Homework Number 4

Due March 6, 2019

1. Do an Ad Hoc Information Retrieval task using TF-IDF weights and cosine similarity scores. Vectors should be based on all the words in the query, after removing the members of a list of stop words.
    1. Download (and unpack) the zip file: Cranfield_collection_HW.zip . It contains the following documents:
        1. the Cranfield collection --- also avaialble from http://ir.dcs.gla.ac.uk/resources/test_collections/cran/. This includes a readme that describes the data, but further details are provided here:
            1. cran.qry -- contains a set of 225 queries numbered 001 through 365, but referred to in cranqrel below as 1 through 225 -- this is an important detail which, if missed, can make debugging confusing.
                - Lines beginning with .I are ids for the queries (001 to 365)
                - Lines following .W are the queries
            2. cran.all.1400 -- contains 1400 abstracts of aerodynamics journal articles (the document collection)
                - Lines beginning with .I are ids for the abstracts
                - Lines following .T are titles
                - Lines following .A are authors
                - Lines following .B are some sort of bibliographic notation
                - Lines following .W are the abstracts
            3. cranqrel is an answer key. Each line consists of three numbers separated by a space
                - the first number is the query id (1 through 225) --- convert 001 to 365 by position: 001 --> 1, 002 --> 2, 004 --> 3, ... 365 --> 225
                - the second number is the abstract id (1 through 1400)
                - the third number is a number (-1,1,2,3 or 4)
                    - These numbers represent how related the query is to the given abstract
                    - Unrelated query/abstract pairs are not listed at all (they would get a score of 5): There are

        1836 lines in cranqrel. If all combinations were listed, there would be 225 * 1400 = 315,000 lines.
        - We will treat -1 as being the same as 1. We suspect it means something like "the best choice for the query", but the specs don't say.
    2. A stoplist (currently written in python) called stop_list.py -- you can use this list to eliminate words that you should not bother including in your vectors
    3. A scoring script (cranfield_score.py) that you can use to score your results.
    4. A sample output file (sample_cranfield_output.txt). You can use this as a guide to the format of your output and also to test the scorer. It was created by randomly changing the answer key. You should not seriously compare it to your output file.
2. For each query, create a feature vector representing the words in the query:
    1. Calculate IDF scores for each word in the collection of queries (after removing stop words, punctuation and numbers)
    2. Count the number of instances of each non-stop-word in each query
    3. The vector lists the TF-IDF scores for the words in the vector
3. Compute the IDF scores for each word in the collection of abstracts, after removing stop words, punctuation and numbers.
4. Count the number of instances of each non-stop-word in each abstract
5. For each query
    1. For each abstract,
        1. Create a vector representing scores for words in the query, based on their TF-IDF values in the abstract, e.g., if the only words contained in both query and abstract are "chicken" and "fish", then values in the vector representing these words would have non-zero values and the other values in the vector would be zero. The vector would be the same length as the query's vector.
        2. Calculate the cosine similarity between vectors for query and abstract
    2. Sort the abstracts by cosine similarity scores. So for each query there should be a ranking of all 1400 documents (see the sample output file which contains 225*1400=315,000 lines)
6. For tokenization, you can use nltk if you want, but if you use some other system of tokenization, please indicate what rules you use.

    7. Your final output should look similar to cranqrel: the first column should be the query number, the second column should be the abstract number, and the third column should be the cosine similarity score. For each query, list the abstracts in order from highest scoring to lowest scoring, based on cosine-similarity. Note that your third column will be a different sort of score than the answer key score -- that is OK. Example line:

       1 304 0.273

2. Systems will be evaluated by the metric: Mean Average Precision based on the precision of your system at each 10% recall level from 10 to 100%.
    1. For each query, establish 10 cutoffs based on recall: 10%, 20%, ... 100%
    2. Average the precision of these 10 cutoffs.
    3. Average these precision scores across all queries, ignoring a query if the system gets a recall score of 0 or if there are no matching abstracts.
3. To score your system, use the cranfield_score.py script (an implementation of MAP)
    1. To run the script from the command line type the following (cranqrel is the answer key)
        1. python cranfield_score.py cranqrel your_outputfile
        2. For example, if you type: python cranfield_score.py cranqrel sample_cranfield_output.txt
            1. You will get the following output:
                - Average MAP is: 0.362325230638
            2. Typical MAP scores are actually lower than this sample. About 20% is normal for this task (although higher scores are possible). Ultimate grades will be based on the results of the class. A general rule of thumb for determining if your system is working is: if your MAP score is in double digits, it is probably working OK. If you are getting a score of less than 3% MAP, there is probably something wrong. [Most likely, it is a format error.]
        3. The scoring program has an optional feature that prints out the scores for each query (to help with debugging), simply add an additional argument True (setting trace to True), e.g.,

            - python cranfield_score.py cranqrel your_outputfile True

5. Possible extensions to make the system work better
    1. Add additional features: incorporate stemming (e.g., treat plural/singular forms as single terms)

    2. Determine if there is a correlation between the TF-IDF based ranks and the relevance score ranks and find a way to predict the rankings, or to simplify the rankings and predict the simplified rankings.
    3. We initially assume a simple method of counting term frequency: a straight count of the number of times a term occurs in a document. However, in practice, there is usually some adjustment for the length of the document, for example:
        1. Divide this count by the number of words in the document
        2. Use the logarithm of the term frequency
        3. Use 1 if the term exists in the document and 0 if it does not
        4. Combine some of the above methods together
        5. Try more than one of these methods and determine which method achieves the highest scores.
    4. Other interesting extensions.
6. Grading is based on your final MAP score and any extensions that you made (see 5).
7. Submit the following to GradeScope in a zip file called NetID-HW4.zip, e.g., alm4-HW4.zip. The zip file should contain:

    - Your source code
    - Instructions for running your system (NetID_README_HW4.txt)
    - Your output file (**output.txt**) in the format described above. It should look very similar to cranqrel and you should make sure that the scoring program works on it. The main differences are:
        - there should be 225*1400=315,000 lines in the file (for all query/summary combinations)
        - your cosine similarity score should be the third column instead of a -1 to 4 ranking