Project-2 Comp 479/6791
Fall 2020

Dept. of Computer Science and Software Engineering
Concordia University

Submitted to: - Dr. Sabine Bergler

Submitted by: Yashika Khurana
Student id: 40094722

Introduction

1. Sub project 1

In this module the main focus is to build the naive indexer, which have structure like “term”: [docid1, docId2…]. So to build the indexer I have used Reuters corpus 21578 and read all the 22 .sgm file. So first the document is split up according to the <!DOCTYPE lewis SYSTEM “lewis.dtd" which is the starting line of any document. Then all the documents are extracted by the tag which is the start tag ‘<REUTERS’

‘</REUTERS>’ which is the ending tag for the document.

After that Title, date and body tag is extracted to get information about the document. Then created the tokens with the doc id e.g. (tokens, id) and then sorted and removed all the duplicated tokens.

Then based on that indexer is developed using the python dictionary where key is the unique term and values are the list of dicid which have these terms. To calculate the time taken by indexer, ‘time’ module of the python is used. The output of the subproject 1 is three file- tokens.txt (list of tokens), sortedTokens.txt (sorted list of unique tokens) and InvertedIndex.json (json file contains the term and list of docid).

In total it take 45-55 seconds to build the naive indexer and the output can also be seen by running the main.py in the console.

2. Sub project 2

In this module I implemented the query processor for single term queries, i.e. queries that consist exclusively of one term. Input is the term and output is the ordered posting list. Sample queries run were

sample_queries = ["logic", "belt", "obtain", “Empire"] and

challenge_queries = ["pineapple", "Phillippines", "Brierley", “Chrysler”, "Philippines"].

The output of sample queries run on the naive indexer is “sampleQueries.json” and output of sample queries run on the index build on subproject 3 is “sampleQueries_subproject3.json”.

The output of the challenge queries on the naive indexer is “challengeQueries.json” and the output of the challenge queries on the subproject 3 is “challengeQueries_subproject3.json”

3. Sub project 3

In subproject 3 the main goal was to create the table similar to 5.1 but with the Reuters 21578. So Firstly I used the same index build in the subproject 1 to get the unfiltered tokens. In the first step removed the number, and second step done the case folding and in the third step removed the 30 stop words (reading the stop words from first30.txt), in the fourth step removed the 150 stop words(reading the stop words from first150.txt) from the link given in the project submission and in the final step applied the Porter stemmer.

To generate the table “texttable” module is used. The output table will also print in the console and store in the “Table.txt”. Final index generated after all the steps is stored in “invertedIndex_subproject3.json”.

To generate the table I have kept the precision of the floating values to get the findings in the detail. From the table we can see the stop words removal Don’t have much affect in the distinct terms, although the non-positional postings are affected nearly 20%.

The major difference analyses from the case folding which is nearly 24% and the non positional postings are 7%. The major difference in terms of non positional postings can be seen in 30 stop words. Stemming distinct terms leads to nearly decrement of 21% whereas its non positional postings leads to nearly 4% decrement in the size.

In total, after doing each steps leads to nearly 49% decrement of the distance terms and non positional postings leads to nearly 48% decrement in the size. So, in general we could say that both the distinct terms and non positional postings leads to nearly 48-49% decrement in the size.

	Distinct terms				Non positional postings	
	number	Δ%	T%	number	Δ%	T%
unfiltered	68153			1840246		
no numbers	65977	-3.193	-3.193	1618492	-12.050	-12.050
case folding	49849	-24.445	-27.638	1496948	-7.510	-19.560
30 stop words	49820	-0.058	-27.696	1294234	-13.542	-33.102
150 stop words	49700	-0.241	-27.937	1159961	-10.375	-43.476
stemming	38858	-21.815	-49.752	1103756	-4.845	-48.322

Table 1.1 lossy dictionary compression techniques

4. Queries report

Sample queries run were

sample_queries = ["logic", "belt", "obtain", “Empire"] and

challenge_queries = ["pineapple", "Phillippines", "Brierley", “Chrysler”, "Philippines"]

These queries were run on both the index-naive and final subproject 3 Index and verified the result by manual checking of the terms.

While quering the terms on naive index it gives the output of the posting lists but to run the queries on the final index generate in subproject 3 needs extra processing of terms since we have done processing case folding, removing of no, stop words removal and porter stemmer. So, it requires all the steps in the similar order. Then the queuing terms results are stored in the respective json file.

Please note that without processing it gives the result “No term found”, but after processing it gives the result which is stored in the json file.

For the term “Phillippines”-> phillippin after all the steps and term “Philippines" -> philippin after all the steps applied. Still for the term “Phillippines” there were no match in both the index. In all the queries Final index have more results compare to the naive indexer. Also the json file for running the queries in final index, the key stored in the file is processed query. The processing done is already mentioned above in the report.

5. Program structure

The program structure is divided into 4 main file, one is for sub project 1 i.e. ‘SubProject1.py’, second is for sub project 2 i.e. ‘SubProject2.py’, third is for sub project 3 i.e. ‘SubProject3.py’ and fourth is the main file which runs all the three module.

To run the project - **Please simply run the “main.py” python file** which calls the function of other sub project file i.e. subproject1(), subproject2() and subproject3().

6. Output files

There will be files generated by running the “main.py”

1. tokens.txt:

Contains list of tokens

2. sortedTokens.txt:

Contains sorted and unique tokens list

3. invertedIndex.json:

Contains the index generated from the sub project 1

4. sampleQueries.json:

Sample queries run on naive indexer

5. sampleQueries_subproject3.json:

Sample queries run on final compression indexer built in sub project 3

6. challengeQueries.json

Challenge queries run on naive indexer

7. challengeQueries_subproject3.json

Challenge queries run on final compression indexer built in sub project 3

8. invertedIndex_subproject3.json:

Contains the final index built in subproject 3.

9. Table.txt:

Contains the lossy dictionary compression techniques table results.