HW2
Due 2/23

In this assignment you will write a program that converts a document collection to the bag-of-words model (also known as the Vector Space Model), and then answer similarity search queries.

In the bag-of-words model, each document is represented as a vector. Each dimension of the vector corresponds to one word in the vocabulary, and its value is the number of times the corresponding word occurs in the document. Consequently, given a collection of *n* documents, we can construct a *m-by-n* term-to-document matrix *X*, where *m* is the number of unique terms in the text collection, *n* is the number of documents, and each element *X(i, j)* is the number of times that the i^{th} word occurring in the j^{th} document. Note that there are other, more sophisticated frequency weighting schemes, but we will use the raw frequency counts for this assignment for simplicity.

Each document in the dataset has a class label (there are 8 classes total). You should store the class labels separately.

You can use any programming language, provided that the program can be run on zeus. Provide a README file with instructions on how to run your program. If your program requires special instructions or software packages, you can make an appointment with the TA to demo your program. More details:

Step 1: Write a program doc2mat that reads in an input text file (attached), and converts the documents in that file to a *m-by-n* matrix as described above. You will need to use some data structure (e.g. hash table) to efficiently store and locate the words. The class labels should not be counted as words in the matrix. Keep the class labels elsewhere for the next step.

Input: Filename of a text file containing a set of documents, with one document per line. Each line starts with a class label for the document, followed by a Tab, and then a sequence of words that appear in the document, delimited by spaces. The documents have already been pre-processed for you – they have been "stemmed", and stop words have been removed. What this means is that all words are reduced to their "root" form (so we don't need to worry about singular/plural, past tense, present tense, etc.), and that common words that don't add any information (like "I", "the", "you") are removed from the documents.

As an example, here is one of the documents in the file:

earn champion product approv stock split champion product inc board director approv two for stock split common share for sharehold record april compani board vote recommend sharehold annual meet april increas author capit stock mln mln share reuter

"earn" is the class label. The rest of the line contains the stemmed words in the actual document.

You may need to use a special sparse matrix format to store the large matrix. Check out the Compressed Column Storage (CCS) format here: http://netlib.org/linalg/html_templates/node92.html. Note that each column in your matrix is a document.

Output: Matrix (can be saved to a file)

Step 2: Write another program that performs similarity search (nearest neighbor search) for document data. Given a term-document matrix *X* and the ID of a query document *Q* (Q ⊆ X), your program should find the document in *X* that is the most similar to *Q*. Use the matrix that you created from the previous step as input. The ID of the query document should range from 1 to n. For example, if the ID of the query document is 10, that means the 10^{th} document in the input file from Step 1 (the 10^{th} column in the matrix). Your program will search in the matrix and find the most similar document to document #10, using (1) cosine similarity, and (2) Euclidean distance, respectively. Report the class label(s) of the results.

Obviously, you want to exclude the query document from the nearest neighbor search, otherwise it'd always be returned as the most similar document.

As an example, your output from one query might look like this, for cosine similarity:

Query	Nearest neighbor	cosine similarity	Query class	Nearest neighbor class
10	233	0.789	1	1

Repeat the experiments for the first 10 documents in each class, for each of cosine similarity and Euclidean distance. You should have 10*8*2 = 160 experiments. Does cosine similarity or Euclidean distance give you more accurate result? To answer this, you would need to compute the accuracy for each measure. Recall that accuracy is (# of correct labels / total # of instances). Report the accuracy per class, as well as the overall accuracy for all classes. You should have two tables that look like the following, one for cosine similarity, and one for Euclidean distance.

For extra credit, use the whole dataset for querying, one document at a time (you'd be doing 1-nearest-neighbor classification, using leave-one-out cross validation). Report the accuracies the same way you did for the above.

Cosine similarity (one for Euclidean Distance as well)

Class #	Class name	# correct	# incorrect	Accuracy
1	acq			
2	crude			
3	earn			
4	grain			
5	interest			
6	money-fx			
7	ship			
8	Trade			
Overall				

Submission: Please submit your source code, output (sample screenshots of program execution), report, and README electronically on Blackboard. Bring a hard copy of the report to class.