Assignment 2

Information Retrieval and Web Search
Winter 2016
Total points: 80
Issued: 01/29/2016 Due: 02/12/2016

All the code has to be your own (exceptions to this rule are specifically noted below). The code must run on the CAEN environment without additional installation or additional files (except for the data files specified in the assignment).

You can discuss the assignment with others, but the code is to be written individually. You are to abide by the University of Michigan/Engineering honor code; violations will be reported to the Honor Council.

[80 points] Vector-space model.

Write a Python program that implements the vector-space model. You will test this program on the Cranfield dataset, which is a standard Information Retrieval text collection, consisting of 1400 documents from the aerodynamics field. The dataset is available from the class web page:
http://web.eecs.umich.edu/~mihalcea/courses/EECS486/Resources/cranfield.zip

Several queries and relevance judgments associated with these queries are also provided from the class web page:
- queries:
http://web.eecs.umich.edu/~mihalcea/courses/EECS486/Resources/cranfield.queries.test
- relevance judgments:
http://web.eecs.umich.edu/~mihalcea/courses/EECS486/Resources/cranfield.reljudge.test

Programming guidelines:
Write a program called *vectorspace.py* that indexes the collection and returns a ranked list of documents for each query in a list of queries. The program will receive three arguments on the command line (in this order):
- one argument indicating the weighting schemes to be used for the documents. For the purpose of this assignment, your program should be able to "understand" at least two values for the term weighting schemes specified in this argument: tfidf, [your-own-weighting scheme].
- a second argument indicating the weighting schemes to be used for the query. For the purpose of this assignment, your program should be able to "understand" at least two values for the term weighting schemes specified in this argument: tfidf, [your-own-weighting-scheme].
- a third argument indicating the name of the folder containing the collection of documents to be indexed. For testing purposes, use the *cranfieldDocs/* folder that you will obtain after unpacking the cranfield.zip archive.

- a fourth argument indicating the name of the file with the test queries. For testing purposes, use the *cranfield.queries.test* query file that you will get from the class webpage.

Include the following functions in *vectorspace.py*:
a. Function that adds a document to the inverted index:
Name: *indexDocument*; input: the content of the document (string); input: weighting scheme for documents (string); input: weighting scheme for query (input); input/output: inverted index (your choice of data structure)

Given the name of a file, this function will:
- preprocess the content provided as input, i.e., apply removeSGML, tokenizeText, removeStopwords, stemWords. You are encouraged to use the functions you implemented for Assignment 1.
- add the tokens to the inverted index provided as input and calculate the numbers necessary to calculate the weights for the given weighting schemes. Note: the inverted index will be updated with the tokens from the document being processed.

b. Function that retrieves information from the index for a given query.
Name: *retrieveDocuments*; input: query (string); input: inverted index (your choice of data structure); input: weighting scheme for documents (string); input: weighting scheme for query (input); output: ids for relevant documents, along with similarity scores (dictionary)

Given a query and an inverted index, this function will:
- preprocess the query, i.e., removeSGML, tokenizeText, removeStopwords, stemWords. You are encouraged to use the functions you implemented for Assignment 1.
- determine the set of documents from the inverted index that include at least one token from the query.
- calculate the similarity between the query and each of the documents in this set, using the given weighting schemes to calculate the document and the query term weights.

The main program should perform the following sequence of steps:
i. open the folder containing the data collection, provided as the third argument on the command line (e.g., *cranfieldDocs/),* and read one file at a time from this folder.
ii. for each file, obtain the content of the file, and add it to the index with *indexDocument*
iii. if necessary for the term weighting schemes, calculate and store the length of each document
iii. open the file with queries, provided as the fourth argument on the command line (e.g., cranfield.queries.test), and read one query at a time from this file (each line is a query)
iv. for each query, find the list of documents that are relevant, along with their similarity scores.

The *vectorspace.py* program should be run using a command like this:
```
% python vectorspace.py  tfidf tfidf cranfieldDocs/ cranfield.queries
```

It should produce a list consisting of pairs of query ids, along with the ids of the documents that are relevant and their similarity score (for each query, list in reverse order of similarity score):

```
queryId1 documentId1 similarityScore1
queryId1 documentId2 similarityScore2
....
queryId1 documentIdx similarityScorex
queryId2 documentId1 similarityScore1
....
....
```

Write-up guidelines:

Use the cranfield.reljudge.test file as a gold standard to measure the performance of your system. Use the following two weighting scheme combinations: (1) tfidf for documents, tfidf for query; (2) your choice of weighting scheme for documents and your choice of weighting scheme for query.

Create a file called answers.txt. Include in answers.txt a description of the weighting schemes you used.

For each of the two weighting scheme combinations, calculate and include the macro-averaged precision and recall when you use:
- top 10 documents in the ranking (for each query)
- top 50 documents in the ranking (for each query)
- top 100 documents in the ranking (for each query)
- top 500 documents in the ranking (for each query)

Which weighting scheme provides better results? Include a one paragraph discussion of how the two weighting schemes compare to one another.

For the definition of precision, recall, and macro-averaging, please refer to lecture slides.

General submission instructions:

- Include all the files for this assignment in a folder called *[your-uniqname].Assignment2/* **Do not** include the data folder, i.e., *cranfieldDocs/*
  For instance, lahiri.Assignment2/ will contain answers.txt, vectorspace.py
- Archive the folder using tgz or zip and submit on Canvas by the due date.
- Make sure you include your name and uniqname in each program and in the answer.txt file
- Make sure all your programs run correctly on the CAEN machines.