HomeWork-2 Report Template

INDEX SIZE

Compressed | Stemmed	63.2 MB
Decompressed | Stemmed	161 MB
Decompressed | Unstemmed	167 MB

Brief Explanation on the process used for Indexing

1. The <DocNo> and <Text> have been extracted. The text content is then converted into lowercase and pre-processed.
2. The text has been tokenized following all the 3 rules mentioned and stop words have been removed.
3. A dictionary has been created in order to store the tokens and their corresponding positions in the documents.
4. This is used to create the inverted-index. This process has been followed for both, stemmed and unstemmed text.

MODEL PERFORMANCE

Index	**Model**	**Old Score**	**New Score**	**Percent (New/Old)**
Decompressed | Stemmed	TF-IDF	0.2835	0.2885	101.7%
Decompressed | Stemmed	Okapi BM-25	0.2800	0.2927	104.5%
Decompressed | Stemmed	Unigram LM with Laplace smoothing	0.2531	0.2319	91.6%

Decompressed | Unstemmed	TF-IDF	N/A	0.2283	
Decompressed | Unstemmed	Okapi BM-25	N/A	0.2358	
Decompressed | Unstemmed	Unigram LM with Laplace smoothing	N/A	0.2101	
Compressed | Stemmed	Okapi BM-25	N/A	0.2927	

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

- *Explain how index was created*
  To create an index, initially a dictionary was made to store the positions of each token document-wise. This was then used to convert into an inverted index. The indexes were stored in text files in string format by changing the format of the dictionary in order to reduce the size. Simultaneously, catalog files were made to keep a track of the positions, which inturn eliminates the need to load the index files into memory, increasing the processing time.

- *Pseudo algorithm for how merging was done*

  To merge the files, every 2 consecutive files has been merged into the corresponding first file.

```
WHILE n > 1 DO
    FOR i FROM 0 TO n - 1 BY 2 DO
        MERGE (i-th index file WITH (i+1)-th index file INTO (i/2)-th index file)

IF n IS ODD THEN
    MERGE ((n/2-1)-th index file WITH (n-1)-th index file INTO (n/2-1)-th index file)
```

- *Explain how merging was done without processing everything into the memory (Important)*

  Catalog files help us find where each word's information starts and ends in the index. They allow us to read specific parts of files without loading everything into memory, which makes things faster and uses less memory. Catalog files follow a particular format, including details like (TermId, Token_start_position, and length), which help locate the needed information easily.

- *How did you do Index Compression (For CS6200)*

Index compression was achieved by reading the input file in binary mode, compressing it using gzip, and writing the compressed data to an output file. Decompression was performed by reading the compressed file using gzip and then writing the decompressed data to an output file. Additionally, the `shutil` module in Python facilitated efficient copying of data between file-like objects, enhancing the compression and decompression processes.

- *Brief explanation on the Results obtained*

  The results indicate that employing stemming led to better precision scores across all vector space models. Conversely, when stemming was not applied (unstemmed), there was a decrease in precision scores, as anticipated. This highlights the significant impact of stemming on improving accuracy in information retrieval tasks.

- *How did you obtain terms from inverted index*

  Catalog files are structured with details like term IDs, starting positions of tokens, and their lengths. With this information, we can easily find where each token begins and ends in its respective index file. This means we can read specific parts of the file without needing to load everything into memory.

PROXIMITY SEARCH (*For CS6200*)

Index	**Score**
Unstemmed	0.2303
Stemmed	0.2921

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

- *Which matching technique you have implemented*

Minimum span matching technique has been used.

- *Pseudo algorithm of your Implementation*

```
FUNCTION find_min_span(position_lists):
    FOR pos_list IN position_lists DO
        IF len(pos_list) == 0 THEN
            RETURN 1e5
        END IF
    END FOR

    min_span = 1e5
    indices = [0] * len(position_lists)

    WHILE NOT all_indices_at_end(position_lists, indices) DO
        min_val = 1e9
        next_index = find_next_index(position_lists, indices)

        min_span = update_min_span(position_lists, indices, min_span)

        indices[next_index] += 1
    END WHILE

    RETURN min_span
END FUNCTION
```

Extra Credits

- *If any EC done, please explain using subheadings*
- *Include any precision increment/ results you have got using EC implementation*
- *If query Optimisation done, then include time in seconds taken for one query*
- *If you have added HEAD field, make a table depicting precision with and without Head Info*