Create a system for an Ad Hoc Information Retrieval task using TF-IDF weights and cosine similarity scores.

1. Read the Cranfield collection file description. The Cranfield collection --- also available 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:
    a. 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
    b. 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
    c. 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. Please see how they are defined in “cranqrel.readme”.
            - 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.
            - 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.

2. Read other file descriptions:
    a. 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
    b. A scoring script (cranfield_score.py) that you can use to score your results.
    c. 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.

3. Create a system for an Ad Hoc Information Retrieval task using TF-IDF weights and cosine similarity scores. Please look at lecture notes for TF-IDF information. Vectors should be based on all the words in the query and abstract, after removing the members of a list of stop words. Create the system in the following steps:
    a. For each query, create a feature vector representing the words:
        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
    b. Compute the IDF scores for each word in the collection of abstracts, after removing stop words, punctuation and numbers.
    c. Count the number of instances of each non-stop-word in each abstract.
    d. For each query: 1. For each abstract, ○ 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.
        - Calculate the cosine similarity between vectors for query and abstract.
        2. Sort the abstracts by cosine similarity scores from high to low. 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)

    e. Your final output should look similar to sample_cranfield_output.txt: 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 cosinesimilarity. 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
4. 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%.
    a. For each query, establish 10 cutoffs based on recall: 10%, 20%, ... 100%
    b. Average the precision of these 10 cutoffs.
    c. 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.

5. To score your system, use the cranfield_score.py script (an implementation of MAP)

a. To run the script from the command line type the following (cranqrel is the answer key): python3 cranfield_score.py cranqrel your_outputfile

b. For example, if you type:

   python3 cranfield_score.py cranqrel sample_cranfield_output.txt

   You will get the following output:

   Average MAP is: 0.3616260725637992

c. Typical MAP scores are actually lower than this sample. About 15%-25% is normal for this task (although higher scores are possible). 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.]

d. 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

6. Possible extensions to make the system work better:
   a. Add additional features: incorporate stemming (e.g., treat plural/singular forms as single terms) and lemmatization.
   b. We initially assume a simple method of TF-IDF. However, in practice, there is usually some adjustment for the length of the document, for example:
      - Divide this count by the number of words in the document;
      - Use 1 if the term exists in the document and 0 if it does not;
      - Adjust weights for tf counts or idf counts;
   c. Add word embedding similarities.

7. Please note that:
   a. You may use any python library or functions.
   b. You may do any type of extensions to enhance your results, but we require you to include the TF-IDF feature. You will get 0 for this lab, if you do not use the TFIDF feature. Please briefly explain all extensions you used in the report.
   c. **You are NOT allowed to cheat by using any information in cranqrel or sample_cranfield_output.txt, or by changing the scoring program. You should NOT modify cranfield_score.py.**

d. You should indicate the MAP score that you got in the report. We will rerun your code to see if your score can be replicated.