CS533 Information retrieval
Project Part III
Due: November 29, 2016, 12:59PM

The retrieval model used for part-3 is the language model with query likelihood ranking. The following is a description of this language model and the formula we will use for ranking.

The language model with query likelihood ranking:
Language models are used to represent text in applications such as speech recognition, machine translation, and handwriting recognition. The simplest form of a language model, is the unigram language model. This model associates an occurrence probability for every word in the collection. For example if the documents contained only five different words the language model might be (0.2, 0.1, 0.35, 0.25, 0.1).

For search applications we use language models to represent the *topical* content of the document. The words associated with the main topic of a document should have high probability. We can represent either the documents as language models, or the topic of a query as a language model.

In the query likelihood retrieval model we rank documents by the probability that the query words could be generated by the document language model. In other words we calculate the probability that we could pull the query words out of the bucket of words representing the document.

Since we start with a query we would in general like to calculate P(D|Q) to rank the documents. Using Bayes' Rule we calculate this by P(D|Q) = P(Q|D)P(D)/P(Q).

Since we are only interested in ranking we can write rank(P(D|Q)) =rank(P(Q|D)P(D)). P(D) is the prior probability of a document. P(D) is assumed to be the same for all documents and so will not affect the rank. So we need to compute P(Q|D).

Assuming the unigram model for the document (in which terms occur independently) we get

$$P(Q|D)=\prod_{i=1}^{n} P(q_i|D)$$

where n is the number of terms in the query. P(q_i|D) can be estimated by

$$P(q_i|D) = tf_{q_i,D}/|D|$$

where $tf_{q_i,D}$ is the number of occurrences of term q_i in the document, and |D| is the number of tokens in D.

One problem with the formula above is that if one query word is missing from the document, its probability will be 0 and this document will not be retrieved. Another observation is that the probability of unseen words that are related to the topic of the query should not necessarily be 0.

To introduce the probability of unseen words to the formula we use the technique of smoothing. We lower (or discount) the probability estimates for words that are seen in the document and assign the "leftover" probability to the estimates for words that are not seen in the text.

The estimates for unseen words are usually based on the frequency of words in the whole collection. If P(q_i|C) is the probability for word *i* in the collection C, then the estimate we use for an unseen word in a document is wP(q_i|C) where w is a coefficient controlling the probability assigned to unseen words.

In order that the probabilities sum to 1, the probability estimate for a word that is seen in the document is now:

$$P(q_i|D) = (1-w)\frac{tf_{q_i}}{|D|} + w\frac{cf_{q_i}}{|C|}.$$

This form of smoothing is called the Jelink-Mercer method. Substituting this estimate in the document score for the query likelihood model gives:

$$P(Q|D) = \prod_{i=1}^{n} ((1-w)\frac{tf_{q_i}}{|D|} + w\frac{cf_{q_i}}{|C|})$$

Small values of 'w' produce less smoothing and therefore act more like an AND query. Large values of w causes to query to act more like an OR query.

For this part of the project use w=0.1

***Example*:**
In this example the database has 6 documents:

docId	Document path
1	/d3003/APW19981017.0151
2	/d3003/APW19981017.0306
3	/d3005/APW19981104.0772
4	/d3005/APW19981105.0282
5	/d3010/APW19981106.0520
6	/d3005/APW19981106.0572

Assume a query Q is "Pinochet killings" will be converted in to "pinochet killing".
Document 1: has tf = 9 for pinochet, and tf = 1 for killing. The length of the document |D|=239.
Document 6: has tf= 1 for killing. (It does not contain the term pinochet.)
The term pinochet occurs 24 times in the collection.
The term killing occurs 4 times in the collection
And the length of the collection is |C| = 954

For document 1:
log2(P(Q| D1)= log2(0.9(9/239)+0.1(24/954)+ log2(0.9(1/239)+0.1(4/954) = -12.6802

For document 6:
$\log_2$(P(Q| /D6)= $\log_2$(0.9(0/603)+0.1(24/954)+ $\log_2$(0.9(1/603)+0.1(4/954) = -17.6656.

This program will perform the retrieval. For simplicity each query q is a list of terms. When you tokenize the query use the same rules as those used to tokenize the document.

First, the program will read the dictionary, postings file, the document table, and |C|, generated by previous assignment part 2.

Then it will start to perform the query processing:

The program should be able to read from the screen many queries, and will terminate only after you type the command EXIT. For each query, it will print the *query*, and then a list of retrieved documents ordered by non-increasing rank. For each result it will print:

1. The directory path to the document, the headline, and the computed "probability"
2. The static snippet

For *example*.

The query is "**Pinochet killings**"
The document is /d3003/APW19981017.0151
The headline is "Lawmaker: Britain must question Gen. Pinochet about killings"
The snippet is "LONDON (AP) _ An influential lawmaker from the governing Labor Party on Saturday backed Spanish requests to question former Chilean dictator Gen. Augusto Pinochet, in London for back surgery, on allegations of genocide and terrorism…."

So the output file will look like below:

Lawmaker: Britain must question Gen. Pinochet about killings
d3003/APW19981017.0151
Computed probablity: -12.999
LONDON (AP) _ An influential lawmaker from the governing Labor Party on Saturday backed Spanish requests to question former Chilean dictator Gen. Augusto Pinochet, in London for back surgery, on allegations of genocide and terrorism…

You query can return multiple documents, then you need to include the result to output file similarly one after another with a single line gap. List the top 5 relevant documents in the output file.

The query and the results will be written also to an output file called output.txt.
If the query did not retrieve any documents the program will print NO RESULTs in the line following the query.

Please run your program with the following queries. The documents shown after each query will be retrieved as well as other documents. They are not sorted by rank.

Please run your program with the following queries. The documents shown after each query will be retrieved as well as other documents. They are not sorted by rank.

human rights violations
d30003t/APW19981017.0151
d30003t/APW19981017.0306
d30003t/APW19981017.0477
d30003t/NYT19981018.0123
d30003t/NYT19981018.0185

osama bin laden indictment
d30005t/NYT19981110.0432
d30005t/NYT19981017.0140
d30005t/NYT19981016.0283
d30005t/NYT19981007.0384
d30005t/NYT19981007.0383
d30005t/APW19981105.0282

bomb attack in israel

```
d30010t/APW19981106.0520
d30010t/APW19981106.0572
d30010t/APW19981106.0851
d30010t/APW19981107.0116
d30010t/APW19981107.0118
d30010t/NYT19981106.0494
```

celestial firework in asia

```
/d30012t/APW19981117.1226
/d30012t/APW19981117.0597
```

Swissair flight crash

```
d30016t/APW19981006.0543
d30016t/APW19981027.1075
d30016t/APW19981027.1082
d30016t/APW19981105.0853
d30016t/APW19981122.0381
d30016t/APW19981122.0382
d30016t/NYT19981001.0440
d30016t/NYT19981029.0472
d30016t/NYT19981112.0477
d30016t/NYT19981118.0185
```

Submission directions:

1. You may write the code using C, C++ or Java. Your program should compile on **remote.cs.binghamton.edu** No exceptions. If you have trouble accessing the remote server please contact CS system administrator.
2. Your need to submit a **.tar.gz** file (not only .tar) which should follow the following naming convention. <usrid>_<language>_part3.tar.gz, after unpacking this .tar.gz it should have a directory named <usrid>_<language>_part3. Example: if your user id is psaha4 and you are writing the assignment in java then your submission file name will be **psaha4_java_part3.tar.gz**. For c++ and c developers it will be psaha4_cpp_part3.tar.gz and psaha4_c_part3.tar.gz respectively.
3. Your submission folder should **not** contain input files, object or class files and readme file. Presence of these file will cost you 5 points.
4. This time your submission directory will contain project part2 directory (unzipped) along with the source code of the part 3 and a makefile for part 3.
5. **For Java developers** your entry point class ,which contains the main method, should have the name as “asignmentP3.java”. So make sure your makefile will produce a assignmentP3.class file after compilation and it can be run by the following command **java assignmentP3**
6. All your .java files should be inside a single folder, don’t create many packages to place your java files. All source code files should be inside a single directory like below.

7. **<u>For C++ and C developers</u>** please write your makefile such a way that it produces object file with name assignmentP3.obj and it can be run with the below command **./assignmentP2.out .**All source code files should be inside a single directory like below.

8. TA will run your part2 code first to generate the required csv files. Then he will run the part 3, which will refer those files to calculate the relevance. Make sure your part3 program can read files generated by by part2 from the part2 directory. If your project 2 has any issues reported by TA make sure you resolve them and include changes. TA will not accept any pre generated output files from part2. If part2 is not working then TA will not grade part 3.
9. Don’t generate any console output. Your output result will be dumped into an output file named result.txt (you can hardcode the filename) inside your project file.