CSE 272-01(HW1 Report) on BATCH RETRIEVAL (SEARCH ENGINE)

Ananya Das(adas13@ucsc.edu)

1. **Software:**
    - **your design decisions and high-level software architecture:**
        - ***Storage of documents***- I am using ByteBuffers directory to store the documents by using Lucene
        - ***Ranking***-I have used existing ranking models Tf, Tf-Idf, Boolean and Relevance feedback for scoring and ranking using Lucene along with my own ranking algorithm
        - ***Software architecture***- The software architecture primarily deals with reading files in the main.java class while indexing the documents in index.java class , searching the queries processed and removing stop words in the search.java class and scoring/ranking in the search.java class as well. It also has utils class where some utils method is written for reading a file in Java.The resources directory contains all the files used for eg: documents file, queries file, relevance file used for comparison
    - **Major data structures (if any):**
        - List to get the term vectors of the terms in the document while expanding the query for relevance feedback
    - **Any programming tools or libraries that you used:**
        - Apache Lucene Java(8.8.1)
        - IntelliJ Idea Ultimate
        - JDK 1.8
        - FileWriter for logging simple text file in Java
        - Lucene
            - StandardAnalyzer
            - ByteBuffersDirectory
            - Lucene-Core(8.8.1)
            - Lucene-analyzers(3.6.2)
            - lucene-queryparser(6.6.0)
            - IndexWriter
            - IndexWriterConfig
            - IndexReader
            - IndexSearcher

- Field
- TextField
- Document
- QueryParser
- ScoreDoc
- BooleanSimilarity
- ClassicSimilarity

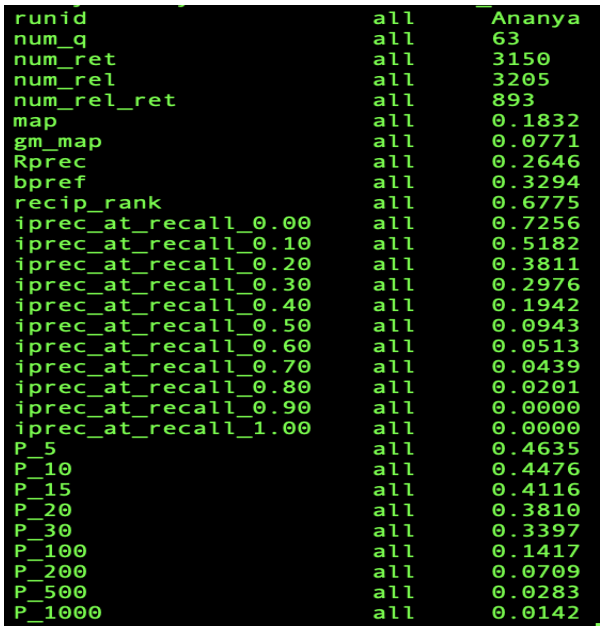
- **Strengths *and* weaknesses of your design, and any problems that your system encountered:**
  - ***Strengths***: The primary strength of the algorithm is it takes significantly less time for indexing as well as for searching since there are no such data structures used for indexing and searching.
  - ***Weakness***: One primary weakness is that it may need to be customized for very large corpus.
  - ***Problems***:
    - It took me a lot of time to understand Lucene in depth and to be able to use its various functions
    - While ranking i also encountered issues while doing relevance feedback since the normal TextField doesnt allow to store term vectors and then i got to know how to store term vectors and use it for relevance feedback
    - I started doing with PyLucene at first since the number of lines of code for Python is comparatively lesser but I could not make Pylucene work in my M1 Apple system and hence shifted to Java where I just created a Maven Project and added dependencies for any library that i had to use
    - It took me a lot of time to parse the data since I was creating a minor mistake while indexing the data
    - Finally, I was ableto index the data and then search it using my own search algorithm and then rank it using different approaches(Tf, Tf-Idf, Boolean, Relevance Feedback)

2. **Your customized new algorithm. Please provide enough detail so that others can also implement it; Some optional data is provided in the hw folder case you want to use them for your new algorithm.**
   - Deleting stop words is an essential part of any indexing and searching algorithm
   - My algorithm focusses on deleting stop words.It goes as follows:
     i. First, read the documents and queries file

ii. Then, index the documents using ByteBuffers directory and StandardAnalyzer
iii. ***Indexing***-Iterate through the documents file, pick the required information and create a new field. While indexing I am removing the unnecessary stop words so that it gives better precision
iv. Add the field to the document after indexing
v. And finally close the document writer after you are done with indexing
vi. ***Searching****:* Iterate through the queries file and read and process the queries
vii. Similar to the indexing step remove unnecessary words
viii. Parse the queries using QueryParser(thanks to Lucene!)
ix. And then score/rank using the ranking mechanism you decide
x. Finally, write the result according to trec_eval format for evaluation
xi. After executing thet trec_eval file on your log file, you can get the metrics of your algorithm.

3. **Experimental results: the Precision, AvgPrec, and running time statistics; any patterns you observed across the set of experiments, for example, about which algorithms or representations were effective; what worked well, what did not not;**
    - ***Precision***: Mean Average Precision(MAP): 0.1832
    - ***Avg precision***:GM_MAP: 0.0771
    - PFB the screenshots for statistics for different ranking algorithms:

Own Algorithm:



runid	all	Ananya
num_q	all	63
num_ret	all	3150
num_rel	all	3205
num_rel_ret	all	893
map	all	0.1832
gm_map	all	0.0771
Rprec	all	0.2646
bpref	all	0.3294
recip_rank	all	0.6775
iprec_at_recall_0.00	all	0.7256
iprec_at_recall_0.10	all	0.5182
iprec_at_recall_0.20	all	0.3811
iprec_at_recall_0.30	all	0.2976
iprec_at_recall_0.40	all	0.1942
iprec_at_recall_0.50	all	0.0943
iprec_at_recall_0.60	all	0.0513
iprec_at_recall_0.70	all	0.0439
iprec_at_recall_0.80	all	0.0201
iprec_at_recall_0.90	all	0.0000
iprec_at_recall_1.00	all	0.0000
P_5	all	0.4635
P_10	all	0.4476
P_15	all	0.4116
P_20	all	0.3810
P_30	all	0.3397
P_100	all	0.1417
P_200	all	0.0709
P_500	all	0.0283
P_1000	all	0.0142

Tf:

```
runid                   all     tf
num_q                   all     63
num_ret                 all     3150
num_rel                 all     3205
num_rel_ret             all     368
map                     all     0.0561
gm_map                  all     0.0075
Rprec                   all     0.1149
bpref                   all     0.1440
recip_rank              all     0.4416
iprec_at_recall_0.00    all     0.4675
iprec_at_recall_0.10    all     0.1906
iprec_at_recall_0.20    all     0.0997
iprec_at_recall_0.30    all     0.0534
iprec_at_recall_0.40    all     0.0226
iprec_at_recall_0.50    all     0.0169
iprec_at_recall_0.60    all     0.0052
iprec_at_recall_0.70    all     0.0000
iprec_at_recall_0.80    all     0.0000
iprec_at_recall_0.90    all     0.0000
iprec_at_recall_1.00    all     0.0000
P_5                     all     0.2127
P_10                    all     0.1762
P_15                    all     0.1651
P_20                    all     0.1460
P_30                    all     0.1328
P_100                   all     0.0584
P_200                   all     0.0292
P_500                   all     0.0117
P_1000                  all     0.0058
```

Tf-Idf:

```
runid                   all     tfidf-Both
num_q                   all     63
num_ret                 all     3150
num_rel                 all     3205
num_rel_ret             all     515
map                     all     0.0828
gm_map                  all     0.0224
Rprec                   all     0.1491
bpref                   all     0.1961
recip_rank              all     0.4806
iprec_at_recall_0.00    all     0.5206
iprec_at_recall_0.10    all     0.2763
iprec_at_recall_0.20    all     0.1512
iprec_at_recall_0.30    all     0.1005
iprec_at_recall_0.40    all     0.0614
iprec_at_recall_0.50    all     0.0304
iprec_at_recall_0.60    all     0.0142
iprec_at_recall_0.70    all     0.0128
iprec_at_recall_0.80    all     0.0000
iprec_at_recall_0.90    all     0.0000
iprec_at_recall_1.00    all     0.0000
P_5                     all     0.2762
P_10                    all     0.2460
P_15                    all     0.2201
P_20                    all     0.2087
P_30                    all     0.1889
P_100                   all     0.0817
P_200                   all     0.0409
P_500                   all     0.0163
P_1000                  all     0.0082
```

Boolean Retrieval:

runid	all	Boolean
num_q	all	63
num_ret	all	3150
num_rel	all	3205
num_rel_ret	all	478
map	all	0.0759
gm_map	all	0.0129
Rprec	all	0.1303
bpref	all	0.1695
recip_rank	all	0.5288
iprec_at_recall_0.00	all	0.5685
iprec_at_recall_0.10	all	0.2732
iprec_at_recall_0.20	all	0.1393
iprec_at_recall_0.30	all	0.0767
iprec_at_recall_0.40	all	0.0354
iprec_at_recall_0.50	all	0.0145
iprec_at_recall_0.60	all	0.0129
iprec_at_recall_0.70	all	0.0038
iprec_at_recall_0.80	all	0.0000
iprec_at_recall_0.90	all	0.0000
iprec_at_recall_1.00	all	0.0000
P_5	all	0.3111
P_10	all	0.2508
P_15	all	0.2243
P_20	all	0.2071
P_30	all	0.1788
P_100	all	0.0759
P_200	all	0.0379
P_500	all	0.0152
P_1000	all	0.0076

Relevance Feedback:

runid	all	RelevanceFeedback
num_q	all	63
num_ret	all	3150
num_rel	all	3205
num_rel_ret	all	452
map	all	0.0739
gm_map	all	0.0096
Rprec	all	0.1291
bpref	all	0.1682
recip_rank	all	0.4616
iprec_at_recall_0.00	all	0.4865
iprec_at_recall_0.10	all	0.2586
iprec_at_recall_0.20	all	0.1190
iprec_at_recall_0.30	all	0.0682
iprec_at_recall_0.40	all	0.0496
iprec_at_recall_0.50	all	0.0315
iprec_at_recall_0.60	all	0.0165
iprec_at_recall_0.70	all	0.0079
iprec_at_recall_0.80	all	0.0050
iprec_at_recall_0.90	all	0.0000
iprec_at_recall_1.00	all	0.0000
P_5	all	0.2571
P_10	all	0.2143
P_15	all	0.1873
P_20	all	0.1770
P_30	all	0.1614
P_100	all	0.0717
P_200	all	0.0359
P_500	all	0.0143
P_1000	all	0.0072

4. **What you have learned from this assignment.**
    - I learned to use Apache Lucene and for Java for batch retrieval
    - I learned to use Query Parser for parsing queries
    - I learned how to index, search , parse and score documents using different ranking mechanisms
    - I learned how Lucene has different ranking algorithms written in Similarity class for Tf, Tf-Idf, Boolean retrieval that can be used on the go
    - The log file is generated as log.txt on the project directory after the algorithm is executed

5. **Please put your software source code at github and submit the URL**
    - Github URL: https://github.com/ananyadas2607/CSE272_HW1_SearchEngine.git