CS 1655: Secure Data Management and Web Applications (Fall 2014)

Department of Computer Science, University of Pittsburgh

Assignment #1: Information Retrieval

Released: September 10th, 2014 **Due:** 11:59pm, Wednesday, September 24th, 2014

Goal

Gain familiarity with information retrieval and relevance ranking.

Description

You are asked to built a simple information retrieval system, with three specific components: (a) the preprocessor, (b) the indexer, and (c) the ranker.

You are asked to built these systems in Java[1] and make sure they compile and run correctly on `unixs.cis.pitt.edu`.

Your programs should adhere strictly to the specifications for command-line arguments and to the input/output formats (for files and for the screen), as explained below. Small sample files that you can use for testing will be posted on the class web site within 24 hours.

(A) The Preprocessor

The preprocessor should perform the following operations in order:

- **Read file** Read in memory the file specified at command line (see below on how to execute the program for the specification of command-line arguments). You have the option of either reading the entire file in memory or read a line at a time (and then perform the remaining operations, in a pipeline fashion). Note that you are processing only one file at a time. You would need a simple shell script to process multiple files, one after the other.

- **Punctuation and special character removal** All special characters, including those used for punctuation (for example: . , : ; ! ? -), should be eliminated. To do this, you should simply convert each special character to a single space character.

- **Case elimination** All characters in the text file should be converted to lower-case. This would make sure for example that "Article" (that appears at the beginning of a sentence) and "article" (that does not appear at the beginning of a sentence) are both considered the same word, as is proper to do.

- **Stop-word elimination** All stop-words (for example: the, a, about, etc) should be eliminated. To do this you should replace a word with an empty string. For example, `|about|` (i.e., the stop-word about, surrounded by `|` symbols) should be replaced by `||` (i.e., an empty string).

  A complete list of stop-words that you are expected to use is provided along with this assignment at `http://db.cs.pitt.edu/courses/cs1655/fall2014/assign/hw1.stopwords.txt`

[1] You can use a different programming language, but you should first get permission from the instructor, by sending an email to cs1655-staff@cs.pitt.edu and specifying which language you want to use and why.

- **Stemming** You should perform simple linguistic normalization by reducing the variant forms of a word to a common form, for example: *connection, connections, connective, connected,* and *connecting* should all be converted to *connect*.

  Although there are many complicated rules for stemming, we will keep things simple for this assignment. In particular you are asked to only strip out the following suffixes of words in the input document:

  - -ion
  - -ions
  - -ive
  - -ed
  - -ing
  - -ly
  - -s
  - -es

  For example, `|buses|` and `|busing` should both be converted to `|bus|`.

- **Counting & Writing to disk** After the previous preprocessing steps have been completed, your program should compute the frequency of each unique word in the input file and write this information back to disk in the following format:

  ```
  word1, count1
  word2, count2
  ...
  ```

  The words need not be in any particular order.

How to execute:
You should name your program for this task `preproc` and you should run it as follows:

```
java preproc filename
```

You program should then expect to find file `filename.txt` (notice the addition of the .txt extension) in the current directory (or exit with an error message if not found), do the preprocessing as specified above, and then write the word counts into a new file named `filename.counts`

We will typically have numbers identifying the different documents, e.g., 101.txt, 102.txt, 103.txt, etc.

(B) The Indexer

The indexer should read **all** the count files in the current directory that were generated by the preprocessing phase (i.e., files that have a .counts extension), build an inverted index for the entire collection of documents, and save the index to disk, as filename `inverted.index`

The inverted index should contain all the keywords that appear in the collection of documents, and for each keyword there should be a listing of which files it appears in (regardless if it appears only once or multiple times in each file). By definition, for each keyword there should be at least one document that contains it.

The inverted index file should be in a comma-separated format as follows:

```
word1, filenameX, filenameY, filenameZ
word2, filenameK, filenameL
word3, filenameM
...
```

Note: For simplicity the filenames should not have their .txt extension (e.g., use 101, instead of 101.txt).

How to execute:
You should name your program for this task `indexer` and you should run it as follows:

```
java indexer
```

You program should then expect to find multiple files with a `.counts` extension in the current directory (or exit with an error message if none is found), execute the index creation process as specified above, and then write the index into a new file named `inverted.index`, using the format specified above.

(C) The Ranker

Given up to three keywords from the user, the ranker program is expected to read in the inverted index created by the indexer program and compute a ranked list of relevant documents which are output, together with the relevance scores.

The relevance score of keyword i to document j is expected to be computed with the following formula:

$$w(i,j) = \begin{cases} (1 + log_2\ f_{i,j}) \times log_2\ \frac{N}{n_i} & \text{if } f_{i,j} > 0 \\ 0 & \text{otherwise} \end{cases}$$

where $f_{i,j}$ is the frequency of keyword i in document j (i.e., what is stored in the .counts file), N is the total number of documents, and n_i is the number of documents that contain keyword i.

The relevance score for a set of keywords (x, y, z) to document j is simply computed by adding up their individual scores $w(z,j) + w(y,j) + w(z,j)$.

Note that you must perform on the user-submitted keywords the preprocessing steps described earlier, for the `preproc` program.

How to execute:
You should name your program for this task `ranker` and you should run it as follows:

```
java ranker keyword1 keyword2 keyword3
```

Note that there can be one, two, or three keywords specified by the user.

Your program should expect to find in the current directory: (a) a file called `inverted.index` and (b) the corresponding files with `.counts` extensions for all files mentioned in the inverted index. The program should quit with an appropriate error message if there is any file missing.

Your program should then use these files to evaluate the relevant documents for the specified keyword(s), compute the appropriate scores, and display the list of documents in the following comma-separated output format.

Output format

```
1, filename1, relevance_score1
2, filename2, relevance_score2
3, filename3, relevance_score3
...
```

where the first column is the rank of the result, the second column is the filename (without a .txt extension), and the third column is the total relevance score, which is computed by summing up the scores for the user-specified keywords for that file. The list is ordered, where the highest score is first, the second highest score is second, etc.

What to submit

Submit all the required java source code files that are needed in order to compile the needed programs `preproc, indexer, ranker` on the `unixs.cis.pitt.edu` machine. Please make sure to follow completely the instructions about how to execute the programs and input/output formats.

Academic Honesty

The work in this assignment is to be done *independently*, by you and only you. Discussions with other students on the assignment should be limited to understanding the statement of the problem. **Cheating in any way, including giving your work to someone else, will result in an F for the course and a report to the appropriate University authority for further disciplinary action.**

How to submit your assignment

We will use a Web-based assignment submission interface. To submit your assignment:

- Go to the class web page `http://db.cs.pitt.edu/courses/cs1655/fall2014` and click the Submit button.
- You can submit up to five files at a time. You can come back and submit more files. Please note that files with the same name will overwrite previously submitted files.
- Use your pittID as the username and the last four digits of your PeopleSoft ID as the password. There is a reminder service via email if you forget your password.
- Upload your assignment file(s) to the appropriate assignment (from the drop-down list).
- Check (through the web interface) to verify what is the file size that has been uploaded and make sure it has been submitted in full. **It is your responsibility to make sure the assignment was properly submitted.**

You must submit your assignment before the due date (11:59pm, Wednesday, September 24th, 2014) to avoid getting any late penalty. The timestamp of the electronic submission will determine if you have met the deadline. There will be no late submissions allowed after 11:59pm, Friday, September 26th, 2014.

[Last updated on September 11, 2014 at 5:42pm EST]