College of Computing and Information Science — CS6200
Northeastern University — Information Retrieval
Fall 2018 — October 22nd, 2018

Homework Assignment #3
Due Date: November 1st, 2018 @ 11:59pm

Instructions:

- The assignment is due on the time and date specified.
- This is an individual assignment. You may discuss the problems with your friends, but the code, analysis, interpretation and write-up that you submit for evaluation should be entirely your own.
- You are encouraged to use the Piazza discussion board, and seek help from TAs and instructors to get clarifications on the problems posed.
- If you receive help from others you must write their names down on your submission and explain how they helped you.
- If you use external resources you must mention them explicitly. You may use third party libraries but you need to cite them, too.

Goal: Implementing your own inverted indexer, text processing and gathering corpus statistics.

Task 1 (20points)**: Generating the corpus**

In this task you will be using the raw Wikipedia articles that you downloaded in HW1 Task 1 (non-focused, breadth-first, using the *Carbon_footprint* seed) to generate the clean corpus following the instructions below:

1- Parse and tokenize each article and generate a text file per article that contains only the title(s) and plain textual content of the article. Ignore/remove ALL markup notation (HTML tags), URLs, references to images, tables, formulas, and navigational components. Keep the anchor text while ignoring the associated URL.

2- Each text file will correspond to one Wikipedia article. The file name is the same as the article title, e.g. http://en.wikipedia.org/wiki/Carbon_footprint → Carbon_footprint.txt (file names must be unique).

3- Your parser should provide options for case folding and punctuation handling. The default setup should perform both. The punctuation handler is expected to remove punctuation from text but preserve hyphens, and retain punctuation within digits (mainly ",", ".", and any other symbols you deem necessary).

Task 2 (40 points): **Implementing an inverted indexer and creating inverted indexes (including compression).**

a. Implement a simple inverted indexer that consumes the corpus in Task 1 as input and produces an inverted index as an output.
Term frequencies *(tf)* are stored in these inverted lists:
TERM → *(docID, tf), (docID, tf), ...*

b. Store the length of the inverted list with each term.

Note: A **TERM** is defined as a *word n-gram*, and n = 1, 2, and 3. Therefore, you will have three inverted indexes, one for each value of n.

c. Store the number of terms in each document in a separate data structure.

d. Generate a positional inverted index for the unigrams. Encode the positions using the gaps between the occurrences.

Task 3 (20 points): **Using Positional Indexes for Boolean Proximity Queries**

1. Implement a program that uses a positional inverted index in Task 2.d to retrieve the list of documents that contain a pair of terms within a proximity window *k*.

2. Using the positional index created in task 2.d above:
    a. Find the list of documents that contain both ***carbon*** and ***emission*** within *k* = 5 and *k* = 10 unigram tokens of each other.
    b. Find the list of documents that contain both ***greenhouse*** and ***emission*** within *k* = 6 and *k* = 12 unigram tokens of each other.

    Note 1: The two terms should be separated by no more than *k* other tokens within the document. The order in which the terms appear in the document does not matter. Terms are not case-sensitive.
    Note 2: You will produce 4 lists of Document IDs in total.

Task 4 (20 points)**: Computing Corpus Statistics**

1- For each inverted index in Task 2a, generate a term frequency table comprising of two columns: *term* and *term frequency* (for the corpus).
Sort the table from most to least frequent.

2- For each inverted index in Task 2a, generate a document frequency table comprising of three columns: *term*, *docIDs*, and *document frequency*. Sort lexicographically based on term.
Note: For tasks 4-1 and 4-2, you will generate six tables in total: two tables for word unigrams, two tables for word bigrams, and two tables for word trigrams.

3- Generate three stop lists, one per index (from Task 2a). How would you choose your cutoff values? Briefly justify your choice and comment on the stop lists' contents.

What to hand in?

1- Your source code for solving all parts of this assignment.
2- A readme text file explaining in detail how to setup, compile, and run your program and what design choices you had to make.
3- The 4 inverted index files and the [Document ID, Term Count] table generated in Task 2.
4- The four Document ID lists generated in Task 3-2.
5- The six tables generated in Task 4.
6- The stop lists and explanations from Task 4-3.