Programming Assignment 3: Relevance Feedback

Assigned: Mar. 6
Due: Apr. 3

As discussed in the class notes on Query Log Mining, relevance feedback is a classical IR technique for improving the quality of search results. It works as follows: You issue an initial query to a search engine. The search engine returns a set of pages. You extract some number of content words that are frequent in those pages and that are closely associated with the query words. You create a new query which is the original query plus the new words, and resubmit to the search engine. Hopefully, the new results will be better than the first set of results.

In true relevance feedback, the user takes the results returned by the search engine and marks the ones that are relevant for his/her purposes, and those are used to extract the new word. In pseudo-relevance feedback, all the pages in the returned by the search engine are used.

For this assignment, you will supplement the query engine you built for programming assignment 1 with pseudo-relevance feedback.

Query Engine

To the front end that you created for programming assignment 1, add an optional box, labelled “Number of terms for relevance feedback”. If left blank, this defaults to 0, and the functionality of the program is unaffected.

If the “Number of terms” field is a non-zero value $K$, then the program proceeds as follows:

- Carry out the specified query $Q$ from the index, using the default Lucene search and score. Get the top 10 documents returned, or all the documents returned, whichever is fewer. Let $r(Q)$ be the set of documents returned.
- Strip html tagging, metadata in the `<head>` section and stopwords.
- For each word $W$ that is in any document in $r(Q)$, but not in $Q$ itself, compute the pseudo-relevance score for $W$, as described below.
- Add the $K$ top-scoring words to the query. Repeat the query. Display the results in the same format as in programming assignment 1.

Score

The score of any word $W$ for a given query $Q$ and a fixed collection of documents is computed as follows.

Let $g(W)$ be the number of indexed documents containing word $W$ (the Lucene function `docFreq` computes that.)

Let $N$ be the total number of indexed documents.

Let $z(W, Q)$ be the number of documents in $r(Q)$ that contain $W$. Let

$$y(W, Q) = max(0, \frac{z(W, Q)}{|r(Q)|} - 2 \cdot \frac{g(W)}{N})$$
Thus \( y(W, Q) \) is positive only if \( W \) is more twice as common in \( r(Q) \) as it is in the collection as a whole. If \( W \) appears in more than half of the indexed documents, then \( y(W, Q) = 0 \); e.g. the word "Wikipedia" in a collection of Wikipedia documents.

For any document \( D \):

Let \( l(D) \) be the number of words in \( D \).

Let \( c(W, D) \) be the number of occurrences of \( W \) in \( D \).

Let \( m(Q, D) \) be the number of occurrences of any query word in \( D \).

Let \( f(W, Q, D) \) be the number of occurrences of query words within 5 words on either side of any occurrence of \( W \). If the query word occurs within 5 words of two occurrences of \( W \) then that counts only once. For instance, suppose that the query \( Q \) is “Mount Vernon” and the only occurrence of either “Mount” or “Vernon” in the document is in the sentence “Martha Custis Washington lived at Mount Vernon with George Washington” Then:

\[
\begin{align*}
  f(“Martha”, Q, D) & = 1 \text{ (5 words from “Mount”, 6 words from “Vernon”) } \\
  f(“Custis”, Q, D) & = 2 \text{ (4 words from “Mount”, 5 words from “Vernon”) } \\
  f(“Washington”, Q, D) & = 2 \text{ (“Mount” and “Vernon” are both within 5 of some occurrence of “Washington”).}
\end{align*}
\]

Note that in all cases, \( q(W, Q, D) \leq m(Q, D) \); equality holds if every occurrence of every query word is within 5 words of \( W \).

The overall score of \( W \) in \( D \), \( s(W, Q, D) \) is equal to 0 if \( c(W, D) = 0 \) (that is, if \( W \) does not occur in \( D \)). Otherwise,

\[
s(W, Q, D) = \sqrt{\frac{f(W, Q, D)}{m(Q, D)}} + y(W, Q) \cdot \sqrt{\frac{c(W, D)}{l(D)}}
\]

The point is: We want to “reward” \( W \) for (a) being in a lot of documents in \( r(Q) \); (b) being significantly more common in documents in \( r(Q) \) than generally in documents in the index; (c) being mentioned numerous times in documents in \( r(Q) \); (d) being close to words of \( Q \) in documents in \( r(Q) \).

This formula accomplishes all that. The specifics of the formula are largely arbitrary. The use of the square root function means that these various considerations are weighted sublinearly; you have to get an increase of a factor of 4 in the underlying measurements to achieve an increase of a factor of 2 in that component of the score.

The total score for \( W \) is the sum of the scores of \( W \) in the documents that contain it.

\[
score(W, Q) = \sum_{D \in r(Q)} s(W, Q, D)
\]

**Experimentation**

Note that your index will have to contain considerably more than 10 documents for this to accomplish anything. I recommend a minimum of 100 documents. You should use the crawler from programming assignment 1 to create this index.

Run at least 3 experiments that “succeed” to the extent that the results after pseudo-relevance expansion are different from the initial results. Give a brief description of the experiments as follows:
What was the crawl that created the index?

For each experiment:

- What was the initial query?
- What was the expanded query?
- How different were the second set of results from the first set?
- Was the second set better, worse, or equal in quality?

Submission

Upload to the NYU Classes site:

- the source code;
- the index;
- an account of the experiments
- a README file.