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. Jedes Dokument soll durch einen Vektor aus TF.IDF-Werten repräsentiert werden. Die gesamte Collection ist also als Matrix zwischen Termen und Dokumenten abgebildet und enthält einen TF.IDF für jeden Term und jedes Dokument.

    Überlegen sie sich eine geeignete Datenstruktur um diese Matrix abzuspeichern und implementieren sie diese.
    Nutzen sie für diese Aufgabe die Methoden aus Teilaufgabe b, c und d.

    b. `public int getTF(String term, Document doc)`

    Diese Methode berechnet die Termfrequenz eines Terms in einem Dokument.

    c. `public float getIDF(String term)`

    Diese Methode berechnet die Inverse Dokumentenfrequenz eines Terms innerhalb der Collection

    d. `public float getTF_IDF_Score(String term, Document doc)`

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

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

3. Um Ähnlichkeit zwischen zwei Dokumenten (oder zwischen einer Query und einem Dokument) zu bestimmen könnten wir nun die Differenz der beiden Vektoren bilden. Wobei liegt hier das Problem?

4. Stattdessen können wir diese Ähnklichkeit anhand des Winkels α, genauer cos(α), zwischen zwei Vektoren bestimmen. Die Formel hierfür lautet

$$cos(\vec{q},\vec{d}) = \frac{\vec{q}\bullet\vec{d}}{|\vec{q}||\vec{d}|} = \frac{\vec{q}}{|\vec{q}|} \bullet \frac{\vec{d}}{|\vec{d}|}$$

   Wir bilden also das Produkt aus Vektoren geteilt durch ihre Länge. Ein Vektor geteilt durch seine Länge hat die Länge 1. Wenn wir also vorher alle Vektoren auf die Länge 1 bringen vereinfacht sich diese Formel zu

$$cos(\vec{q}, \vec{d}) = \vec{q} \bullet \vec{d} = \sum_{i=1}^{N} q_i d_i$$

   Implementieren sie folgende Methoden

```
public float[] normalize(int[] vector)
```

   Diese Methode Längen-normalisiert einen Vektor und bringt auf die Länge 1. Verändern sie anschließend ihre Matrix damit sie Längen-normalisierte TF.IDF Werte speichert.

5. Im Vektorraummodell wird eine Query genau wie ein Dokument behandelt. Um also mit der Query umgehen zu können erstellen Sie zuerst ein Objekt der Document Klasse für die Query.

   Jetzt können wir endlich die Suchfunktion implementieren

```
ArrayList<Integer> vectorSearch(Document query, int k)
```

   Diese Methode berechnet cos(α) zwischen dieser Query und allen Dokumenten und gibt die ID's der Dokumente mit den k höchsten Werten zurück.

   Die Vorgehensweise ist dabei:

   a. Nehmen sie die Terme der Query, berechnen sie den TF.IDF Wert zwischen diesem Term und der Query (Dem Document-Objekt dass Sie am Anfang dieser Aufgabe erstellt haben) und speichern Sie diese in einem Array.

b. Errechnen sie einen Wert für jedes Dokument indem sie für jeden Term der Query den TF.IDF Wert zwischen einem Term und der Query mit dem entsprechenden zwischen dem Term und diesem Dokument multiplizieren und die Ergebnisse addieren.
c. Wiederholen sie das für alle Dokumente. und speichern sie alle Ergebnisse in einem Array
d. Geben sie die k höchsten Einträge dieses Arrays zurück.

Hilfreiche Folien für dieses Arbeitsblatt:

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

```
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[]
```