Exercise 2 - Lucene, BM25, and evaluation

Group09
Samira Mahdi Zadeh- 1329639
Thiago Carneiro-1028762

The prototype

Our prototype for indexing and searching using Lucene 4.7.1 consists of 3 classes:

- IndexLucene.java
  In this class we created the Lucene index for 20_newsgroups_subset and stored it in a directory called “indexLucene”.
  To do so we used the “Standard analyzer” of Lucene to build up the tokens as bag of word model. The analyzer is also responsible for the vocabulary normalization (e.g. case folding and removing stopwords). Then we traversed the subset recursively to find the files to be indexed. For each file we created a Lucene Document which is the basic unit for indexing and searching and contains a list of fields. The fields that we stored in this document are:
- “path”: the path of file > indexed, the original field value stored in the index
- “contents”: the contents(text) of the file > indexed and tokenized
  In which the path value and contents were used in searching and printing results for SearchLucene.java

- SearchLucene.java
  In this class we used Lucene functionality to search for top 100 results from already indexed subset which was stored in “indexLucene” directory from previous run of IndexLucene.java.
  In order to make query from “topic” files, we read each file and excluded the especial characters (+ - && || ! ( ) { } [ ] ^ " ~ * ? : \ /) from it; because otherwise the queryParser of Lucene would manipulate them as a part of query syntax. We then parsed this query using the same analyzer (StandardAnalyzer) that we used for indexing and then called search method of Lucene to perform search and return top 100 matching results and finally we printed the results according to the specified format:
  topic1 Q0 misc.forsale\76057 1 5.6695824 group09_experiment2
  topic1 Q0 misc.forsale\76442 2 4.0179787 group09_experiment2
  …

- BM25LSimilarity.java
  Considering the assignment requirement for providing BM25L score calculator, we modified the Lucenes’ BM25Similarity class which implements the following formula:

$$f(q,D) = \frac{(K_1+1)c(q,D)}{k_1\left(1-b+b\frac{|D|}{avdl}\right)+c(q,d)}$$

To avoid overly penalizing of very long documents we introduced the δ parameter with default value of 0.5 as the new class member variable of our BM25L implementation and modified the normalization formula to:

$$f(q,D) = \frac{(K_1 + 1)[c(q,D) + \delta]}{k_1\left(1 - b + b\frac{|D|}{avdl}\right) + [c(q,d) + \delta]}$$

How to run the prototype

First run the Index Lucene by executing the jar file with the command:
java –jar IndexLucene.jar

To run the Search Lucene execute the jar file with the command:
java –jar SearchLucene.jar

After running the SearchLucene the user has to choose one of the 3 possible Similarity modes, that are Lucene Default, BM25 and BM25L that are selectable by entering 1, 2 or 3 respectively.

How the score is calculated for two documents adjacent in the ranked list

In order to do this we used a simple query of "small case" on our indexed subset and retrieved 100 most relevant results. We used Lucene's searcher.explain() method and print out the returned explanation objects as follows:

```
0.47844946 = (MATCH) sum of:
  0.28310627 = (MATCH) weight(contents:small in 1327) [DefaultSimilarity], result of:
    0.28310627 = score(doc=1327,freq=1.0 = termFreq=1.0
), product of:
      0.76923084 = queryWeight, product of:
        3.92574 = idf(docFreq=428, maxDocs=8000)
        0.19594544 = queryNorm
      0.36803812 = fieldWeight in 1327, product of:
        1.0 = tf(freq=1.0), with freq of:
          1.0 = termFreq=1.0
        3.92574 = idf(docFreq=428, maxDocs=8000)
        0.09375 = fieldNorm(doc=1327)
  0.1953432 = (MATCH) weight(contents:case in 1327) [DefaultSimilarity], result of:
    0.1953432 = score(doc=1327,freq=1.0 = termFreq=1.0
), product of:
      0.6389709 = queryWeight, product of:
        3.2609634 = idf(docFreq=833, maxDocs=8000)
        0.19594544 = queryNorm
      0.30571532 = fieldWeight in 1327, product of:
        1.0 = tf(freq=1.0), with freq of:
          1.0 = termFreq=1.0
        3.2609634 = idf(docFreq=833, maxDocs=8000)
        0.09375 = fieldNorm(doc=1327)

topic1 Q0 comp.sys.ibm.pc.hardware\60400 1 0.4784495 group09_experiment2
```

```
0.3987079 = (MATCH) sum of:
  0.2359219 = (MATCH) weight(contents:small in 1332) [DefaultSimilarity], result of:
    0.2359219 = score(doc=1332,freq=1.0 = termFreq=1.0
), product of:
      0.76923084 = queryWeight, product of:
        3.92574 = idf(docFreq=428, maxDocs=8000)
        0.19594544 = queryNorm
      0.30669844 = fieldWeight in 1332, product of:
        1.0 = tf(freq=1.0), with freq of:
          1.0 = termFreq=1.0
        3.92574 = idf(docFreq=428, maxDocs=8000)
        0.078125 = fieldNorm(doc=1332)
  0.16278599 = (MATCH) weight(contents:case in 1332) [DefaultSimilarity], result of:
    0.16278599 = score(doc=1332,freq=1.0 = termFreq=1.0
), product of:
      0.6389709 = queryWeight, product of:
        3.2609634 = idf(docFreq=833, maxDocs=8000)
        0.19594544 = queryNorm
      0.25476277 = fieldWeight in 1332, product of:
        1.0 = tf(freq=1.0), with freq of:
          1.0 = termFreq=1.0
        3.2609634 = idf(docFreq=833, maxDocs=8000)
        0.078125 = fieldNorm(doc=1332)

      topic1 Q0 comp.sys.ibm.pc.hardware\60415 5 0.3987079 group09_experiment2
```

According to the output of explanation, the total score of a query in a documents is equal to sum of the weight of each term in query in that document. So:

score = weight (“small”) + weight (“case”)

where: weight(term) = queryWeight x fieldWeight

where: queryWeight = idf x queryNorm ; fieldWeight = tf x idf x fieldNorm

putting them all together we come up with:

$$score(q,d) = \sum_{t\ in\ q} tf_{t\ in\ d}\ .(idf_t)^2\ .Norm_{f\ in\ d}\ .Norm_q$$

Which is actually a form of tf.idf similarity function used by Lucene as the Default similarity function.

Evaluation Results

	our index	Lucene Defalut Similarity	BM25	BM25L
topic1	0.1034	0.1736	0.1523	0.1346
topic2	0.0000	0.3775	0.5373	0.4603
topic3	0.0417	0.4687	0.6040	0.5538
topic4	0.5000	0.5333	0.5270	0.5196
topic5	0.2500	0.5759	0.7399	0.6922
topic6	0.0217	0.5000	0.5000	0.5000
topic7	0.0549	0.5700	0.6632	0.6369
topic9	0.0312	0.6429	0.7500	0.7500
topic10	0.6667	1.0000	1.0000	1.0000

topic11	0.0000	0.2112	0.1910	0.2064
topic13	0.0400	0.4450	0.4638	0.4753
topic14	0.0072	0.5954	0.5638	0.5674
topic15	0.0111	0.6760	0.7181	0.6884
topic16	0.0139	0.4854	0.4912	0.4742
topic17	0.0286	0.3011	0.4390	0.3883
topic18	0.0568	0.5174	0.5131	0.5155
topic19	0.0333	0.6667	0.6667	0.6667
all	0.1094	0.5141	0.5600	0.5429