CS1114 Assignment 6

out: Friday, April 20, 2012.
due: **Friday, April 27, 2012, 5PM.**

In this assignment, you will implement an authorship detector which, when given a large sample size of text to train on, can then guess the author of an unknown text.

You can find stub files in the directory `/courses/cs1114/student_files/A6/`. There is also training and testing data in the `TrainingData` and `TestData` subdirectories.

1 Detecting authorship

The algorithm you will implement works based on the following idea: An author's writing style can be defined quantitatively by looking at the words he or she uses, and the order in which the words appear. Some authors use certain words—and certain pairs of words—more or less often than other authors.

To make things significantly simpler, we're going to assume that the author always follows a given word—"the", for instance—with the same distribution of other words. Of course, this isn't true in reality, since the words one chooses when writing depends on context. Nevertheless, this simplifying assumption should hold over a large amount of text, where context becomes less relevant.

In order to implement this model of an author's writing style, we will use a Markov chain to model text as a sequence of words. As we learned in lecture, a Markov chain is a set of states—in this case, words—with the Markov property—that is, the probability of a word appearing at a given position depends only on the previous word.

As we saw, a Markov chain can be represented as a directed graph, weighted by probabilities. Each node is a state (again, words, in our case), and a directed edge going from state S_i to S_j represents the probability we will go to S_j next when we're currently at S_i. We will implement this directed graph as a transition matrix. Given a set of possible words $\{W_1, W_2, ...W_n\}$, you will construct an $n \times n$ transition matrix P, where P_{ij} is the probability of a transition from word W_i to word W_j.

The edges, in this case, represent the probability that word j follows word i from the given author. Among other things, this means that the sum of the weights of all outgoing edges from each word must add up to 1.

For each author, we can construct a transition matrix from a large sample text corpus of that author's writing (for instance, one or more novels from that author). Our next step is to determine the author of an unknown, short chunk of text. To do this, we simply compute the probability of this unknown text occurring, using the words in that order, from each of our Markov chains. We will then attribute the text to the the author with the highest likelihood.

You will implement the Markov chain model of writing style. We have given you some sample texts from a number of authors for you to train your model on, as well as some test data that you can use to evaluate your code.

2 Constructing the transition matrix

The first step is to construct the transition matrix representing our Markov chain. First, you must read the text from a sample file. To do so, you can use the Matlab command `fopen` to open a file (e.g., `TrainingData/Austen/all.txt`) and `textscan` to read the text into a cell array (making sure to `fclose` the file once you're done reading it).

```
f = fopen('filename.txt', 'r'); % open the file for reading
cell = textscan(f, '%s');       % read each space-delimited word into a cell array
fclose(f);                      % close the file
D = cell{1};                    % the array we're interested in is actually the
                                %   first element of 'cell'
```

The cell array `D` above will contain all the space-delimited words read by `textscan` as strings. (Note that the `textscan` function isn't smart enough to split out punctuation or deal with capitalization. These are things you must do yourself if you so choose.)

Once you have the cell array of words, you must count the number of unique words appearing in that particular text corpus. This is necessary for knowing the size of the transition matrix. You can do so with the use of a hash table, as we covered in section. Recall that you can create a Java hash table using the following Matlab code:

```
hashtable = java.util.Hashtable;
```

Recall that an entry into a hash table has a *key* and a *value*—the key is used as the index into the hash table, and the value is what gets stored there (for this class, we are not worried about how a hash table is implemented, although this is a very interesting topic in itself). With hash tables in Matlab (which, confusingly, are really Java hash tables, but this is also a topic for another day), you can add an entry using the `put` function, and retreive an entry with the `get` function. Getting a key which hasn't been previously added returns an empty matrix:

```
hashtable = java.util.Hashtable;
hashtable.put('llama', 10);
hashtable.get('llama');    % Returns 10
hashtable.get('platypus'); % Returns []
```

For this assignment, you will probably find it very useful to create two hash tables: one that maps each word (encoded as a string) to a location (row/column index) in your transition matrix (we will call this the *dictionary* for the transition matrix), and the other that does the inverse, storing a mapping from each word's location to the word itself.

Once this is done, you can construct a transition matrix from the words in your cell array. You will want to create a sparse array using the Matlab `sparse` function, otherwise you will very likely run out of memory. Along with the transition matrix, you will be creating a corresponding *histogram* vector that contains word frequencies (normalized by the total number of words in the document (including repeated words)). For instance, if 5% of the words in the document are `and`, the entry of this vector corresponding to the word `and` would be 0.05.

You will write this in a function called `build_transition_matrix`, which takes in a filename, and returns the transition matrix, the dictionary, the inverse dictionary, and the histogram of word frequencies.

$\Longrightarrow$ Write the function `build_transition_matrix.m`.

3 Calculating likelihood

Once you have your transition matrix, you can calculate the likelihood of an unknown sample of text. We have included for you several pieces of literature by various authors, as well as excerpts from each of the authors. Your goal is to identify the authors of each excerpt.

To do so, you will need to calculate the likelihood of the excerpt occurring in each author's transition matrix. Recall that each edge in the directed graph that the transition matrix represents is the probability that the author follows a word with another. You can load the text of the excerpt into Matlab using `fopen` and `textscan` in the same way as before.

Since you will be multiplying numerous possibly small probabilities together, your calculated likelihood will likely be extremely small. Thus, you should compare $\log(\text{likelihood})$ instead. Keep in mind the possibility that the author may have used a word he has never used before. Your calculated likelihood should not eliminate an author completely because of this. Similarly, as discussed in lecture, you should handle transitions that were unobserved in the training data in a graceful manner.

$\Longrightarrow$ Write the function `compute_text_likelihood.m`.

5 Finding the author with the maximum likelihood

Now that you can compute likelihoods, the next step is to write a routine that takes a set of transition matrices and dictionaries (represented as cell arrays), and a sequence of text (again, represented as a cell array), and returns the index of the transition matrix that results in the highest likelihood. You will write this in a function `classify_text`, which takes a cell array `tmatrices`, a cell array `dictionaries`, a cell array `histograms`, and the name of the file containing the test text, and returns a single integer `best_index`. This will allow you to test your code using our routines—we will be using this to test your code on new authors.

$\Longrightarrow$ Write the function `classify_text`.

6 Testing your authorship prediction

Now that you can calculate the likelihood that a text is written by an author, take a look at the texts we have provided you. In the folder `TrainingData`, you will find literature by several authors, taken from Project Gutenberg. In the folder `TestData`, you will find five unknown excerpts from each of the authors. Create a test function which creates author models using the training data, and predicts the authors of the test data. Note that in addition to particular works, in each author's directory there is a file `all.txt` that contains all of the words in a single file.

$\Longrightarrow$ Write a series of tests that adequately demonstrates the operation of your code. Also document any tests you have done, and the results you got. This series of tests should also include a function for *summarizing* the training data, which prints, for each author, the number of unique words used by that author, the most common words used by that author, and the most common words that follow a few common words, such as "`the`" and "`and`". Note any interesting statistics you find.

7 Challenges

Once you have finished this, you may be interested in a harder challenge. Shortly after the founding of the United States, various founding fathers collectively published a series of 85 essays called the Federalist Papers under the pseudonym "Publius." It is generally accepted that the papers were written by Alexander Hamilton, John Jay, and James Madison. While historians have agreed on who wrote most essays, there are a few whose authorship is known only via statistical analysis, and we would like to corroborate this analysis with evidence from your own analysis. In the folder `TrainingData`, you will find sample writing from Hamilton, Jay, and Madison. In the folder `TestData`, you will find the first 20 Federalist Papers. Can you identify the authors of each? It may be possible that two people collaborated on a paper. Can you tell? You will get one extra credit point for writing code that tests on this data set.[1]

There are many possible ways to improve on the method described above. You will get an additional extra credit point for implementing an improved method for predicting authorship. This is completely open-ended—you are free to brainstorm your own method, but you must explain what you did to get the extra credit.

Finally, you will be turning in your code for this assignment, and we will be testing your code on a completely different data set (which you won't have access too). The code that performs the best in our authorship tests will be awarded at least two extra credit points (note that you will likely need to write an improved algorithm to get better results than the baseline method described in Sections 1-6).

One additional item of extra credit is to write a function that takes your computed statistical models, and *generates* more text from a given author. The more plausible the text you can generate, the more extra credit you will receive.

[1]See `http://en.wikipedia.org/wiki/Federalist_papers#Disputed_essays` for more information.