CSI4107, Winter 2018

Assignment 1
Due Saturday February 17, 10:00 PM

Microblog information retrieval system [50 points]

Note: **The assignment should be done in groups of two.**

You will implement an Information Retrieval (IR) system based for a collection of documents (Twitter messages), available here. The collection is a subset of the data used in the TREC 2011 Microblog retrieval task. Each line contains the id of the Twitter message and the actual message.

You will submit the results of your system on a set of 49 test queries (available here).
An example topic could be in the following format:

```
<top>
<num> Number: MB001 </num>
<title> BBC World Service staff cuts </title>
<querytime> Tue Feb 08 12:30:27 +0000 2011 </querytime>
<querytweettime> 34952194402811904 </querytweettime>
</top>
```

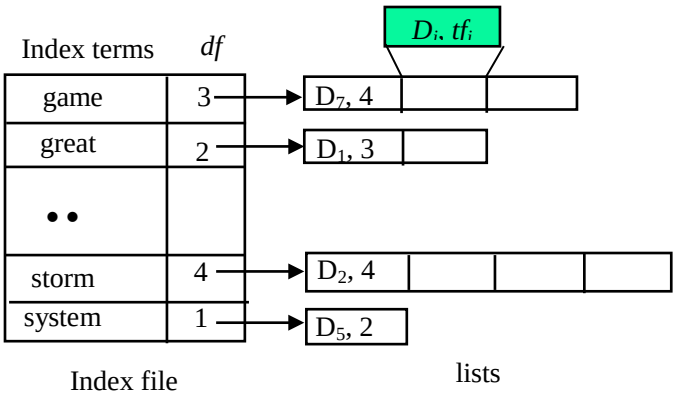
Implement your IR system using the steps pointed below as guidelines. Alternatively, you can use any existing IR system available, and adapt it to accomplish the same steps, on the provided Twitter collection.

Step1. [5 points] **Preprocessing**: Implement preprocessing functions for tokenization and stopword removal. The index terms will be all the words left after filtering out punctuation tokens, numbers, stopwords, etc. Optionally, you can use the Porter stemmer to stem the index words.

- Input: Documents that are read one by one from the collection
- Output: Tokens to be added to the index (vocabulary)

Step 2. [5 points] **Indexing**: Build an inverted index, with an entry for each word in the vocabulary. You can use any appropriate data structure (hash table, linked lists, Access database, etc.). An example of possible index is presented below.

- Input: Tokens obtained from the preprocessing module
- Output: An inverted index for fast access



For weighting, you can use the tf-idf weighting scheme (w_ij = tf_ij x idf_i). For each query, your system will produce a ranked list of documents, starting with the most similar to the query and ending with the least similar. For the query terms, you can use a modified tf-idf weighting scheme $w_{iq} = (0.5 + 0.5\, tf_{iq}) \cdot idf_i$

Step 3. [5 points] **Retrieval and Ranking**: Use the inverted index (from step 2) to find the limited set of documents that contain at least one of the query words. Compute the similarity scores between a query and each document (using cosine or other method).

- Input: One query and the Inverted Index (from Step2)
- Output: Similarity values between the query and each of the documents. Rank the documents in decreasing order of similarity scores.

Step 4. [10 points] **Results file:** Run your system on the set of test queries. Include the output in your submission as a file named Results.
The file should have the following TREC format. Include the top-1000 results for each query (the queries should be ordered in ascending order):

```
topic_id Q0 docno rank score tag
```

where: topic_id is the topic/query number, Q0 is an unused field (the literal 'Q0'), docno is the tweet id, rank is the rank assigned by your system to the segment (1 is the highest rank), score is the computed degree of match between the segment and the topic, and tag is a unique identifier you chose for this run (same for every topic). Example:

```
MB01 Q0 3857291841983309 1 0.999 myRun
MB01 Q0 3857291841983302 2 0.878 myRun
MB01 Q0 3857291841983301 3 0.314 myRun
...
MB01 Q0 3857291214283390 1000 0.000001 myRun
...
```

Step 5. [10 points] **Evaluation:**
For evaluation, you can use the trec_eval script. The main evaluation measures will be MAP (mean average precision) and P@10 (precision in the first 10 documents retrieved). Compare your results with the expected results, from the relevance feedback file, available here, in the following format:

```
Q_no  0     tweet_id          Relevance
```

where `Q_no` is the query number and Relevance can be 0 (not relevant) or 1 (relevant).
For example:

```
1     0     30986508492087297     1
1     0     30951976883716096     0
1     0     30945386776498176     0
…
```

Resources: Here is a list of stopwords that you can use in Step 1 (or any other list you prefer). In Step1 you can use the Java regular expressions library to help with the filtering. If you want to use stemming, you can use the Porter stemmer.
You can use any other resources from the Internet, as long as you explain in your report how you used them (compilation, installation, adaptation, etc.).
One resource you can explore is the package ir.vsr. Some documentation on ir.vsr is available here.
Other IR systems are available on the Internet (Lucene, Terrier, Lemur, Smart, etc.).

Note: while you implement your system, use a small subset of the text collection if you run into computational problems.

Submission instructions:

- write a README file (plain text or Word format) **[15 points for this report]** including:
  * your names and student numbers. Specify how the tasks were divided between the team members
  * a detailed note about the functionality of your programs
  * complete instructions on how to run them
  * Explain the algorithms, data structures, and optimizations that you used in each of the three steps. How big was the vocabulary? Include a sample of 100 tokens from your vocabulary. Include the first 10 answers to queries 1 and 25. Discuss your final results.
- include the file named Results with the results for all the 49 test queries, in the required format.
- make sure all your programs run correctly in Java.
- submit your assignment, including programs, README file, and Results file, as a zip file through Blackboard Learn.
- don't include the initial text collection or any external tools.

Have fun!!!