Search Engines:
11-442 / 11-642

HW1: Boolean Retrieval
Due February 6, 11:59pm

The purpose of this assignment is to learn about Boolean retrieval and evaluation of retrieval results.

This assignment consists of three major parts:

1. Write software that will evaluate queries and produce a ranking of documents (50%);
2. Use your software to conduct retrieval experiments (25%); and
3. Write a report that describes your software, the experiments, and your conclusions (25%).

The report is an important part of your grade. Leave enough time to do a good job on it.

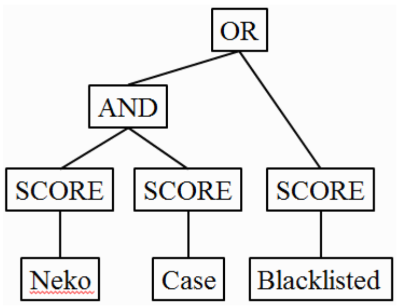
Software Development

Your software should run a single retrieval experiment. A single retrieval experiment is a loop over a set of queries.

For each query:

1. Read one query from the query file. which file, write java file by ourself?
2. Parse the query to create a corresponding query tree in which the internal nodes are query operators and the leaves are index terms. You will use a prefix query language, so parsing is easy. An example is shown below.

QUERY: #OR (#AND (neko case) blacklisted)



Your query parser should handle the following cases:
   a. If a query has no explicit query operator, default to #OR;
   b. If a query term has no explicit field (see below), default to 'body'; and
   c. Provide lexical processing (including stemming and discarding stopwords) of query terms using the provided `tokenizeQuery` method in the instructional code.

3. Evaluate the query, using a document-at-a-time strategy. The evaluation of a leaf node should fetch the inverted list for the index term, if one is available. Note that some query terms may not occur in the index.
4. Sort the matching documents by their scores, in descending order. The external document id should be a secondary sort key (i.e., for breaking ties). Smaller ids should be ranked higher (i.e. ascending order).
5. Write the information about the best *n* documents retrieved to a file. The trecEvalOutputLength parameter specifies *n*. Use trec_eval format (shown below).

What is score means

If the score is the same, we compare them with document id

You must compare two different scoring algorithms (step 3, above):

1. Unranked Boolean retrieval: Every document that matches the query gets a score of 1; and
2. Ranked Boolean retrieval: The score for matching a query term is its term frequency (tf) in the document.

Your software must support three query operators.

1. OR: return a document if at least one of the query arguments occurs in the document. Use the MAX function to combine the scores from the query arguments.
2. AND: return a document if all of the query arguments occur in the document. Use the MIN function to combine the scores from the query arguments.
3. NEAR/n: return a document if all of the query arguments occur in the document, in order, with no more than n-1 terms separating two adjacent terms. For example, #NEAR/2(a b c) matches "a b c", "a x b c", "a b x c", and "a x b x c", but not "a x x b c". The document's score will be the number of times the NEAR/n operator matched the document (i.e., its frequency). seems like there is an algorithm in the text book

The query language uses a prefix-style query language, because it is easy to parse. For example, the OR operator is defined as "#OR(term_1 term_2 ... term_n)".

Your software must also support **field-based retrieval** in which a term matches only against the specified portion of a document. The field name (if any) is specified in the query using a simple suffix-based syntax of the form 'term.field', as in 'apple.title'. If the query does not specify a field (e.g., if the user types 'apple'), your software should default to using the 'body' field.

The index provides 4 types of fields: 'url', 'keywords' (from the html 'meta' tag), 'title', and 'body'. The index provides the correct type of inverted list upon request, so to your software this field-based retrieval is just a matter of making the proper request to the index.

Source Code

Your software must be written in Java, and **must run under Java version 1.8** so that we can test it.

Example software that illustrates the main architecture of the search engine is provided for you as a guideline. This software is located at QryEval-3.3.2.zip and QryEval-3.3.2.tgz. The **documentation** describes the class hierarchy. You can extend this software to complete your homework assignment, or you can ignore it and write your own software from scratch.

Your software should be well written and provide clear documentation within the source code.

The example software uses classes from the Lucene search engine library. You can install a full copy of Lucene 6.6.0 on your laptop if you wish. However, a more convenient solution is to download just the necessary jar files (zip or tgz), and to include them on your Java classpath (e.g., java -cp ".:lucene-4.3.0/*" QryEval).

Input

Your software must accept only one parameter which is the name of a parameter file. The parameter file must be located in the current working directory. This parameter file must contain all of the parameters necessary to run your software. Your software must support the following parameters.

- **queryFilePath=** The path to the query file.
- **indexPath=** The path to the Lucene index directory. Typically this will be something like "indexPath=index".
- **trecEvalOutputPath=** The path to the file where your software will write its output for trec_eval.
- **trecEvalOutputLength=** The maximum number of documents per query to write to the trec_eval file.
- **retrievalAlgorithm=** Either "UnrankedBoolean" or "RankedBoolean".

An example parameter file is provided. You will need to update this file with the necessary query files path and index path. When testing your software, we will use a parameter file in the same format.

Output

Your software must write search results to the trecEvalOutputPath file in a format that enables trec_eval 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	GX270-76-5299838	1	16	run-1
	10	Q0	GX000-72-8784276	2	11	run-1
	10	Q0	GX000-25-2008761	3	9	run-1
	:	:	:	:	:	:
	11	Q0	GX000-38-6348494	1	18	run-1

The QueryID should correspond to the query ID of the query you are evaluating. Q0 is a required constant. The DocID should be the external document ID. The scores should be in descending order, to indicate that your results are ranked. The Run ID is an experiment identifier which can be set to anything.

When no documents are retrieved for a query, which may be the case for some structured queries, output a line with 'dummy' as DocID and 0 as score for your trec_eval file. For instance for query 10, your dummy output should look like below.

```
10     Q0     dummy          1     0     run-1
```

Testing Your Software

There are two ways to test your software.

1. Run your software on your computer, save the results to a file in trec_eval format, and upload the file to the trec_eval web service.

2. Package your source code in a .zip or .tgz file, and upload the file to the package to the software test web service. We will use this service to assess your software when we grade your homework, so we strongly suggest that you use it when developing your software.

These web services are accessed via the HW1 testing web page.

Data (Index)

The corpus is 588,146 documents from the Gov2 dataset. The corpus was indexed with Lucene. The Lucene index is available for downloading and running the experiments locally. The index is provided in two different file formats:

- A .zip file (2.3 GB or 2,383,134,677 bytes compressed, 2.9 GB uncompressed). md5 checksum: 3a908ec0c11108b69a7355f131cdf0af
- A .tgz file (2.3 GB or 2,381,913,572 bytes compressed, 2.9 GB uncompressed). md5 checksum: 0e36ebd86d4dcb4854502e1f858aa01d

You should download the index as soon as possible in order to avoid longer download times as the homework deadline approaches.

The Experiment

You must create three query sets.

1. **Bag-of-words (BOW) with #OR:** Use the query set exactly as it was provided, i.e., using only the default #OR operator for each query. This is the "high Recall" strategy.
2. **Bag-of-words (BOW) with #AND:** Use the query set exactly as it was provided, i.e., using only the provided #AND operator for each query. This is the "high Precision" strategy.
3. **Structured:** Manually create a set of structured queries that uses the three query operators (#OR, #AND, #NEAR/n). Use the same file format (*queryID:query*) as in the the query set while creating your structured queries file. Your goal is to improve over the baseline #OR queries (i.e., to get a good mix of Recall and Precision). You may decide how and where to apply the query operators to improve results, but you must use each query operator in at least 2 queries. At least 4 of your structured queries must use "url", "keywords", and/or "title" fields. A query can use just one type of field (e.g., "url") or several types of fields (e.g., "url" and "title"); you have the freedom to try different query structures.

How to improve the precision

Although the goal for the Structured set is to beat the Baseline, don't obsess over achieving high accuracy. However, do be sure that your software produces the correct results for each query set. The textbook notes "*... AND operators tends to produce high precision .. while using OR operators gives low precision ... and it is difficult or impossible to find a satisfactory middle ground*". In your report, you might want to comment on whether you had a similar experience.

Use trecEvalOutputLength=100 for your experiments.

You must test each query set (BOW-OR, BOW-AND, Structured) with each retrieval algorithm (unranked Boolean, ranked Boolean), thus you will have 6 sets of experimental results.

For each test (each query set) you must report the following information:

1. Precision at 10, 20, and 30 documents;
2. Mean average precision (MAP); and
3. "wall clock" running time.

The Report and Source Code

You must submit one .zip, .gz, or .tar file to the software test web service that contains a report and your source code. Your latest submission before the homework deadline will be used for grading. When you submit your materials for grading, you do not need to select tests to run. We will run a complete set of tests on your software.

1. **Report:** You must describe your work and your analysis of the experimental results in a written report. A report template is provided in Microsoft Word and pdf formats. Please address all the questions in this template report within the specified section. Do not change the section headings.

   Submit your report in pdf format. Name your report *yourAndrewID*-HW1-Report.pdf. When your uploaded file is unzipped, the report should be in the same directory as your source code.

2. **Source code:** Your submission must include all of your source code, and any files necessary to make and run your source code. Your source code must run within our testing service, so please check that it does before you make your final submission. We will look at your source code, so be sure that it has reasonable documentation and can be understood by others.

FAQ

If you have questions not answered here, see the HW1 FAQ and the Homework Testing FAQ.
