CS574 Language Technologies
Fall 2002
Assignment 2

Due at the beginning of class, on Monday, October 28

Goal for the assignment: to gain a bit of experience in the design, implementation, and evaluation of question-answering (QA) systems.

As in assignment 1, this assignment is fairly open-ended. You are to implement a QA system that will operate in the standard TREC QA framework: the input to the system is a question, the output is a ranked list of five guesses for the answer. No human intervention is allowed in deriving answers.

For the assignment, we are providing a QA corpus that contains a set of questions and the expected answer(s) for each question. Since we can't make available to you the actual 9GB TREC collection used in the TREC QA studies, we will instead provide the top 20 documents retrieved by the Smart IR system (from a similarly large text collection) for each question in the corpus. Answers to each question are to be extracted from these 20 documents. Note that it is possible for some questions that none of the 20 retrieved documents contains the answer.

As noted above, the assignment is completely open-ended: you are free to build whatever components you'd like to include in your QA system and are free to use any publicly available software that you wish. You can even share components that you build with others in the class. The primary caveats are that your system cannot use the answers provided and must make clear in the write-up what components you used that you did not write yourself.

Assume that your system has entered the 50-byte (short answer) QA track so all answers should be 10 or fewer words in length. In addition, the output for each question should be the following:
question# document-id answer-text(for top-ranked guess)
question# document-id answer-text(for second guess)
question# document-id answer-text(for third guess)
question# document-id answer-text(for fourth guess)
question# document-id answer-text(for fifth guess)

The document-id refers to the document where the answer string was found. Use "nil" as the answer-text if your system finds no answer for a particular question.

What we will provide

- **questions.txt**: the questions. Feel free to change the format of this file if it makes automatic processing of the questions easier. Alternatively, you can use the questions as they appear in the "answers" file described below. In either case, you will need to keep around the question number to include as part of the answers file that your system produces.
- **answers.txt**: all answers found by TREC assessors for each question. The format of this file should be pretty clear. For each question, the file contains: (1) one line with the question number, (2) one line with the question, (3) list of document id's followed by answer strings, one per line, (4) a blank line separates the information for each question. Feel free to modify the format of this file if it's easier for your system to process.

- **top 20 documents retrieved for each question:** A gzipped-tar file with the top 20 documents retrieved for each question by Smart can be downloaded from `ftp.cs.cornell.edu/pub/cardie/top20.tar.gz`. (WinZip should open this file as well.)

Suggestions for how to proceed

- Start simple!! Select some really really dumb strategy to produce answers for each question just to make sure that you will have something to evaluate and to turn in. Only after you can do that should you proceed to something more sophisticated.
- It's fine to try a strategy very different from anything discussed in class. It's even fine if the system that you produce does terribly in terms of performance. You just need to be able to argue (in your write-up) why the strategy that you investigated MIGHT have worked.
- One possibility is to try using Lemur to implement a passage retrieval strategy for question answering. Another is to instead focus on one type of question, e.g. "who" questions, and develop a strategy specifically for that question type.

What to turn in

1. A description of your QA system. Enough detail should be provided so that, in theory at least, I could re-implement it. The description should explain each component in your QA system, the steps that your system takes to answer a question, any additional on-line sources of information used by the system, etc. Make clear which components of the system you built yourself vs. downloaded from elsewhere vs. got from another student in the course.
2. The output file of answers produced by your system for the questions from the development corpus that we provided. The answers should be in the format described above.
3. An evaluation (e.g. using the mean reciprocal rank evaluation measure) and analysis of your system's performance on the questions from the development corpus provided. How well did the system work? What worked? What didn't work? Can you say anything about which component is strongest/weakest?
4. A detailed walk-through of what your system did to handle one question (any one) in the corpus.
5. The output from your system for the question selected in (4) above. Enough information should be included in the output to convince me that the system is following the steps described in (1). It is not necessary to submit your code, but I may ask to see it in cases where the system description is unclear.