HomeWork-2 Report

Karthik Chintamani Dileep

INDEX SIZE

Compressed | Stemmed	79.3
Compressed | Unstemmed	84.8
Decompressed | Stemmed	157.5
Decompressed | Unstemmed	162.5

Brief Explanation on the process used for Indexing

1. Text preprocessing: The documents are tokenized, lowercased, filtered for stopwords. Porter stemming is also applied. Two dictionaries are created one with stemming and one without.
2. Inverted index construction: An inverted index is built that maps each term to a posting list of document ids and term positions within that document. This is done both with and without stemming.
3. Batch creation: To handle the large data, the index is split into smaller batches of 1000 documents in each file with pickle and gzip compression applied. Multithreading is used for faster processing.
4. Merging batches: The smaller index batches are merged iteratively using a merge procedure while maintaining the compression. This results in a final merged index.
5. Term statistics: Additional term statistics like term frequencies (tf), document frequencies (df) are gathered from the final merged index to help with scoring schemes.

MODEL PERFORMANCE

Index	Model	Old Score	New Score	Percent (New/Old)
Decompressed | Stemmed	TF-IDF	0.2934	0.3015	102%
Decompressed | Stemmed	Okapi BM-25	0.2889	0.2976	103%
Decompressed | Stemmed	Unigram LM with Laplace smoothing	0.2604	0.2648	101%
Decompressed | Unstemmed	TF-IDF	N/A	0.2397	N/A
Decompressed | Unstemmed	Okapi BM-25	N/A	0.2277	N/A
Decompressed | Unstemmed	Unigram LM with Laplace smoothing	N/A	0.2262	N/A
Compressed | Stemmed	Okapi BM-25	0.2889	0.2976	103%

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*

Index Creation:

- The index was created by tokenizing, lowercasing, stopword removal on the documents. Porter stemming was applied.
- An inverted index was built that mapped terms to posting lists of document ids and term positions.
- To handle large data, smaller inverted index batches were created using multithreading. Batches were compressed using pickle and gzip.

Pseudo algorithm for merging:

- Followed an iterative merge approach by merging two catalogs/indexes at a time.
- The catalogs are loaded fully into memory since they are small. The index files are accessed sequentially in chunks.
- Each catalog is sorted by term_id.
- For the same term, the compressed posting lists are concatenated.
- For new terms, the posting list is directly copied to the merged index.
- The catalogs are updated with new offsets and sizes after each merge.
- In the next iteration, a new catalog file and index file is merged with previously merged catalog and index file. This continues iteratively over all the batches until one file merged index file and

catalog file are created.

Merging without Loading in Memory:

- The key is processing batches sequentially without loading everything into memory.
- Batch indexes are read from disk chunk-by-chunk, decompressed, processed and written out to the merged index.
- Only catalog files are loaded in memory.

Index Compression:

- Used pickle to serialize each posting.
- Applied gzip compression to reduce storage.

Results Explanation:

- Stemming improved the retrieval performance showing the importance of conflating word variants.
- BM25 and TF-IDF worked better than language models.

Obtaining Terms:

- The catalog contains the mapping from term id to offset and size in the index file.
- Used this to lookup into the index, decompress and decode the posting list string back into python dictionaries.

PROXIMITY SEARCH (*For CS6200*)

Index	**Score**
Unstemmed	0.2310
Stemmed	0.3038

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

I kept proximity threshold as 40. If terms are within 40 distance I would add tfidf score to its score or else I would only keep okapiBM25 score.
For proximity search. I got term positions for each query and created window for each document and calculated min and max value. If the absolute difference is more than the threshold I would return True else False.