WSM Project 2: Building IR systems based on the Lemur Project

In this project you will implement several different retrieval methods, i.e. algorithms that given a user's request (query) and a corpus of documents assign a score to each document according to its relevance to the query. Some of these retrieval methods will be the implementation of the basic retrieval models studied in the class (e.g. TF-IDF, BM25, Language Models with different Smoothing). In this case, you will need the toolkits of Lemur Project, which includes search engines, browser toolbars, text analysis tools, and data resources that support research and development of information retrieval and text mining. You may also need to heavily read its wiki, Lemur Project and Indri Search Engine Wiki, in order to understand how to use the toolkits.

Document Corpus

Please download the WT2g data collection for this project (**Don't distribute it!!!**). The collection contains Web documents, with being a 2GB corpus. (Here you can find details about the corpus (WT10g instead), in case you are interested.) Use the corpus to test your algorithms, and run your experiments.

For the corpus, you need to construct two indexes, (a) with stemming, and (b) without stemming. Both indexes contain stopwords. For stemming, you can use the porter (Porter) or the kstem (Korvatz) stemmer. Below is an example of index information:

IndexID	Index Description	Statistics
0	WT2G with stemming	terms= 261,742,791 unique_terms= 1,391,908 docs= 247,491
1	WT2G without stemming	terms= 261,742,791 unique_terms= 1,526,004 docs= 247,491

Queries

Here is a set of 50 TREC queries for the corpus, with the standard TREC format having topic title, description and narrative. Documents from the corpus have been judged with respect to their relevance to these queries by NIST assessors.

Ranking Functions

Your task is to run the set of queries against the WT2g collection, return a ranked list of documents (the top 1000) in a particular format, and then evaluate the ranked lists.

Implement the following variations of a retrieval system:

1. Vector space model, terms weighted by Okapi TF (see note) times an IDF value, and inner product similarity between vectors.

   Note: You will have to use for the weights `OKAPI TF x IDF` where `OKAPI TF = tf/(tf + 0.5 + 1.5 * doclen / avgdoclen)`. For queries, Okapi TF can also be computed in the same way, just use the length of the query to replace `doclen`.

   Also note that the definition of `OKAPI TF` is `tf / tf + k1((1 - b) + b * doclen / avgdoclen)`. In the above formula, you can set k1 = 2 and b = 0.75, to end up with: `tf / (tf + 0.5 + 1.5 * doclen / avgdoclen)`.

2. Language modeling, maximum likelihood estimates with Laplace smoothing only, query likelihood.

   Note: If you use multinomial model, for every document, only the probabilities associated with terms in the query must be estimated because the others are missing from the query-likelihood formula (please refer to our slides).
   For model estimation use maximum-likelihood and Laplace smoothing. Use formula (for term i)

   $$\rho_i = \frac{m_i+1}{n+k}$$

   where m = term frequency, n=number of terms in document (doc length) , k=number of unique terms in corpus.

3. Language modeling, Jelinek-Mercer smoothing using the corpus, 0.8 of the weight attached to the background probability, query likelihood.

   The formula for Jelinek-Mercer smoothing is,

   $$\rho_i = \lambda P + (1-\lambda)Q$$

   where P is the estimated probability from document (max likelihood = m_i/n) and Q is the estimated probability from corpus (background probability = cf / terms in the corpus).

4. Implement any of your ideas to improve one of the above three IR models.

   You have to do something that can really improve the rank quality of the chosen IR models, and explain and showcase why your modifications can work.

Evaluation

Run all 50 queries and return at top 1,000 documents for each query. Do not return documents with score equal to zero. If there are only N<1000 documents with non-zero scores then only return these N documents. Save the 50 ranked lists of documents in a single file. Each file should contain at most 50*1000 = 50,000 lines in it. Each line in the file m ust have the following format:

query-number Q0 *document-id rank score* Exp

where *query-number* is the number of the query (i.e., 401 to 450), *document-id* is the external ID for the retrieved document, *rank* is the rank of the corresponding

document in the returned ranked list (1 is the best and 1000 is the worst; break the ties either arbitrarily or lexicographically), and *score* is the score that your ranking function outputs for the document. Scores should descend while rank increases. "Q0" (Q zero) and "Exp" are constants that are used by some evaluation software. The overall file should be sorted by ascending *rank* (so descending *score*) within ascending *query-number*.

Run all four retrieval models against the two WT2g indexes. This means you will generate 4 (models) * 2 (indexes) = 8 files, with at most 50,000 lines in total.

To evaluate a single run (i.e. a single file containing 50,000 lines or less), first download the qrel file (here you can find the qrel file for the WT2g corpus. Then, you can use the evaluation tool (ireval.jar) in Lemur Toolkit or you can download the script of trec_eval.pl and run:

```
perl trec_eval.pl [-q] qrel_file results_file
```

(The -q option outputs evaluation metrics values for each query; the average overall queries will be returned is -q is not used). trec_eval provides a number of statistics about how well the retrieval function corresponding to the results_file did on the corresponding queries) and includes average precision, precision at various recall cut-offs, and so on. You will need some of those statistics for this project's report, and you may find others useful.

OKAPI TF-IDF on query 401

We ran the okapi tf-idf model on query "401. foreign minorities, germany" for the WT2g collection (with stemming), without doing any fancy query processing (just word tokenization). Below you can find some of the statistics I got back by running trec-eval on the results for this query:

```
Queryid (Num):      401
Total number of documents over all queries
    Retrieved:     1000
    Relevant:        45
    Rel_ret:         42
Interpolated Recall - Precision Averages:
    at 0.00       1.0000
    at 0.10       0.4375
    at 0.20       0.3250
    at 0.30       0.3182
    at 0.40       0.2769
    at 0.50       0.2604
    at 0.60       0.2366
    at 0.70       0.2361
    at 0.80       0.2209
    at 0.90       0.0586
    at 1.00       0.0000
Average precision (non-interpolated) for all rel docs(averaged over queries)
                  0.2605
Precision:
  At    5 docs:   0.4000
  At   10 docs:   0.4000
  At   15 docs:   0.4000
  At   20 docs:   0.3500
  At   30 docs:   0.3000
  At  100 docs:   0.2500
  At  200 docs:   0.1850
  At  500 docs:   0.0800
  At 1000 docs:   0.0420
R-Precision (precision after R (= num_rel for a query) docs retrieved):
```

```
    Exact:        0.3111
```

We ran the language model with Laplace smoothing "401. foreign minorities, germany" for the WT2g collection (with stemming), without doing any fancy query processing (just word tokenization). Below you can find some of the statistics I got back by running trec-eval on the results for this query:

```
Queryid (Num):      401
Total number of documents over all queries
    Retrieved:     1000
    Relevant:        45
    Rel_ret:         35
Interpolated Recall - Precision Averages:
    at 0.00       0.3333
    at 0.10       0.3333
    at 0.20       0.3333
    at 0.30       0.2969
    at 0.40       0.2969
    at 0.50       0.2738
    at 0.60       0.2547
    at 0.70       0.0790
    at 0.80       0.0000
    at 0.90       0.0000
    at 1.00       0.0000
Average precision (non-interpolated) for all rel docs(averaged over queries)
                  0.1818
Precision:
  At    5 docs:   0.2000
  At   10 docs:   0.1000
  At   15 docs:   0.1333
  At   20 docs:   0.3000
  At   30 docs:   0.3333
  At  100 docs:   0.2500
  At  200 docs:   0.1450
  At  500 docs:   0.0660
  At 1000 docs:   0.0350
R-Precision (precision after R (= num_rel for a query) docs retrieved):
    Exact:        0.2889
```

We ran the language model with Jelinek-Mercer smoothing "401. foreign minorities, germany" for the WT2g collection (with stemming), without doing any fancy query processing (just word tokenization). Below you can find some of the statistics I got back by running trec-eval on the results for this query:

```
Queryid (Num):      401
Total number of documents over all queries
    Retrieved:     1000
    Relevant:        45
    Rel_ret:         39
Interpolated Recall - Precision Averages:
    at 0.00       0.2000
    at 0.10       0.0795
    at 0.20       0.0683
    at 0.30       0.0677
    at 0.40       0.0633
    at 0.50       0.0562
    at 0.60       0.0562
    at 0.70       0.0514
    at 0.80       0.0514
    at 0.90       0.0000
    at 1.00       0.0000
Average precision (non-interpolated) for all rel docs(averaged over queries)
                  0.0561
Precision:
  At    5 docs:   0.2000
```

```
  At   10 docs:   0.1000
  At   15 docs:   0.0667
  At   20 docs:   0.1000
  At   30 docs:   0.0667
  At  100 docs:   0.0700
  At  200 docs:   0.0600
  At  500 docs:   0.0520
  At 1000 docs:   0.0390
R-Precision (precision after R (= num_rel for a query) docs retrieved):
    Exact:        0.0889
```

If your system is dramatically under-performing that, you probably have a bug.

What to hand in

Provide a short description of what you did and some analysis of what you learned. You should include at least the following information:

- Un-interpolated mean average precision numbers for all 8 runs.
- Precision at rank 10 documents for all 8 runs.
- An analysis of the advantages or disadvantages of stemming, IDF, and the different smoothing techniques.

Feel free to try other runs to better explore issues like the last on mentioned above, e.g. how does Okapi tf compares to raw tf, what impact do different values of lamba have on smoothing, etc.

Hand in a printed copy of the report (in English, one column with at least 5 pages). The report should not be much longer than 20 pages (if that long). In particular, do not include trec_eval output directly; instead, select the interesting information and put it in a nicely formatted table.

Grading

This project is worth 150 points. To get about 100 points, you need to do the things described above. Additional points will be awarded for doing things such as:

- Extra runs, particularly interesting runs that do more than vary parameters (up to 15 points).
- Recall/precision graphs, per query or more interestingly averaged over all queries (up to 15 points).
- Analysis beyond the minimum required (up to 20 points). A good analysis should investigates interesting phenomena in details. For instance, failure analysis per query, i.e. looking at the query-by-query performance and explore what makes some queries fail, etc.

Submission Details

- Due: 18:10 in class, Tuesday, 21 May
- Late policy: the penalty for late homework is 20 points per day.