IR-Grundlagen - Worksheet 3

Dieses Worksheet behandelt ranked Retrieval, die Möglichkeiten einem Dokument mit einem Score zu versehen, TF-IDF und das Vektorraummodell. Ziel dieses Arbeitsblattes ist es für eine Query die besten 10 Ergebnisse zurückzugeben (mit Ranking).

Zuerst beschäftigen wir uns damit unser System mit den notwendigen Eigenschaften auszustatten. Wir möchten in der Lage sein die Termfrequenz eines Terms innerhalb eines Dokuments, sowie die Inverse Dokumentenfrequenz eines Terms innerhalb der Collection zu ermitteln. Diese beiden Werte sollen im TF.IDF-Wert kombiniert werden können und anhand dieses Wertes sollen die besten Matches für eine eingegebene Query ermittelt werden können.

1. Wie schon bei den ersten beiden Worksheets benötigen wir auch diesmal wieder eine verwendbare Repräsentation unseres Indexes. Wir verwenden die `Index` Klasse. Wie immer gilt: Legen sie, wenn nötig, zusätzliche Variablen und Methoden an. Implementieren sie die vorgegebenen Methoden:

    a. `public Index(ArrayList<Document> collection)`

    Indexieren Sie die einzelnen Dokumente des `collection` Objekts. Denken Sie dabei an das “Bag of Words”-Model und überlegen sie sich eine geeignete Datenstruktur dafür.

    b. `public int getTF(String term, int docID)`

    Diese Methode berechnet die Termfrequenz eines Terms in einem Dokument.

    c. `public int getIDF(String term)`

    Diese Methode berechnet die Inverse Dokumentenfrequenz eines Terms innerhalb der Collection

    d. `public int getTF_IDF_Score(String query, int docID)`

    Diese Methode berechnet einen TF.IDF Wert zwischen einer Query und einem Dokument.

2. Ist in diesem System Length-Normalization enthalten?
   Warum verwenden wir die Dokumenten-Frequenz und nicht die Collection-Frequenz?

3. Nun sind wir in der Lage die Collection als eine Matrix aus TF.IDF Werten zwischen Dokumenten und Termen zu repräsentieren. Das heißt jedem Dokument ist eine Spalte (oder ein Vektor) von TF.IDF Werten zugeordnet, und zwar genausoviele wie sich Terme in der Collection befinden (Anzahl N). Wir können also Dokumente (und Queries) als Vektoren in einem N-Dimensoinalen Raum betrachten. Das ermöglicht es uns die Ähnlichkeit eines Dokuments und einer Query anhand des Winkels α, genauer cos(α), zwischen ihnen zu bestimmen. Implementieren sie folgende Methoden

    a. `public ArrayList<Integer> normalize(ArrayList<Integer> vector)`

       Diese Methode längen-normalisiert einen Vector. Dies ist hilfreich da die Formel für cos(α) für längennormalisierte Vectoren einfacher ist.

    b. `public ArrayList<Integer> vectorSearch(String[] query, int k)`

       Diese Methode berechnet cos(α) für zwischen dieser Query und allen Dokumenten und gibt die ID's der Dokumente mit den k höchsten Werten zurück. Implementieren sie diese Suche nach dem lnc.ltc Schema (siehe Slides 41, 43-44). Eine Pseudocode-Implementierung befindet sich auf Seite 3

4. Warum verwenden wir cos(α) statt der Differenz zwischen dem Query-Vector und dem Dokumenten-Vector?

Hilfreiche Folien für dieses Arbeitsblatt:

13-21, 24-26, 28, 35-37, 41-44

Anhang

```
CosineScore(q)
 1  float Scores[N] = 0
 2  float Length[N]
 3  for each query term t
 4  do calculate w_{t,q} and fetch postings list for t
 5      for each pair(d, tf_{t,d}) in postings list
 6      do Scores[d] += w_{t,d} × w_{t,q}
 7  Read the array Length
 8  for each d
 9  do Scores[d] = Scores[d]/Length[d]
10  return Top K components of Scores[]
```