Homework 7: Information Retrieval [starter code]

Due: Tuesday May 8 at 11:59pm Pacific Daylight Time

In this assignment you will be improving upon a rather poorly-made information retrieval system. You will build an inverted index to quickly retrieve documents that match queries and then make it even better by using term-frequency inverse-document-frequency weighting and cosine similarity to compare queries to your data set. Your IR system will be evaluated for accuracy on the correct documents retrieved for different queries and the correctly computed tf-idf values and cosine similarities.

Data

You will be using your IR system to find relevant documents among a collection of 60 short stories by the famed Rider Haggard. The training data is located in the data/ directory under the subdirectory `RiderHaggard/`. Within this directory you will see yet another directory `raw/`. This contains the raw text files of 60 different short stories written by Rider Haggard. You will also see in the `data/` directory the files `queries.txt` and `solutions.txt` (and `solutions_java.txt` for a slightly more Java-friendly version). We have provided these to you as a set of development queries and their expected answers to use as you begin implementing your IR system.

Your Assignment

Improve upon the IR system given. This involves implementing:

- **Inverted Index:** Implement an inverted index - a mapping from words to the documents in which they occur.
- **Boolean Retrieval:** Implement a Boolean retrieval system, in which you return the list of documents that contain all words in a query (yes, we only support conjunctions...)
- **TF-IDF:** Compute and store the term-frequency inverse-document- frequency value for every word-document co-occurrence:
  $w_{t,d} = (1 + \log_{10} \mathrm{tf}_{t,d}) \times \log_{10}(N/\mathrm{df}_t)$
- **Cosine Similarity:** Implement cosine similarity in order to improve upon the ranked retrieval system, which currently retrieves documents based upon the Jaccard coefficient between the query and each document. The lecture on *Calculating TF-IDF Cosine Scores* will be helpful for this task. Specifically, you should look at slide 5 of the corresponding lecture slides. **Also** note that when computing $w_{t,q}$ (i.e. the weight for the word w in the query) do *not* include the idf term. That is, $w_{t,q} = 1 + \log_{10} \mathrm{tf}_{t,q}$.
  Note that the reference solution uses *ltc.lnn* weighting for computing cosine scores (see

slides 2 and 3 from the lecture *Calculating TF-IDF Cosine Scores* for reference).

To improve upon the information retrieval system, you must implement and/or improve upon the following functions (described in terms of the Java code, but they function identically in the Python version - although the names have underscores rather than being in camelCase):

- `void index()` This is where you will build the inverted index. The documents will have already been read in for you at this point, so you will want to look at some of the instance variables in the class (analogous data structures are available in the Python code):
  - `List<String> titles`
  - `ArrayList<ArrayList<String>> documents`
  - `ArrayList<String> vocab`
- `ArrayList<Integer> getPosting(String word)` This function returns a list of integers (document IDs) that identifies the documents in which the word is found. This is basically just an API into your inverted index, but you must implement it in order to be evaluated fully.
- `ArrayList<Integer> booleanRetrieve(ArrayList<String>query)` This function performs Boolean retrieval, returning a list of document IDs corresponding to the documents in which all the words in `query` occur.
- `void computeTFIDF()` This function computes and stores the tf-idf values for words and documents. For this you will probably want to be aware of the class variables `vocab` and `documents` which hold, respectively, the list of all unique words and the list of documents, where each document is a list of words.
- **double getTFIDF(String word, int doc):** You must implement this function to return the tf-idf weight for a particular word and document ID.
- `PriorityQueue<Integer> rankRetrieve(ArrayList<String> query)` This function returns a priority queue of the top ranked documents for a given query. Right now it ranks documents according to their Jaccard similarity with the query, but you will replace this method of ranking with a ranking using the cosine similarity between the documents and query.

We suggest you work on them in that order, because some of them build on each other. It also gets a bit more complex as you work down this list.

Evaluation

Your IR system will be evaluated on a development set of queries as well as a held-out set of queries. The queries are encoded in the file **queries.txt** and are:

- separation of church and state
- priestess ritual sacrifice
- demon versus man
- african marriage queen
- zulu king

We test your system based on the four parts mentioned above: the inverted index, boolean retrieval, computing the correct tf-idf values, and implementing cosine similarity using the tf-idf values.

Running the code

For Java, execute

```
$ cd java
$ mkdir classes
$ javac -d classes *.java
$ java -cp classes IRSystem
```

This will run you IR system and test it against the development set of queries. If you want to run your IR system on other queries, you can do so by replacing the last line above with

```
$ java -cp classes IRSystem "My very own query"
```

where **My very own query** is your query.
Similarly, for python, execute

```
$ cd python
$ python IRSystem.py
```

And, again, you can run your own queries with

```
$ cd python
$ python IRSystem.py "My very own query"
```

Note that the first time you run this, it will create a directory named `stemmed/` in `../data/RiderHaggard/`. This is meant to be a simple cache for the raw text documents. Later runs will be much faster after the first run. *However*, this means that if something happens during this first run and it does not get through processing all the documents, you may be left with an incomplete set of documents in `../data/RiderHaggard/stemmed/`. If this happens, simply remove the `stemmed/` directory and re-run!

Submitting Your Solution

We have provided a submission script/program which will run your program and test it in the same way that just running your program does. We run half of the tests with development queries (exactly the same ones we give you), and half of the tests with held out queries. When you run the submission program, it will programmatically call your script and then prompt you for your username and a password. The password can be found at the top of the Assignments page on this website. It will also ask you about which parts of the assignment you wish to submit. For this assignment your system will be evaluated in eight parts: each of the four parts of the assignment (inverted index, boolean retrieval, tf-idf, and ranked retrieval with cosine similarity) will be evaluated on both the development queries and a held-out set of queries.
Here are the basic shell commands which you will need to execute for Java:

```
$ cd java
$ mkdir classes
$ javac -d classes *.java
```

```
$ java -cp classes Submit
```

and for Python

```
$ cd python
$ python submit.py
```

It is important that these scripts be executed from either the java or python directories so that they can easily import the code you've written and so that the relative path `../data/` correctly points to the training and development data (it is hard-coded into the submission scripts).

Bugs and notes:

The following are some bugs that will be included in the next version (v2) if there are enough to make it necessary to post a new version:

- The comments for `boolean_retrieve` and `booleanRetrieve` are wrong (and misleading). They should say

  ```
  Given a query in the form of a list of *stemmed* words, this returns
  the list of documents in which *all* of those words occur (ie an AND
  query).
  ```

  Notice that the words are **stemmed** when they come in, *not* unstemmed.

Notes

- Some of the filenames have colons in them. This may cause issues with some systems. On Windows, for example, you may want to try a utility other than WinZip (e.g. TugZip) to extract, or use 7-Zip and edit the file names before exporting them so that they export correctly. In the end, you should have 60 `.txt` files in `../data/RiderHaggard/raw/`.