**Homework-2 Report**

Name: Rohith Chandra Kandambeth

**INDEX SIZE**

|  |  |
| --- | --- |
| Compressed | Stemmed | 60.6MB |
| Decompressed | Stemmed | 149.4MB |
| Decompressed | Unstemmed | 154.6MB |

Brief Explanation on the process used for Indexing

First, I loaded the documents (ap89\_collection) into a dictionary. Mapped the document id to integers (1 to 84674). Then I performed preprocessing – tokenization and removal of stop words. We made two dictionaries of documents with the terms and positions in them – one of which is stemmed and the other without stemming. I split each dictionary (stemmed and non-stemmed) into chunks of 1000 documents and converted each chunk into inverted index. The inverted Index for each chunk was saved into a file, in the format for example

– *term:doc position position,doc position position position,doc position;*

I created a catalog file for each chunk to maintain the start (intial offset) and size of the string. Using these values we can reference the corresponding index file. Next step is to create a merged catalog and merged index. I merged the catalog one by one and updated the mergedCatalog file, after each merge of the catalog the corresponding index was also merged using the mergedCatalog as a reference. We get the mergedCatalog and mergedIndex files. Finally we compress each index file by splitting the corresponding catalog into three sections. Using the offset and size values from the catalog we divide the index file into 3 parts and compress each part. The information regarding the start and size of each index is noted in a compressed\_metadata file.

**MODEL PERFORMANCE**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Index** | **Model** | **Old Score** | **New Score** | **Percent (New/Old)** |
| Decompressed | Stemmed | Okapi TF | 0.2448 | 0.1334 | 54.45% |
| Decompressed | Stemmed | Okapi BM-25 | 0.2517 | 0.1675 | 66.54% |
| Decompressed | Stemmed | Unigram LM with Laplace smoothing | 0.2279 | 0.1586 | 69.59% |
| Decompressed | Unstemmed | Okapi TF | 0.2448 | 0.2358 | 96.32% |
| Decompressed | Unstemmed | Okapi BM-25 | 0.2517 | 0.2912 | 115% |
| Decompressed | Unstemmed | Unigram LM with Laplace smoothing | 0.2279 | 0.2355 | 103% |
| Compressed | Stemmed | Okapi BM-25 | - | - | - |

**Inference on above results** *( Make sure to address below points in your inference)*

* *Explain how index was created*
* *Pseudo algorithm for how merging was done*
* *Explain how merging was done without processing everything into the memory ( Important )*
* *How did you do Index Compression ( For CS6200 )*
* *Brief explanation on the Results obtained*
* *How did you obtain terms from inverted index*

Algorithm used for merging – Stemmed and Non stemmed

1. Merge first two catalogs – create a dictionary containing information of the repeating terms and terms from first or second catalog
2. Using this dictionary create the updated catalog (mergedCatalog) with new offset values and size
3. Using the help of mergedCatalog Offset values and sizes, and the dictionary from the catalog files, merge the two index files into mergedIndex file
4. Next merge mergedCatalog with next catalog file and mergedIndex with next Index file
5. Repeat steps 2-5 until all files are merged

Here only the catalog file is added to the memory. Using the start and end index that we get from Catalog file we can extract parts of the index file using functions like seek(). The result for the stemmed file is better than nonstemmed, this could be because the queries are stemmed. Almost all retrieval models performed better than before for stemmed index. The terms are obtained from the inverted index using catalog file offset and size value for the term. Using the seek function we can start at any point in the file and extract the required part without loading the whole index file to memory.

**PROXIMITY SEARCH ( *For CS6200* )**

|  |  |
| --- | --- |
| **Index** | **Score** |
| Unstemmed | - |
| Stemmed | 0.2243(Okapi TF) |

**Inference on the proximity search results** *( Make sure to address below points in your inference)*

* *Which matching technique you have implemented*
* *Pseudo algorithm of your Implementation*

The algorithm used here is bigram MST. First, we get pairs of query terms and required document terms to calculate the proximity using minimum spanning tree . The results are then normalized and then integrated with okapi TF score.