CS 6200 Information Retrieval
Seattle/Fall 2018

Assignment 2 Data Transformation & Indexing

Announced Oct 9, 2018. [Sneak peek Oct 5, 2018]
Due by 9am, Oct 23, 2018.
Small changes made for clarification on Oct 18th, marked in red.

In Assignment 1, you implemented a simple web crawler, and used it to crawl a portion of Wikipedia. In this assignment, you will first transform crawled documents and to make them ready to be indexed. You will also write a simple inverted index system, to create an index for these documents and test the indexing.

In later assignments, you will develop retrieval & ranking systems, and an evaluation component. At the end of the course, you will have all the components in hand for a simple search engine. The search engine will be specialized and will be specific to the domain you choose to crawl.

Keep in mind that you will be using and building further upon this code and using the output of this code in the next assignment(s). You may also need to slightly modify this code as we get into later assignments. So make sure to write good, maintainable code.

For this assignment, you'll find a regular expression package like **re** will be a big help. Do not use any package or library that tokenizes text for you or any that creates an inverted index. You must write your own code, and ensure it is well-documented. ***As always, keep in mind the academic integrity rules we discussed.***

Now here are the inputs and outputs, and what's to be done in this assignment.

Inputs and Outputs

1. The input parameters to both the data transformer and the indexer will be:
    a. A folder named **FolderName** where there is a set of documents that you crawled. If you saved the contents of the 900 documents you crawled for Assignment 1, you can use those document contents. Otherwise you may have to run the crawler program again.
        Optional: If you want to handle a different domain
        Recall that you restricted the URLs to be crawled to those of the form https://en.wikipedia.org/wiki/ , so that made the domain specific to Wikipedia. If you want to crawl a different domain, you can choose a different starting URL, and one or more specific domains so that you crawl only URLs from those specific domain(s). Remember: to keep matters simple, you can choose to crawl only HTML or plain text documents, and avoid processing PDF, Microsoft Office formats, ODF, or purely image/audio/video files etc.
    b. An integer **NumFilesToProcess**. This specifies the number of files to process.

2. The outputs to be submitted will be as follows:

    a. One or more **program files** which implement this assignment.
    b. The **RunDataTransformer** script (described in Part 1 below).
    c. The **CreateIndex** script (described in Part 2 below).
    d. A file named **stats.txt** with the following information:

- Total file size of all the input files (in bytes):
- Total number of tokens across all input files:
- Total number of unique tokens across all input files:
- Total index size, that is total size of the three index files (in bytes):
- Ratio of the total index size to the total file size:

e. A file named requirements.txt (for Python submissions) which contains the list of installed packages, in requirements format. This makes evaluation easier and is good dev practice. You can get this file using the command
```
pip freeze > requirements.txt
```

f. A plain-text file named **README.txt** or a Markdown file named README.md which contains:
   i. A one or two sentence answer each to the questions: What was the most difficult part of this assignment? What was the easiest part?
   ii. Details about the format of the 3 index files (described in Part 2 below) you created.
   iii. A run of the query Q and the results you got (see Part 3).
   iv. Optional: Any additional information you may wish to provide about running the scripts.

Part 1: Data Transformation and Tokenization

In Part 1, you will write code to tokenize the documents you crawled previously. Assume that all the documents you get are in English. Minimally you must deal with markup like HTML tags and split the rest of each file into a series of tokens, as discussed in lecture on transforming data. In general, splitting on white space and punctuation is good. Keep things simple, and keep in mind the following:

a. Decide how to handle words with apostrophes (e.g. McDonald's, don't, Tim O'Hare). You can either delete the apostrophe, or remove it and replace with a space, for now. Be consistent in your transformations of queries and documents.
b. Convert acronyms like I.B.M. to IBM, U.N. to UN etc.
c. Skip (i.e. delete) any non-English or Unicode characters if they cause problems in your processing. *In general, do not worry about internationalization. You do not have to do any normalization, decompounding etc.*
d. Ignore odd combinations like .Net, 2Pac etc.
e. Ignore periods or commas in numbers. For example, given numbers 90.3, and 99,543, you can split them into tokens 90 3 and 99 543 respectively.
f. Do not (for this assignment) look for phrases or named entities or assemble ngrams.

Once you have the code ready, write a small (shell) script named **RunDataTransformer** (with an appropriate file extension like .sh, .py, etc) to run the code. This takes in the parameters *FolderName* and *NumFilesToProcess* as inputs and outputs a set of terms (tokens), one per line, that are obtained from processing the specified number of files from that folder. You will need to also keep track of which tokens come from which input file, so feel free to modify the line format, e.g. by adding the file name on each line (not very efficient) or adding the file name with a special marker as you start processing each new file.

Part 2. Indexing

In Part 2, you will develop code to create an inverted index. You will use the output from Part 1 as the input to this part. You will process all the terms or tokens (we'll call them terms from now on) from the specified set of files to create the following files that constitute an index:

(you can write out the files in binary file, or in JSON, or in some serialized form, but remember to document what you do in README.txt/README.md)

1. A file **TermIDFile** that contains data structure(s) that map terms into TermIDs and stores document frequencies with the TermIDs. The document frequency for a term is the number of files that the term occurs in.
2. A file **DocumentIDFile** that contains data structure(s) that map DocumentIDs to document names and stores the document length with each DocumentID. For document length, store the number of the tokens in the document. (For document length, you could alternatively store the document file length in bytes, but total number of tokens is better).
3. A file named **InvertedIndex** that stores a collection of inverted lists, one for each TermID. Each inverted list is a list of postings, where each posting contains a DocumentID and the relevant term frequency. This will be the biggest of the 3 files.

If it is not clear, the total number of unique tokens across all input files is the total number of terms in the TermIDFile, and also the total number of inverted lists in InvertedIndex.

To keep things simple, as you process input files, create an id for each new document and maintain a list of document names and document ids. Use the code from Part 1 to tokenize the document into terms. *For this assignment, you do NOT need to do use stop-words or do any stemming. You also do not have to compress the index.*

Suggestion: You may want to build the inverted index file in stages. As you handle each term from each document, first create a new TermID if the term is new and maintain a list of terms and unique TermIDs. Then keep a file for each unique term and add information about the document (or DocumentID) that contains the term. At the end, create the inverted index by processing these files.

Once this code is ready, write a small (shell) script named **CreateIndex** (with an appropriate file extension like .sh, .py, etc) to take in the parameters *FolderName* and *NumFilesToProcess* as inputs, call RunDataTransformer to generate tokens from files, and then create and output the 3 index files.

Finally (for this part), to test the index, write methods/functions in a file named **UseIndex** (with the appropriate .py~~, .java etc.~~ extension):

1. to takes in a term Term, and return the corresponding TermID,
2. to take a TermID and looks up its inverted list iList,
3. to take a term, find the TermID and the corresponding inverted list, and return the DocumentIDs where this term occurs,
4. to take a DocumentID and returns the document name it corresponds to.

and any other (helper) functions you need.

Part 3. Running the Code

1. When the coding is completed, run the **CreateIndex** script with the following parameters:
   FolderName: use the folder where you have the documents you crawled
   NumFilesToProcess: 900
2. Now choose a query Q appropriate to your domain. Test the functions in **UseIndex**, by submitting the query to function(s) there to return all the document names that the query Q occurs in. You can just run the query in a 'terminal' session, you do not need any fancy interface. Now you have the beginnings of a search engine!! Congrats!

   Save the query and the answers you got and add it to the README.txt file that you submit. More queries and results are welcome, but at least 1 is required.

Assemble the outputs, as specified in the Inputs & Outputs section above, and submit them.