CS 4300/INFO 4300 Information Retrieval Due Oct 30, 2014

Project 1

Instructor: Chris Buckley *Total Points: 100*

Course Policy: Assignments are due Thursday, 10/30 11:39am via CMS and in hard copy at the beginning of class on Thursday, 10/30.
Write your NetIDs on the first page of the submitted hard copy.
Projects must be done in groups of 2 or 3.
Late assignments lose 10 points the first day, and an additional 5 points each day after that. Late hard copies should be submitted directly to the TAs. Assignments submitted after solutions are made available will not be accepted.
All sources of material must be cited. Assignment solutions will be made available along with the graded project solutions. The University Academic Code of Conduct will be strictly enforced, including runnign cheating detection software.

For Homework 2, you downloaded Lucene and ran an experiment with the CACM test collection. In Project 1, you'll build on that, first investigating Lucene's default similarity, and then using Lucene to help you build your own mini search engine to look at several other similarity alternatives.

You'll be running your experiments on two collections, the CACM collection previously used, and the Medlars collection. Medlars is another small collection (1033 documents, 30 topics) of medical documents. It will be available via download from CMS, if it is not already there, and will have the same format as the CACM collection.

1. Implement the MAP evaluation measure within the Java code used for Homework 2. Run that setup (stopwords turned off and no stemming) with the default Lucene similarity, retrieving 100 documents per query, evaluated with the MAP measure. Do this for both the CACM and Medlars collections.

2. Implement your own mini search engine. In the interest of simplicity later, you should NOT build an inverted index for this project, but instead build a direct document index, storing an indexed vector with frequency counts for each document in the collection. Your direct index should have methods to both iterate over all the documents, and to fetch a random indexed document vector. Use the TokenStream from the Standard Tokenizer in Lucene to tokenize the documents. The tokenizer should be setup to both remove stopwords and perform stemming. The tokenizer filters should use the Indri stopword list (included in the Homework 2 data in data/stopwords/stopwords_indri), and the Porter stemmer (already implemented within Lucene). Index both the CACM and Medlars document collections. Give a short (half-page or so) description of your mini implementation including dictionary approach, format of your direct index, and any other supporting permanent data structures.

3. Implement the following tf*idf similarity measures in your mini search engine. For each of the measures, run on the two collections you indexed in part 2, retrieving

the top 100 documents, and reporting the MAP evaluation figures for your two retrievals. You'll be iterating over each indexed document vector in the collection for each query, calculating a score for each Remember to use the same stemming and stopwords for the queries as you did for the documents!

The tf*idf variants are given using SMART notation - a run is two triples ddd.qqq describing the weighting schemes for the document and the query. The first element of the triple gives the tf variant, the second gives the idf variant, and the 3rd gives the length normalization.

a) atc.atc weighting

a:	augmented tf	tf component is $0.5 + \frac{tf}{max\_tf\_in\_doc}$
t:	idf	idf component is $\log(\frac{N}{n})$
c:	cosine	cosine normalization

b) atn.atn weighting

a:	augmented tf	tf component is $0.5 + \frac{tf}{max\_tf\_in\_doc}$
t:	idf	idf component is $\log(\frac{N}{n})$
n:	none	no normalization

c) ann.bpn weighting (an extended probabilistic model)

a:	augmented tf	tf component is $0.5 + \frac{tf}{max\_tf\_in\_doc}$
b:	binary tf	tf component is 1 or 0 (term present or missing)
n (second):	none	no idf
p:	idf	idf component is $\max(0, \log\frac{N-n}{n})$
n (third):	none	no normalization

d) Your choice! Come up with your own tf*idf variant that you think may perform well. Report what it is, and why you think it is worth looking at. (You might consider it in light of question 5 below, but you don't have to).

4. Implement the BM25 similarity measure, assuming that no relevance information is available. Set parameters b=0.75, k1=1.2, k2=100. Run on the two collections you indexed in part 2, retrieving 100 documents, and reporting the MAP evaluation figures for your two retrievals.

5. You did not implement an inverted index approach in your Mini implementation, using a direct index instead. Discuss possible problems in implementing methods 3a and 3b if you had used a inverted index implementation. What are the difficulties (especially in a dynamic collection)? What approximations could be done? Write about a half page explaining your answers.

6. Failure analysis. Choose one of the two collections and take the best performing retrieval run (as determined by MAP) from parts 3 and 4 on it. Find a query that performs poorly in that run, but substantially better (according to MAP) on some of the other runs (you'll have to print out query by query MAP performance for all your

runs). Analyze why that was the case – what weighting factors resulted in the poor performance for this query in your "best" run and the better performance in other runs? Look at some of the final document weights in the top retrieved documents, and the unretrieved relevant documents for the runs and how the different weights affected the different rankings. Repeat this for a query that performs well in your best run, but poorly in other runs. Your analysis itself should take about a page, with an additional page or so of supporting evidence – possibilities here might include: query vectors, partial document vectors, partial ranked results file, evaluation results. Note that this will take some time on your part, and perhaps some minor programming.

7. Discuss your results from parts 1, 3, and 4. Which measures worked well? Which measures were poor and why might they have been poor? What might account for any differences between the collections? What would you recommend? This discussion itself should be about a page.

8. Include a rough description of the contributions of your team members (one line per person).

To turn in.

- A report with your description and discussion of what you did, answering the questions above.

- A folder called mini which contains all your code building the mini indexer in part 2 and the evaluator and similarity functions of parts 3 and 4. Include a file "README" which explains what is there and how to use it. Include an executable shell script that can be used to run your search engine, if that is appropriate.

- A folder called lucene which contains your code for part 1.