Problem Set 7: Due April 26

For this assignment, you'll implement one or more machine learning algorithms to process text data. The companion handout "NaiveBayes for Document Classification" gives some additional details about algorithm details that you should read before beginning work. The assignment will only sketch out high level requirements.

1. (10 points) Build a python script `classify.py` that can be run on the command line and takes three arguments as input: (1) a file of training data; (2) a file of testing data; and (3) a 'function' to execute. When run with an incorrect number of arguments, the script should print out help that indicates the names of allowable functions and what they do. Below I've listed the required functions. In general, all functions should process in training data to build some representation of the target concept and then (possibly) test themselves on the testing data. You can assume that the training data is organized in a single file with 1 example per line. 'Words' are separated by spaces and the first word on each line corresponds to the class label. For this assignment you can assume a binary classification problem (two possible class labels).

2. (15 points) Add a function `tf` that uses the training data to determine *term frequencies* for each word in the training set. Specifically, this function should indicate for any given class label, how many times that word has been observed (across all examples with the given label). `tf` should build term frequencies for all words, and (1) save them to a file called tf.csv where each line in the csv contains a word and the corresponding term frequencies in each class; and (2) you should print the terms with the top 5 term frequencies in each class (so 10 words should be printed total along with their frequencies).

3. (10 points) Given your term frequency data, make a function `tfgrep` that classifies a document based on the *most discriminating term*. Assume $tf_{i,0}$ is the term frequency for word i in class 0. The most discriminating term is the term i such that $\forall j \quad |tf_{j,0} - tf_{j,1}| \leq |tf_{i,0} - tf_{i,1}|$. When the script is run with the `tfgrep` function, the scripts should output: (1) what term it is using to perform the classification, and two confusion matrices: the first indicating performance on the training data (the data used to compute the tf tables) and the second indicating performance on the test data.

4. (5 points) Make a function `priors` that computes the class priors for the training set and then classifies a document using 0-R (i.e., by simply guessing the majority class from the training data). As before, the output of this function should be two confusion matrices, one showing results on the training data, the other showing results on the testing data.

5. (25 points) Make a function `mnb` that implements the multinomial naive bayes model to predict the most likely class. This function should make use of your term frequency

counts from before, but should also perform Laplacian smoothing to allow test documents to contain words that don't appear in the training set (for a given class). Specifically, with smoothing, the probability of seeing a word w in class c should be computed as $(1 + \mathit{tf}_{w,c})/(1 + \sum_v \mathit{tf}_{v,c})$. Thus, $P(w|c)$ will not be 0 or 1 for any word. As before the output from this script should be two confusion matrices one for the training data and one for the testing data.

6. (15 points) Make a function `df` that computes the document frequency for each term in the training set. Specifically, instead of counting how often any given word occurs in the class, here, you should compute the number of training examples (from a specific class) that have one or more occurrences of any given term. As with `tf`, output a csv file `df.csv` that indicates the probability that a document from class c has the term w, so values in this csv file should be document frequencies divided by the class prior. Additionally, print the top 5 probabilities for each class to stdout.

7. (25 points) Make a function `nb` that implements the multivariate Bernoulli model to predict the most likely class. This function should make use of your document frequency counts from before, but should also perform Laplacian smoothing to allow test documents to contain words that don't appear in the training set (for a given class). Specifically, with smoothing, the probability of seeing a word w in class c should be computed as $(1 + \mathit{df}_{w,c})/(2 + |c|)$ where $|c|$ represents the number of documents from class c in the training set. Don't forget that for this model, you need to deal explicitly with $P(w|c)$ and $P(\neg w|c)$. Check the handout for more details. As before the output from this script should be two confusion matrices one for the training data and one for the testing data.

8. (15 points) Make a function `mine` that modifies one of the classifiers (either `mnb` or `nb`) to use a reduced dictionary of terms. Specifically, try to limit the "complexity" of the model by throwing out some of the words that actually occured in the training set. Note; this may mean you'll need to recalculate df or tf values depending on how you represent things internally. Provide a blurb of text with the output of your function that describes how you've changed the default implementation and why you've made your choice. You will be graded here on correctness of your approach given your stated intention, and on the merits of your idea. You should aim for an idea that seems as though it could improve performance of the system; 5 bonus points if it actually does improve the results.

9. (?? points) For upto 20 additional points of extra credit, implement either a perceptron model of learning, or DNMB as described in the handout.