Assignment 2: Retrieval Algorithm and Evaluation

Z534: Search Fall 2015

Task 1: Implement your first search algorithm

Based on the Lucene index, we can start to design and implement efficient retrieval algorithms. Let's start from the easy ones. Please implement the following ranking function using the Lucene index we provided through Oncourse (*index.zip*):

$$F(q, doc) = \sum_{t_i \in q} \frac{c(t_i, doc)}{length(doc)} \cdot \log\left(1 + \frac{N}{df(t_i)}\right)$$

, where q is the user query, doc is the target (candidate document in AP89), t_i is the query term, $c(t_i, doc)$ is the count of term t_i in document doc, N is total number of documents in AP89, and $df(t_i)$ is the total number of documents have the term t_i. Please use Lucene API to get the information. From retrieval viewpoint, $\frac{c(t_i,doc)}{length(doc)}$ is called normalized TF (term frequency), while $\log\left(1 + \frac{N}{df(q_i)}\right)$ is IDF (inverse document frequency).

The following code (using Lucene API) can be useful to help you implement the ranking function:

```
IndexReader reader = DirectoryReader.open(FSDirectory.open(Paths
                .get("/Users/chunguo/Downloads/index")));
IndexSearcher searcher = new IndexSearcher(reader);

/**
 * Get query terms from the query string
 */
String queryString = "New York";

// Get the preprocessed query terms
Analyzer analyzer = new StandardAnalyzer();
QueryParser parser = new QueryParser("TEXT", analyzer);
Query query = parser.parse(queryString);
Set<Term> queryTerms = new LinkedHashSet<Term>();
searcher.createNormalizedWeight(query,
false).extractTerms(queryTerms);
System.out.println("Terms in the query: ");
for (Term t : queryTerms) {
      System.out.println(t.text());
}
System.out.println();

/**
 * Get document frequency
 */
int df=reader.docFreq(new Term("TEXT", "police"));
System.out.println("Number of documents containing the term
\"police\" for field \"TEXT\": "+df);
```

```
            System.out.println();

            /**
             * Get document length and term frequency
             */
            // Use DefaultSimilarity.decodeNormValue(…) to decode normalized
            // document length
            DefaultSimilarity dSimi = new DefaultSimilarity();
            // Get the segments of the index
            List<LeafReaderContext> leafContexts =
reader.getContext().reader()
                        .leaves();
            // Processing each segment
            for (int i = 0; i < leafContexts.size(); i++) {
                  // Get document length
                  LeafReaderContext leafContext = leafContexts.get(i);
                  int startDocNo = leafContext.docBase;
                  int numberOfDoc = leafContext.reader().maxDoc();
                  for (int docId = 0; docId < numberOfDoc; docId++) {
                        // Get normalized length (1/sqrt(numOfTokens)) of the
document
                        float normDocLeng =
dSimi.decodeNormValue(leafContext.reader()
                                    .getNormValues("TEXT").get(docId));
                        // Get length of the document
                        float docLeng = 1 / (normDocLeng * normDocLeng);
                        System.out.println("Length of doc(" + (docId +
startDocNo)
                                    + ", " + searcher.doc(docId +
startDocNo).get("DOCNO")
                                    + ") is " + docLeng);
                  }
                  System.out.println();

                  // Get frequency of the term "police" from its postings
                  PostingsEnum de =
MultiFields.getTermDocsEnum(leafContext.reader(),
                              "TEXT", new BytesRef("police"));

                  int doc;
                  if (de != null) {
                        while ((doc = de.nextDoc()) !=
PostingsEnum.NO_MORE_DOCS) {
                              System.out.println("\"police\" occurs " +
de.freq()
                                          + " time(s) in doc(" +
(de.docID() + startDocNo)
                                          + ")");
                        }
                  }
            }
```

For each given query, your code should be able to 1. Parse the query using Standard Analyzer (Important: we need to use the SAME Analyzer that we used for indexing to parse the query), 2. Calculate the relevance score for each query term, and 3. Calculate the relevance score $F(q, doc)$.

The code for this task should be saved in a java class: EasySearch.java

Task 2: Test your search function with TREC topics

Next, we will need to test the search performance with the TREC standardized topic collections. You can download the query test topics from Oncourse (*topics.51-100*).

In this collection, TREC provides a number of topics (total 50 topics), which can be employed as the candidate queries for search tasks. For example, one TREC topic is:

<top>
<head> Tipster Topic Description
<num> Number: 054
<dom> Domain: International Economics
<title> Topic: Satellite Launch Contracts
<desc> Description:
Document will cite the signing of a contract or preliminary agreement, or the making of a tentative reservation, to launch a commercial satellite.
<smry> Summary:
Document will cite the signing of a contract or preliminary agreement, or the making of a tentative reservation, to launch a commercial satellite.
<narr> Narrative:
A relevant document will mention the signing of a contract or preliminary agreement , or the making of a tentative reservation, to launch a commercial satellite.
<con> Concept(s):
1. contract, agreement
2. launch vehicle, rocket, payload, satellite
3. launch services, commercial space industry, commercial launch industry
4. Arianespace, Martin Marietta, General Dynamics, McDonnell Douglas
5. Titan, Delta II, Atlas, Ariane, Proton
<fac> Factor(s):
<def> Definition(s):
</top>

In this task, you will need to use two different fields as queries: <title> and <desc>. The former query is very short, while the latter one is much longer.

Your software must output up to top 1000 search results to a result file in a format that enables the trec_eval program to produce evaluation reports. trec_eval expects its input to be in the format described below.

QueryID	Q0	DocID	Rank	Score	RunID
For example:					
10	Q0	DOC-NO1	1	0.23	run-1

10	Q0	DOC-NO2	2	0.53	run-1
10	Q0	DOC-NO3	3	0.15	run-1
:	:	:	:	:	:
11	Q0	DOC-NOk	1	0.042	run-1

The code for this task should be saved in a java class: SearchTRECTopics.java

Task 3: Test Other Search Algorithms

Next, we will test a number of other retrieval and ranking algorithms by using Lucene API and the index provided through Oncourse (*index.zip*). We consider the top 1000 documents retrieved for each query.

For instance, you can use the following code to search the target corpus via BM25 algorithm.

```
String queryString = "police";

String index = "/Users/chunguo/Downloads/index";
IndexReader reader = DirectoryReader.open(FSDirectory.open(Paths
            .get(index)));
IndexSearcher searcher = new IndexSearcher(reader);
Analyzer analyzer = new StandardAnalyzer();
searcher.setSimilarity(new BM25Similarity());

QueryParser parser = new QueryParser("TEXT", analyzer);
Query query = parser.parse(queryString);
System.out.println("Searching for: " + query.toString("TEXT"));

TopDocs results = searcher.search(query, 1000);

//Print number of hits
int numTotalHits = results.totalHits;
System.out.println(numTotalHits + " total matching documents");

//Print retrieved results
ScoreDoc[] hits = results.scoreDocs;
for(int i=0;i<hits.length;i++){
      Document doc=searcher.doc(hits[i].doc);
      System.out.println("DOCNO: "+doc.get("DOCNO"));
}

reader.close();
```

In this task, you will test the following algorithms

1. Vector Space Model (org.apache.lucene.search.similarities.DefaultSimilarity)
2. BM25 (org.apache.lucene.search.similarities.BM25Similarity)
3. Language Model with Dirichlet Smoothing (org.apache.lucene.search.similarities.LMDirichletSimilarity)

4. Language Model with Jelinek Mercer Smoothing
   (org.apache.lucene.search.similarities.LMJelinekMercerSimilarity, set λ to 0.7)

You will need to compare the performance of those algorithms (and your algorithm implemented in Task 1) with the TREC topics. For each topic, you will try two types of queries: short query (<title> field), and long query (<desc> field). So, for each search method, you will need to generate two separate result files, i.e., for **BM25**, you will need to generate **BM25longQuery.txt** and **BM25shortQuery.txt**

The code for this task should be saved in a java class: CompareAlgorithms.java

Task 4: Algorithm Evaluation

In this task, you will need to compare different retrieval algorithms via various evaluation metrics, i.e., precision, recall, and MAP.

Please read this document about trec_eval:
http://faculty.washington.edu/levow/courses/ling573_SPR2011/hw/trec_eval_desc.htm
And, you can download the trec_eval program from
http://trec.nist.gov/trec_eval/trec_eval_latest.tar.gz
We will use this code to evaluate the search result performance.

TrecEval can be used via the command line in the following way:
trec_eval -m all_trec groundtruth.qrel results (the first parameter is the ground truth file or to say judgment file, and the second parameter is the result file you just generated from the last task.
trec_eval --help should give you some ideas to choose the right parameters

You can download the ground truth file from Oncourse (*qrels.51-100*).

Please compare the different search algorithms (files generated in task2 and task 3) and finish the following tables:

Short query

Evaluation metric	Your algorithm	Vector Space Model	BM25	Language Model with Dirichlet Smoothing	Language Model with Jelinek Mercer Smoothing
R-prec					
recip_rank					
P@5					
P@10					
P@100					
MAP@5					
MAP@10					
MAP@100					

NDCG@5					
NDCG@10					
NDCG@100					

Long query

Evaluation metric	Your algorithm	Vector Space Model	BM25	Language Model with Dirichlet Smoothing	Language Model with Jelinek Mercer Smoothing
R-prec					
recip_rank					
P@5					
P@10					
P@100					
MAP@5					
MAP@10					
MAP@100					
NDCG@5					
NDCG@10					
NDCG@100					

Submission: Please submit the source code and report via Oncourse system.