CS 662
Artificial Intelligence Programming
Homework #5
Using Information Retrieval to Classify Text
Due: Thursday, 10/31, 10am

Summary

In this assignment you'll be using some basic information retrieval methods to categorize and classify Usenet postings using Python. We'll be working with the 20 Newsgroups dataset.
note: Please create a directory/folder called **hw5** and place all your assignment's code in there. Also, you **must** provide a README that clearly explains how to use your code. Please do not expect me to go to great lengths to figure it out. Create a zip file from your hw5 folder. Do **not** use the .rar format: zip is available on the cslabs machines.

1. **(5%)**To begin, you'll build an **article** Class that stores the category (newsgroup name) of an article, a string containing its raw text, and a dictionary that maps words (except stopwords and non-words) to their TFIDF score. (You may add other member variables to the Class if you like.) The TFIDF calculation itself is built up in the components below. You can treat any alphanumeric string as a word (\w* as a regular expression) and everything else as a non-word; however, as in HW1, you should first remove punctuation and convert all words to lowercase. Feel free to extend this to hyphens within words, etc., but document your choices in the README.

2. **(10%)** Next, implement a method called **computeDocumentFrequency**. You should be able to give it a directory as an argument and have it return a dictionary holding each word that occurs in any article (as keys) and its inverse document frequency. (again, leave out stopwords and non-words.) Use this method to build an IDF dictionary for each newsgroup.

3. **(20%)** Now, implement the **computeTFIDF** method within the article class. It should use the IDF dictionary for the appropriate category to compute a TFIDF score for each word in an article.

4. **(10%)** Use this to build a method called **computeTFIDFCategory**. It should be able to take as input a directory, find the IDF dictionary for that directory, and use that to construct a dictionary of the 1000 highest-scoring words and their TFIDF scores summed over all articles in that category. You will want to pickle these dictionaries.

5. **(20%)** Next, write a function called **cosineSimilarity**. It should take as input two TFIDF dictionaries and return the cosine of the angle between their vectors.

6. **(20%)** We are now ready to build our first classifier. Write a method called **classify** that takes as input an article, computes its cosine similarity with each

of the 20 newsgroup category vectors, and selects the most similar one.
To test your classifier, write a separate program called **tester.** It should take one command-line argument, which is the number of articles to try to classify. It should then choose this many articles at random, call *classify*, on them, and print out the fraction of articles correctly classified. (This is assuming that the articles are chosen from the different 20 newsgroups folders randomly)

7. **(15%)** Last, we'll use your TFIDF category vectors to perform a task called *hierarchical clustering*. This will create a tree showing the relative similarity of different categories; elements closer to each other in the tree are deemed more similar.
   Write a method called **hCluster** that performs the following algorithm:
   1. Let S be the set of all category TFIDF dictionaries.
   2. While |S| > 1 do
      1. Find the two most similar elements *e1* and *e2* in S using cosine similarity
      2. Replace them in S with {e1 U e2} to create a parent of these elements.

   Compute the clustering (dendrogram) corresponding to the 20 newsgroup data set and write a method that displays it. This can be as simple as a list of the clusters (see below), or an “ascii art” drawing that outputs individual categories at the top and the root node of all categories at the bottom (see below), or something fancier if you wish.

   Say we have categories a-f and we’ve grouped a,b; c,d; e,f, and then ab U cd, then abcd U ef. You could just print out “{ { { a U b } U { c U d } } U { e U f } } }”.
   Or:
   Sample dendogram picture (where pq means S contains {a U b}) of the same grouping:

```
a b  c d  e f
| |  | |  | |
ab   cd   ef
 |    |    |
 abcd     ef
   |       |
    abcdef
```