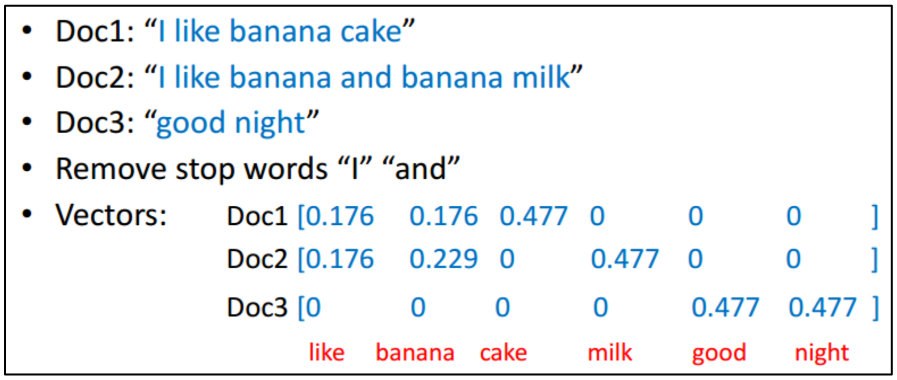
CS5344 Lab 2

AY2022/2023 Semester 2

Write a Spark program that finds the top-10 most relevant documents given a query comprising of set of keywords and identify the most relevant sentence in these documents to the query. This is an individual lab assignment.

A *document* can be modelled as a vector of words (or terms). Each entry in the vector is a **TF-IDF** value that reflects how important a word is to a document in a collection, computed as **TF-IDF = (1 + log (TF)) * log (N/DF)** where N is total number of documents, TF is the count of the word in a document, and DF is the count of documents having the word. Figure 1 shows a simple example.



- Doc1: “I like banana cake”
- Doc2: “I like banana and banana milk”
- Doc3: “good night”
- Remove stop words “I” “and”
- Vectors:

Doc1	[0.176	0.176	0.477	0	0	0]
Doc2	[0.176	0.229	0	0.477	0	0]
Doc3	[0	0	0	0	0.477	0.477]
	like	banana	cake	milk	good	night

Figure 1. Example of representing documents as vectors.

A *query* can also be represented as a vector where each entry represents a word with a value 1 if the word is in the query, and 0 otherwise. We can compute a *relevance score* for each document *d* to a query *q* based on the based on the cosine similarity of their corresponding vectors V_1 and V_2 and rank the documents with respect to a query:

$$\textbf{relevance (q, d)} = cosine(\overrightarrow{V_1}, \overrightarrow{V_2}) = \frac{\overrightarrow{V_1} \cdot \overrightarrow{V_2}}{||\overrightarrow{V_1}|| \times ||\overrightarrow{V_2}||}$$

Algorithm.

Step 1. Compute term frequency (TF) of every word in a document.
This is similar to the Word Count program in Lab 1.

Step 2. Compute TF-IDF of every word w.r.t a document.
Use key-value pair RDD and the groupByKey() or reduceByKey() API for this step.

Step 3. Compute normalized TF-IDF of every word w.r.t. a document.

If the TF-IDF value of *word1* in *doc1* is t_1 and the sum of squares of the TF-IDF of all the words in *doc1* is S, then the normalized TF-IDF value of *word1* is $\frac{t_1}{\sqrt{S}}$.

Step 4. Compute the relevance of each document w.r.t a query.

Step 5. Sort and get top-10 documents.

Step 6. For each of the top-10 document, compute the relevance of each sentence w.r.t the query. A sentence is delimited by a full-stop or you could set the rules about how to split the sentences.

Step 7. Output the most relevant sentence in each of the top-10 document.

Input: (a) set of documents (in "datafiles" folder),
(b) set of keywords for a query (in query.txt),
(c) stopwords to remove (in *stopwords.txt*).

Output: One line per document in the following format:
<docID> <document relevance score> <relevant sentence> <sentence relevance score>

The output should be sorted in descending order of the relevance of the documents to the query.

Deliverables: Zip your executable **Spark program with documentation in the code**, **the output files**, and upload it to the Lab2 folder in Luminus. The zipped folder should be named as Student ID_Lab2.

Important Notes:

- Your code should be executable either on the virtual machine configuration given in Lab 1 or on stand-alone Spark configuration.
- Specific the python version used in your program.
- For data preprocessing, all words should be transformed to lowercase.
- The logarithm for TFIDF is log10.
- You could use tokenizers like nltk to process the data.
- The scope of TF-IDF is different in step 2 and step 6. When you calculate the sentence relevance, the scope is the document that the sentence is in, i.e., N is the count of sentences in the document and DF is the count of sentences that contain the word in the document.