CAB431 Tutorial (Week 5): TF*IDF Calculation

****

TF-IDF is the product of two statistics, term frequency and inverse document frequency, to measure **the weight of a term's appearance in a document**. Various ways for determining the exact values of both statistics exist.

Term Frequency:

The TF term in TF*IDF can be the raw term frequency $f_{d,t}$. However, a term that occurs 10 times is not generally 10 times as important as a term that occurs once. Therefore, an alternative formulation of the TF component is:

$$1 + \log(f_{t,d})$$

Inverse Document Frequency:

If *N* is the number of documents in the corpus, and df_t is the number of documents that contain term *t*. Then the IDF of *t* is defined as:

$$\mathrm{idf}_t = \log \frac{N}{\mathrm{df}_t}$$

For example, suppose there are 10 documents and a word "tutorial" appears in three of them. Then, mathematically, its Inverse-Document Frequency, IDF = log(10/3).

Smoothing and Document-Length-Normalized version:

$$\mathrm{TF{\cdot}IDF}_{t,d} = \frac{(1+\log(f_{t,d}))\cdot\log\frac{N}{df_t}}{\sqrt{\sum_{i=1}^{T}\left[(1+\log(f_{i,d}))\cdot\log\frac{N}{df_i}\right]^2}}$$

Note: there are lots of variant formulations and combinations! Whatever formulation is used, the unit-length-normalized TF*IDF scores are the precomputed and stored, so that similarity comparison is just a dot product.

TASK 1: Calculate Document-Frequency of each term and store them in a term:df dictionary (in Python, C#) / HashMap (in Java). Call the created method then display a list of TERM: DF for whole RCV1v2 document collection, and save the output into a text file (file name is “your full name_wk5_t1.txt”).

Example of output

```
There are 10 documents in this data set and contains 816
terms.
share: 5
market: 4
compani: 4
three: 4
royal: 4
public: 3
strong: 3
busi: 3
hold: 3
sector: 2
higher: 2
follow: 2
signific: 1
jihad: 1
katyusha: 1
morel: 1
westwood: 1
settlement: 1
hole: 1
privat: 1
andrea: 1
depend: 1
aug: 1
articl: 1
deviat: 1
swap: 1
```

TASK 2: Calculate TF*IDF value of every term in a BowDocument in a given BowDocument collection. Create a method in your processor class, and call calculateTfIdf method to generate a term:tfidf HashMap for each document, then fill a HashMap of docId: tfidfHashMap, **print out top 20 terms** (with its value of tfidf) for each document if it has more than 20 terms, and save the output into a text file (file name is "your full name_wk5_t2.txt").

Example of output

```
Document 741299 contains 96 terms.
german: 0.168450
soper: 0.168450
victori: 0.168450
race: 0.168450
second: 0.168450
tyre: 0.168450
struggl: 0.168450
belgian: 0.168450
lehto: 0.168450
schneider: 0.168450
lead: 0.145938
car: 0.145938
dalma: 0.099489
four: 0.099489
austrian: 0.099489
fifth: 0.099489
han: 0.099489
swap: 0.099489
els: 0.099489
merced: 0.099489
```

Appendix

TASK 1 Specification

Create a method in your processor class:

```
/**
   *
    * @param docCollection - a Collection of BowDocument, i.e. HashMap<String,
    *    BowDocument> of docId:aBowDocument
   * @return a HashMap<String, Integer> of term:df
   */
  private HashMap<String, Integer> calculateDF(HashMap<String, BowDocument>
docCollection) {
        //your code here
  }
```

*Note: you can discard the parameter above, instead, directly use the class variable of docCollection which have been filled when parse each document in the document collection (see week 3 tutorial, task 1).

- Load all files in the given document set, build up a BowDocument collection as a HashMap docCollection.
- Call above created method then display a list of TERM: DF for whole document set, it'd better to output a sorted version.

TASK 2 specification

Create a method in your processor class:

```
/**
   * @param aDoc - a BowDocument
   * @param noDocs - number of documents in given document set
   * @param dfs - a HashMap of term:df
   * @return a HashMap of term:tfidf for every term in a document
   */
  private HashMap<String, Double> calculateTfIdf(BowDocument aDoc, int noDocs,
HashMap<String, Integer> dfs) {
   //your code here, may use Math.log10(), Math.pow(), Math.sqrt() … for your calculation
}
```

- Call **calculateTfIdf** method to generate a term:tfidf HashMap for each document, then fill a HashMap of docId: tfidfHashMap.