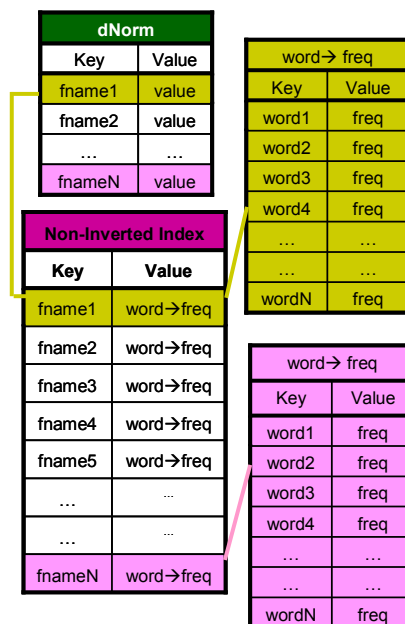
Quelle: Dokumente

Merke: fname = docid



Nicht-Invertierter Index

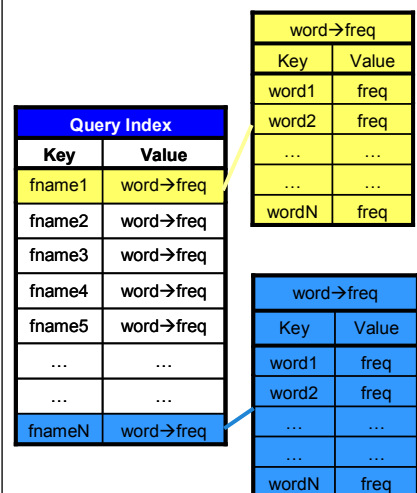
Quelle: Dokumente



Anfrageindex

Quelle: Anfrage

Merke: fname = queryid



PseudoCode - Teil 1.1

\$ -> Variable , @ -> Array , % -> Hash

indexing of documents and queries{

```
    foreach $doc in @documents { #create inverted and non-inverted index
        get @terms by tokenizing $doc;
        foreach $term in @terms {
            %invIndex{$term}{$doc} += 1 # increment frequency in inverted index
            %nonInvIndex{$doc}{$term} += 1 # increment frequency in non-inv index
        }
    }
    foreach $query in @queries{ #create index of queries
        get @terms by tokenizing $query;
        foreach $term in @terms {
            %queries{$query}{$term} += 1 # increment frequency
        }
    }
}
```

PseudoCode - Teil 1.2

\$ -> Variable , @ -> Array , % -> Hash

calculate all idfs and document normalizers{

```
    foreach $doc in %nonInvIndex{
        %dNorm{$doc} = 0;
        foreach $word in %nonInvIndex{$doc} {
            %idf{$word} = log ( ( 1 + totalNummerOfDocuments) / ( 1 +
documentFrequency) )
            $a = %nonInvIndex{$doc}{$word} * %idf{$word}
            %dNorm{$doc} += $a * $a;
        }
        %dNorm {$doc} = Math.sqrt(%dNorm{$doc})
    }
}
```

Architektur MiniRetrieve

accumulator

accu	
Key	Value
doc 1	acc. value
doc 2	acc. value
doc 3	acc. value
doc 4	acc. value
...	...
...	...
doc N	acc. value

Der Akkumulator summiert das Produkt von idf und Termhäufigkeit für jedes Word, das im entsprechenden Dokument vorkommt.

dNorm

dNorm	
Key	Value
fname1	value
fname2	value
fname3	value
fname4	value
fname 5	value
...	...
...	...
fnameN	value

Dokumentenorm „dNorm“ wird für alle Dokumente vorberechnet

idf

idf	
Key	Value
word1	value
word2	value
word3	value
word4	value
word5	value
...	...
...	...
wordN	value

Der idf (Inverse document frequency) wird für alle Wörter in allen Dokumenten vorberechnet, ggf. auch für Anfrageterm mit df=0.

Gewichtungsformel RSV

$$a_{i,j} := ff(\varphi_i, d_j) * idf(\varphi_i)$$

$$b_i := ff(\varphi_i, q) * idf(\varphi_i)$$

$$RSV(q, d_j) := \frac{\sum_{\varphi_i \in \Phi(q) \cap \Phi(d_j)} a_{i,j} * b_i}{\sqrt{\sum_{\varphi_i \in \Phi(d_j)} a_{i,j}^2} * \sqrt{\sum_{\varphi_i \in \Phi(q)} b_i^2}}$$

%accu

%dNorm

\$qNorm

Wobei:

- RSV = retrieval status value
- ff = feature frequency
- idf = inverse document frequency
- d = document
- q = query
- φ = term

PseudoCode - Teil 2.1

```
process queries{
  foreach $query in %queryIndex {
    $qNorm = 0
    create new %accu
    foreach $queryWord in %queryIndex{$query} { # process all query terms
      if(! %idf{$queryWord}{
        %idf{$queryWord} = log( 1 + totalNummerOfDocuments);
      }
      $b = %queries{$query}{$queryWord} * %idf{$queryWord}
      $qNorm += ($b * $b)
      if( %invIndex{$queryWord} is defined ) { # if query term occurs in collection
        foreach $document in @invindex{$queryWord} {
          # document scores are added up in accumulators. filename serves as document identifier
          $a = %invIndex{$queryWord}{$document} * $idf
          %accu{$document} += ( $a * $b );
        }
      }
    }
    #..... 2. Teil .....
  }
}
```

PseudoCode - Teil 2.2

```
process queries{
  foreach $query in %queryIndex {
    #..... 1. Teil .....
    $qNorm = Math.sqrt( $qNorm )
    foreach $document in %accu { # normalize length of vectors
      %accu{$document} /= (%dNorm{$document}) * $qNorm )
    }
    set @results = sort %accu by values # sort and return 1000 best results

    foreach $result in @results{
      print "$queryid Q0 $document $rank $accuValue"
    }
  }
}
```

Kollektion

Wir benutzen in diesem Praktikum die Cranfield Testkollektion. Diese besteht aus 1400 Dokumenten und 225 Anfragen. Sowohl die Filenamen der Dokumente (Ordner „documents“) als auch der Anfragen (Ordner „queries“) entsprechen jeweils der eindeutigen Identifikationsnummer des Dokumentes/Anfrage.

Aufgabe

Es ist Ihnen überlassen, in welcher Programmiersprache Sie Ihr IR-System auf die Beine stellen wollen. Hilfe erhalten Sie aber auf alle Fälle in Python, Perl und Java.

In Python und Perl lässt sich ein funktionierendes IR-System in weniger als 150 Zeilen Code umsetzen. In Java müssen Sie dazu im Allg. etwas mehr Code schreiben (je nach Ansatz).

Es ist ein Grundgerüst in Python vorgegebenen, das Ihnen viel Programmierarbeit abnimmt. Dieses Gerüst versucht die Vorgabe aus dem Pseudo-Code möglichst direkt umzusetzen. Die Verwendung dieses Gerüsts ist selbstverständlich freiwillig. Wenn Sie den Anspruch haben, ein IR-System von Grund auf neu zu programmieren, können direkt bei Aufgabe 2 beginnen.

Aufgabe 1

Nehmen Sie das vorgegebene Grundgerüst „miniretrieveGrundgeruest.py“ und implementieren Sie darauf basierend ein funktionsfähiges IR-System. Das Grundgerüst folgt der bereits bekannten Architektur und dem Pseudo-Code. D.h. die Datenstruktur der Indizes und ein Grossteil der Programmstruktur sind vorgegeben. Studieren Sie in einem ersten Schritt das vorgegebene Grundgerüst. Versuchen Sie dieses vollständig zu verstehen. Suchen Sie die Verbindungen zur Version in Pseudo-Code.

Sobald Sie das Gerüst verstanden haben, machen Sie sich daran, die fehlenden Teile zu implementieren.

Aufgabe 1.1 – Erstellung invertierten und nicht-invertierten Index

Implementieren Sie die Erstellung des invertierten und nicht-invertierten Indexes. Das Einlesen der Textdateien ist vorgegeben. Machen Sie sich Gedanken, wie Sie tokenisieren müssen (Hinweis: es ist sehr einfach, Stichwort: Regular Expressions). Der Text der Dateien steckt in der Variable "text". Benennen Sie die Datenstruktur für den invertierten Index "inindex" und die Datenstruktur für den nichtinvertierten Index "noninindex".

Aufgabe 1.2 – Berechnungen IDF und Normen

Implementieren Sie die Berechnung der Dokumentennormen und idfs

- Durchlaufen Sie alle Dokumente
- Durchlaufen Sie pro Dokument alle Terme und berechnen Sie den IDF und die Dokumentennorm
- Füllen Sie die berechneten Werte in die vorgegeben Dictionaries „dnorm“ und „idf“ ab

Variablen, die Sie dazu u.a. nutzen sollten:

- inindex -> hier können Sie "nachschiagen", in wievielen Dokumenten ein Begriff vorkommt
- noninindex -> hier finden Sie alle Terme eines Dokuments
- dnorm -> in dieses Dictionary speichern Sie die berechneten dnorm-Werte
- idf -> in dieses Dictionary speichern Sie die berechneten idf-Werte

- numdocs ->Anzal der Dokumente in der Kollektion

Aufgabe 1.3 – Normalisierung der Vektoren

Implementieren Sie die Normalisierung der Vektoren

- Lesen Sie die Norm der Dokumente aus
- Multiplizieren Sie den Wert des Akkumulators vor der Division noch mit Faktor 1000
- Dividieren Sie den Wert des Akkumulators durch das Produkt von Dokument- und Anfragenorm
- Speichern Sie den neu berechneten Wert wieder im Akkumulator

Variablen, die Sie dazu u.a. nutzen sollten:

- accu -> Dictionary mit den rohen Akkuwerten
- dnorm -> Dokumentennorm
- qnorm -> Anfragenorm mit dem aktuellen Normwert

Aufgabe 1.4 – Berechnungen des Akkumulators

Implementieren Sie die Berechnung der Gewichte noch fertig aus

Wir iterieren über alle Terme der Anfrage

- Berechnen Sie das Gewicht „b“ des Queryterms und dividieren Sie qnorm auf.

Wir überprüfen, ob der Anfrageterm im invertierten Index vorhanden ist. Falls ja:

- Berechnen Sie das Gewicht "a" des Queryterms im Dokument
- Berechnen Sie mit „a“ und „b“ den Akkumulatorenwert und speichern diesen entsprechend

Variablen, die Sie dazu u.a. nutzen sollten:

- invindex -> zum Nachschlagen des tf
- accu -> zum Speichern von a*b
- qnorm -> Anfragenorm, um den Normwert zu speichern

Aufgabe 2 - Für die ambitionierten Teilnehmer

Implementieren Sie gemäss Architektur und Pseudo-Code ihr Retrieval System von Grund auf neu. Falls Sie wollen, können Sie den prozeduralen Pseudo-Code je nach gewählter Programmiersprache in ein geeignetes objektorientiertes Design bringen. In erster Linie geht es aber darum, das System IR-technisch korrekt zu implementieren (keine Programmieraufgabe im eigentlichen Sinn).

Aufgabe 3 – Erweiterung IR-System (für Fanatiker)

Falls Sie Lust haben, Ihr kleines IR-System weiter auszubauen, überlegen Sie sich, wie Sie dieses noch optimieren können. Wie kann man Stoppporte/Stemming einbauen?

Kontrolle

Um Ihren Programmcode auf Richtigkeit zu verifizieren, vergleichen sie ihr Resultat mit dem Resultfile der Musterlösung „top10ResultsOutput.txt“, das im Trec-Format vorliegt. Durch Rundungsfehler kann Ihr Resultat ein klein wenig abweichen, sollte aber die gleiche Rangierung der Dokumente zurückgeben. Pro Anfrage finden Sie jeweils die ersten zehn Resultate.

Trec-Format:

- Spalte 1 : Anfragenummer
- Spalte 2 : Konstante „Q0“
- Spalte 3 : Rangierung
- Spalte 4 : Dokumentennummer
- Spalte 5 : RSV-Value
- Spalte 6 : Systemname