CSC 206: NLP
Spring 2015

Name (Print): _____

Problem Set 5: Information Retrieval

Due: Monday, May 4 2015, midnight

Please create a folder for this problem set, and save all files you need to submit in this folder. Zip up the folder and submit the .zip file on Nexus.

Questions:

1. Thinking about *precision* and *recall.* [From *Language and Computers*, Chapter 4, Exercise 5, p. 121]

   For the following scenarios, describe whether precision or recall is more important and why.

   1. Identifying cases where a cancer-curing drug has a side effect of nausea.
   2. Identifying cases where a cancer-curing drug has a side effect of death.
   3. Identifying case running red lights.
   4. Identifying and removing weeds that look like (desired) native flowers in your garden.

2. Improving the document retrieval system.

   How could you improve tf-idf based document retrieval system you worked with in class? The improvement can be in form of better (more relevant) search results or in the form of faster answers to your queries.

   Here are some suggestions of what you could try, but you are free to come up with your own ideas:

   - Exclude stop words (from the inverted index and the query).
   - Use a stemmer to reduce words to their stems (e.g., houses -¿ house) before adding them to the inverted index or computing the tfidf-score for query words.
   - Do more pre-computation in order to improve the efficiency. For example, store the normalized term frequency instead of the raw word count in the inverted index. Create two separate indices: the inverted index to store the normalized tf scores, and a second index to store the pre-computed idf scores for each word.
   - Expand the query to include synonyms of query keywords. (NLTK provides synonyms through the WordNet corpus. See chapter 2 of the NLTK book.)
   - Research *cosine similarity*, implement it and use to determine the relevance of each document.
   - Research bi-grams and create an inverted index of bigrams so that you can take into account not just individual words but also pairs of words.

Pick two ways of improving your IR system. Implement them each in a separate file so that you should end up with three versions of the tfidf retrieval program:

1. the original version you worked with in class
2. the original version modified to include your first improvement (Save this in a file called `tfidf-first.py`.)
3. the original version modified to include your second improvement (Save this in a file called `tfidf-second.py`.)

From a user's perspective running your retrieval programs should be the same for all three versions. That is, I should be able to use all three versions by first running the command

```
index = create_idf_index (wiki)
```

to create the index (or both indices, in case you decided to precompute both the tf and the idf values for all words). Once the index is created I should be able to query the document collection like this:

```
pretty_print_results (retrieve("zebras food", wiki, index), wiki)
```

(In both commands `wiki` refers to a document collection.)

3. Evaluate your improvements.

   Now evaluate your improvements by comparing the precision of their results and their efficiency to the original version.

   For this evaluation, you should create a set of at least 10 test queries. Try to make the queries diverse by including queries that ask about different things, include more or less common words, have different lengths, ... (You should hand in your list of test queries together with the analysis of your results.)

   Comparing efficiency. You can use a Python module called `timeit` to measure how long the retrieval function takes. One way to use it is as follows:

```
start=timeit.default_timer()
results=retrieve("zebra food", wiki, index)
stop = timeit.default_timer()
print stop-start
```

   This times how long the query *zebra food* takes and prints out the result in seconds. If you want to use it from the shell, you can do the following:

```
>>> start=timeit.default_timer(); results=retrieve("zebra food", wiki, index); stop =
timeit.default_timer();
>>> print stop-start
```

Comparing precision. Save the output that each system gives you for each of the test queries in a file. (Hand that in as well together with your analysis.) Use this output to evaluate the relevance of each document, and then calculate the *Mean Average Precision* for each system. How you do these calculations is up to you. If you want to write some Python code to help you with, that's great, but if you prefer to use a spreadsheet or do the calculations by hand, that is find, too.

Write a report on your improvements and the evaluation. This report should contain the following information

- a description of each improvement
- your prediction on how the improvement will impact the efficiency and accuracy of the retrieval system with an explanation of why you think so
- a detailed description of the evaluation results (e.g. the running times for all queries and systems as well as the average running time for each system, the average precision for each query and system as well as the mean average precision for each system)
- an analysis of what these results mean in terms of whether the improvement attempts were successful
- a discussion of whether the evaluation results confirmed your prediction or not (If not, do you have an explanation why not?)
- the list of test queries you used

What to Submit:

- a pdf document containing the answer to Question 1
- a second pdf document containing your report on Questions 2 and 3 (as described in Question 3)
- the Python files `tfidf-first.py` and `tfidf-second.py`
- three text files (one for each system) containing the search results for your test queries