CSC 483/583: Programming Project (200 pts)

Building (a part of) Watson

Due before 11:59 P.M., December 5

For this project you must submit:

- An archive containing all the source code.
- Binaries/jars (if a language that requires compilation is used) and/or instructions how to run the code.
- A PDF document that must contain at least:
  - Description of the command line to run the python code, with an example.
  - Description of the code. You don't have to describe every function implemented. But you should describe the main part of the code and indicate where each question is addressed.
  - Results, i.e., the output of your code, for all the questions that required programming.
  - Answers for all questions that do not require programming.

Answers are always graded by inspecting both code and documentation. Missing code yields no credit. Missing documentation yields partial credit (if code exists and produces correct results).

Because the credit for graduate students adds to more than 200, graduate students' grades will be normalized at the end to be out of 200. For example, if a graduate student obtains 220 points on this project, her final grade will be $220 \times \frac{200}{250} = 176$. Undergraduate students do not have to solve the problems marked "grad students only." If they do, their grades will not be normalized but will be capped at 200. For example, if an undergraduate student obtains 210 points on this project (by getting some credit on the "grad students only" problem), her final grade will be 200.

IBM's Watson is a Question Answering (QA) system that "can compete at the human champion level in real time on the TV quiz show, Jeopardy." This, as we will see in class, is a complex undertaking. However, the answers to many of the Jeopardy questions are actually titles of Wikipedia pages. For example, the answer to the clue "This woman who won consecutive heptathlons at the Olympics went to UCLA on a basketball scholarship" is "Jackie Joyner-Kersee", who has a Wikipedia page with the same title: `http://en.wikipedia.org/wiki/Jackie_Joyner-Kersee`. In these situations, the task reduces to the classification of Wikipedia pages, that is, finding which page is the most likely answer to the given clue. This is the focus of this project.

In this project you will use the following data (see D2L project folder):

- 100 questions from previous Jeopardy games, whose answers appear as Wikipedia pages. The questions are listed in a single file, with 4 lines per question, in the following format: `CATEGORY CLUE ANSWER NEWLINE`. For example:

```
NEWSPAPERS
The dominant paper in our nation's capital, it's among the top 10 U.S. papers in circulation
The Washington Post
```

- A collection of approximately 280,000 Wikipedia pages, which include the correct answers for the above 100 questions. The pages are stored in 80 files (thus each file contains several thousand pages). Each page starts with its title, encased in double square brackets. For example, BBC's page starts with "[[BBC]]".

Your project should address the following points:

1) (100 pts) Indexing and retrieval: Index the Wikipedia collection with a state of the art Information Retrieval (IR) system such as Lucene (`http://lucene.apache.org/`) or Terrier (`http://terrier.org/`). Make sure that each Wikipedia page appears as a separate document in the index (rather than creating a document from each of the 80 files). Describe how you prepared the terms for indexing (stemming, lemmatization, stop words, etc.). What issues specific to Wikipedia content did you discover, and how did you address them? Implement the retrieval component, which takes as query the Jeopardy clue and returns the title of the Wikipedia page that is most similar. Describe how you built the query from the clue. For example, are you using all the words in the clue or a subset? If the latter, what is the best algorithm for selecting the subset of words from the clue? Are you using the category of the question?

2) (25 pts) Measuring performance: Measure the performance of your Jeopardy system, using one of the metrics discussed in class, e.g., precision at 1 (P@1), normalized discounted cumulative gain (NDCG), or mean reciprocal rank (MRR). Note: not all the above metrics are relevant here! Justify your choice, and then report performance using the metric of your choice.

3) (25 pts) Changing the scoring function: Replace the scoring function in your system with another. For example, by default Lucene uses a probabilistic scoring function (BM25). You can replace this default choice with cosine similarity based on *tf.idf* weighting. How does this change impact the performance of your system?

4) (50 pts) Error analysis: Perform an error analysis of your best system. How many questions were answered correctly/incorrectly? Why do you think the correct questions can be answered by such a simple system? What problems do you observe for the questions answered incorrectly? Try to group the errors into a few classes and discuss them.

Lastly, what is the impact of stemming and lemmatization on your system? That is, what is your best configuration: (a) no stemming or lemmatization; (b) stemming, or (c) lemmatization? Why?

5) (50 pts) Improving retrieval (GRAD STUDENTS ONLY): Improve the above standard IR system using natural language processing and/or machine learning. For this task you have more freedom in choosing a solution and I encourage you to use your imagination. For example, you could implement a positional index instead of a bag of words; you could use a parser to extract syntactic dependencies and index these dependencies rather than (or in addition to) words; you could use supervised learning to implement a reranking system that re-orders the top 10 (say) pages returned by the original IR system (hopefully bringing the correct answer to the top).