ITEC4020 GROUP 006

**ASSIGNMENT 3**

Boyao Zhang

Shengnan Zhang

Wei Luo

Xiaoyi Li

**Contents:**

1. Introduction

2. Pre-processing input file

3. Indexing

4. Java applet for demonstrating output

5. Readme.txt

6. Appendix

Stopwords list

**REPORT**

# 1. Introduction

The objective is to gain experience in building indexes for a Web-based search engine.

Web search engines can provide us all kinds of information. It is a crucial issue for a Web search engine to provide accurate results of a given query. In this assignment, you are going to build an index for a set of given Web documents. You will be given a collection of Web documents. Each web document is embedded within tag <DOC> and </DOC> and has a unique document ID, which is also embedded within tag <DOCNO> and<DOCNO>.

In a sequential manner, what our program does is to:

1. takes crawler output-like input;
2. chuck it into separate parts;
3. remove all irrelevant strings of characters;
4. stem words into their root form;
5. index by hashmap with TF-IDF weighting scheme, document term weight;
6. demonstrate output index files in applet interface.

# 2. Pre-Processing Dataset.txt File

**Analysing dataset.txt**

In this Assignment, Dataset.txt can be considered as an example of crawler collected files. Raw crawler collected files usually appear in a form similar to xml file. It contains tags, meaningless words, punctuations, URLs, number and scramble codes etc. Our objective here is to index Dataset.txt into a file with only useful words in the right form. This process not only help search engine find the right ranking, also mitigate server burden. Here is the list of all kinds of elements/characters needs to be processed.

1. HTML tag striping
2. HTTP\_Link removal
3. Stopwords removal
4. Number removal
5. Stemming
6. special characters/ punctuations removal

**Steps of Process**

1. Chunking file for efficiency

Because of limited heap space when running such huge dataset on java, we have decided to use readline to chunk the dataset into smaller pieces. That way it is more efficient and you could notice errors (if any) at a much earlier stage rather than having to wait for a while and find out the program was at fault and incomplete. In our project, the file size for query topics in our case is smaller, we could parse it all at once, on the contrary, it is best to have the same consistency on how we process the documents for maximum efficiency.

1. jsoup-1.8.1.jar

Jsoup was used for data processing because documents can be parsed even if they include informal or improper tag. Moving forward, we loop through our parsed documents and load the dataset into our Java program for stop words removal and stemming. To do that, we get the text of each document from the chunk files, then we split the texts in *“string array”* with the removal of special characters and meaningless numbers.

1. weka-3.7.3.jar

we used Weka isStopword() to look for and remove terms/words that are irrelevant. The processes of HTML tag striping, HTTP\_Link removal, special characters/ punctuations removal and Number removal are also done at this part.

1. terrier-3.5-core.jar

Terrier Porter Stemmer performs stemming process on the remaining text as the final step before indexing. The purpose of stop words removal is to eliminate all high frequency words with no significant meaning and have no effect in information retrieval, such as “the”, “a”, “is” and so on. The purpose of stemming is to improve the recall and to reduce the size of the indexing structure by suffix removal or stripping of the word in English to set the words back to the stem. For example, “teacher”, “teaching”, “teaches” and “teach” will all be reduced to “teach”.

# 3. Indexing (inverted index)

After stop words removal and stemming, we output the dataset via Hashmap to count frequency for each term.

Also, we split output into 10 indexes for later demonstration uses.

Fig 1. 10 index files created

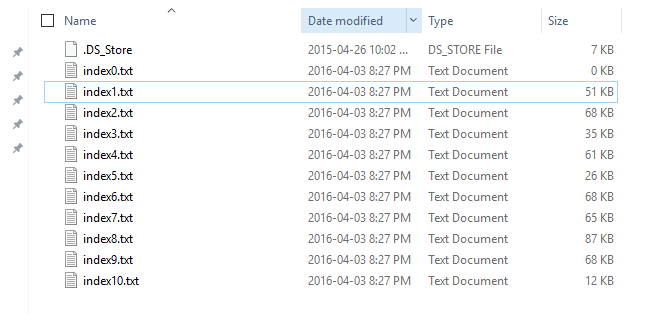
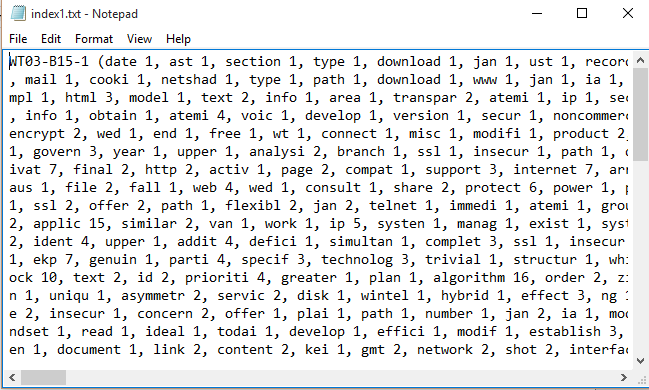


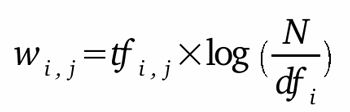
Fig 2. Example of one of 10 indexes



**Method/ theory used:**

TF-IDF (frequency–inverse document frequency) is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus. It is often used as a weighting factor in information retrieval and text mining. The TF-IDF value increases proportionally to the number of times a word appears in the document, but is offset by the frequency of the word in the corpus, which helps to adjust for the fact that some words appear more frequently in general.

**Formula for TF-IDF weighting scheme, document term weight:**

To have TF-IDF, we have the equation that

* tf is the still term i frequency
* IDF is Inverse Document Frequency as where N is the total number of documents in the dataset.txt and is the number of document containing term i in dataset

With the above calculation, we then have document represented as follow example:

WTX013-B14-470 ( 1, 1, 2, 1, 1 )

# 4. Output Index via Java Applet

Using Java applet as an interface to demonstrate output index 10 files:

1. The applet allow user to select and view specific index document;
2. The applet has scroll function to view specific part of an index document.



Fig 4. Applet output Demo

Besides the output from applet, index files are also stored at JAVA EE workspace.

# 5. README

README

Run environment:

1. Eclipse MARS .2 EE IDE (recommended)
2. Dataset.txt
3. Java APIs and Libraries, including:

JRE system Library [jdk1.8.0\_31] (recommended), jsoup-1.8.1.jar, terrier-3.5-core.jar, weka-3.7.3.jar, commons-io-2.4.jar

Please follow the following steps to run the program:

1. Create the following folders in the package directory (4020G6A3): chunk, data\_index
2. Import your package into Eclipse by clicking File at the left top corner, and select import from the menu, click import, select existing projects into workspace, and next, then select 4020G6A3.zip at select archive file and select the project contain within. Click finish to complete the import process.
3. Name and store dataset as “Dataset.txt” into 4020G6A3 in the JAVA EE workspace
4. Right click on your project, and select Properties, click on Java Build Path, go to the libraries tab, add jars by referencing to our libraries included in the lib folder in our package, click apply and ok.

Make sure all below .jar is in the library. If not, import them manually.

JRE system Library [jdk1.8.0\_31] (recommended), jsoup-1.8.1.jar, terrier-3.5-core.jar, weka-3.7.3.jar, commons-io-2.4.jar

1. If your computer has low java heap memory space, add “-Xmx8G” to extend maximum memory by selecting the little arrow next to the run button -> click on run configuration -> click on the argument tab -> copy and paste at the argument at the VM argument session -> click apply -> if you are already at the main class click run, otherwise click ok
2. Run “Viewmain.java” without any server configuration as a JAVA Applet
3. Result file will be outputted in the pop-out applet interface. The output will be stored in 10 .txt file. The applet is for the ease of use and view.

# 6. Appendix

Default English stopwords list from Rainbow

(weka-3.7.3.jar uses this stopwords list)

|  |  |
| --- | --- |
| a about above after again against all am an and any are aren't as at be because been before being below between both but by can't cannot could couldn't did didn't do does doesn't doing don't down during each few for from further had hadn't has hasn't have haven't having he he'd he'll he's her here here's hers herself him himself his how how's i i'd i'll i'm i've if in into is isn't it it's its itself let's me more most mustn't my myself no nor not of off on once only or other ought our ours | ourselves out over own same shan't she she'd she'll she's should shouldn't so some such than that that's the their theirs them themselves then there there's these they they'd they'll they're they've this those through to too under until up very was wasn't we we'd we'll we're we've were weren't what what's when when's where where's which while who who's whom why why's with won't would wouldn't you you'd you'll you're you've your yours yourself yourselves |