Programming Assignment #6
CS253, Fall 2017
Due: Tuesday, Oct. 24th

"TFIDF"

Motivation & Description

In P2 through PA5, you have written code to read text files, separate punctuation from meaningful words, resolve capitalization, stem the words using the Porter algorithm and a table of exceptions, and count the frequencies with which words and punctuation streams occur. Now let's do something fun with that.

For PA6, we will recreate the matching component of early web browsers. They pre-processed text files much as you have done. They then compared files using a measure known as TFIDF: Term Frequency Inverse Document Frequency. That is what you are going to do for PA6: read in N files and compare how similar they are in topic using TFIDF.

Task

Your program will take three or more (generally more) file names as arguments. The first file is the stemming exceptions file, as in PA5. The remaining N files are input files. Your program will read all the input files and process them (including stemming with exceptions) as in PA5 to create a set of stemmed words. (For the purposes of this assignment, punctuation strings can be ignored.) Instead of printing word frequencies, however, you will use the word counts to compare input files to each other using TFIDF (see below). The output will no longer be a set list of term frequencies. Instead, the output will have N lines of text, where N is the number of input files, excluding the exceptions file. Each line of output contains N TFIDF similarity scores. The first line of output contains the TFIDF score between the first file (excluding the exceptions file) and itself, then the first file and the second file, the first file and the third file, and so on, ending with the score between the first file and the Nth file. The second line of input contains the TFIDF scores between the second file and the first file, the second file and itself, and so on. The scores on a line should be separated by whitespace.

The first step is to associate a TFIDF score with every word in the document (punctuation strings can be ignored). The TFIDF score for a word in a document is the product of two numbers, one of which you have already computed:

$$\mathrm{TFIDF}(\mathrm{word}, \mathrm{doc}) = \mathrm{tf}(\mathrm{word}, \mathrm{doc}) * \mathrm{idf}(\mathrm{word})$$

In this equation, *tf(word, doc)* is simply the number of times the word appears in the document. This is what you have been computing and printing in PA5. The second term is the inverse document frequency, and it is defined in terms of *N* (the total number of documents) and *n*, the number of documents containing at least one instance of the word:

$$\mathrm{idf}(\mathrm{word}) = \log_{10}(N / n)$$

The idea of idf is that matching a really common word is less important than matching a rare one. In the extreme, a word might appear in every document, in which case idf = log(1) = 0.

To measure the similarity between two documents, find the set of words that appear in both documents. For every word in the intersection of the documents, multiply the two TFIDF scores together, and sum those products. In other words,

$$\mathrm{Sim}(doc1, doc2) = \Sigma_w \mathrm{TFIDF}(w, doc1) * \mathrm{TFIDF}(w, doc2)$$

Where the summation is over all words (including capitalized words, etc., but excluding punctuation) that are common to both documents.

For PA6, your task is to compare every input document to every input document and print out the matrix of document similarity. As described above, this should be N lines of output, each with N similarity scores in order. Since similarity scores are higher when documents are similar, the diagonal elements of this table (which compare documents to themselves) should be large.

Submitting Your Work

To submit your program, first create a single tar file containing all of your source files (i.e. your .cpp and .h files) and your makefile, but not your compiled files (no executable or .o files, please). Then submit the tar file as PA6 on Canvas.

Grading Your Homework

To grade your assignment, the GTAs will unpack your archive in an empty directory. They will compile your code by typing 'make'. This command must compile your code from scratch, and it must produce an executable called PA6 in a directory called PA6. If your code does not compile using make, you will receive a zero for the assignment. Compiling your code should not produce any warning messages. Although we will not deduct points for warning messages, we may in future assignments so get in the habit of making sure you code compiles cleanly. Assuming your program compiles, it will be graded by how it performs on test cases, including test cases with errors in one or both of the input files. If the input file contains errors, you will receive full credit if your program returns -1 to main. (In case of an error, it should also print a meaningful error message to std::cerr.) If the input does not contain errors, your program should return 0 to main.

Hints

- We are still grading for correctness, not efficiency.
- Useful observation: Sim(A,B) = Sim(B.A)
- Similarly, Sim(A,B) >= 0
- Remember, Sim(A,B) depends on more than the documents in A & B. It also depends on the rest of the corpus (i.e. the whole set of documents), because of the idf term. Modify one document, and every similarity score may potentially change.

Policies

All work you submit must be your own. You may not submit code written by a peer, a former student, or anyone else. You may not copy or buy code from the web. The department academic integrity policies apply.

You may not submit your program late. To receive credit, it must be submitted on Tuesday, Oct. 24th. There is no late period. The exception is an unforeseeable emergency, for example a medical crisis or a death in the immediate family. If an unforeseeable emergency arises, talk to the instructor.