Project (100 points)

Objective: Improve web search using relevance feedback.

Description: In assignment 3, you implemented a web search interface using API. Now you're asked to improve its relevancy performance using a simple relevance feedback technique. To do so, you should allow the user to specify 1 ~ 5 relevant search results for a query. After the user click the "re-rank" button, the search results should be re-ranked based on the user feedback.

You may use the following simple approach to implement relevance feedback: combine the titles and snippets of the user-specified results into one document. Compute the cosine similarity between this document and each search result (which can be represented as a document containing title and snippet). Re-rank the search results in decreasing order of the computed similarity values.

You should use the vector space model for such similarity computation. You are encouraged to perform linguistic preprocessing (e.g., case folding, porter's stemming algorithm, stop word removal) for the documents before similarity computation. Note that linguistic preprocessing is essential in almost any kind of document analysis because we almost always need to compute similarity between documents, and we want to capture semantic, instead of syntactic, similarity in this computation, where "Friend" from one document should match "friends" from another document. By removing stop words (especially in our case where the documents are very short), we make sure documents are similar only if they share meaningful (not stop words) in common. For tf-idf term weighting in the vector space model, as it's hard to get df information, you may assume idf = 1, i.e., only use tf in weighting terms.

Feel free to provide extra buttons to compare different implementations. For example, cosine similarity vs. Jaccard coefficient, between popular variants of the vector space model, with or without preprocessing …

Submission: Write a short summary in free style describing your implementation (platform, language, pre-processing, vector space model, similarity measure, experiments, etc), observations, and comments. Please also include the URL of your web interface so that the grader can test it out. Submit this summary together with all your source code in a single zip file to Tracs.

Optional Extension: If you're interested, you may extend the project by adding a Lucene-based local search functionality. With this extension, the project will have a weight of 40% (i.e., 40% homework, 40% project, and 20% exam).

Use the same search interface (and under the same URL), add a switch that allows the user to select web search or local search. For the local search functionality, install/configure Lucene and index a collection of documents of your own choice. You may search the Web for interesting document collections.

You may use Lucene, or another open source platform (such as Solr or Elasticsearch) of your choice.

Submission: Write a short summary in free style describing your implementation, document collection, example queries, observations, and comments. Submit this summary together with all your source code in a single zip file to Tracs.