Homework 5

Software Design **Allen B. Downey**
Spring 2006

Due: Wednesday 1 March

The reading for this assignment is Chapters 11–12 of *How to think...*.

5.1 Markov Analysis

The goal of this homework is to develop a program that reads a text, performs Markov analysis, and generates a new, random text that is statistically similar to the original.

For example, my version of this program read Doctorow's *Someone Comes to Town, Someone Leaves Town* and generated the following text:

> He drove. "For real, he could give her a junky tool kit with cracked handles and chipped tips, and I trust my intuition to take the subway all night, watched the brother and sister stare at one another by means of some strange pollen carried on the sofa for a while, trying to work in this cave? They'd carved his cradle. Fed him. Taught him to move someone with a toe, smoothing over the scar on her gentlest spin cycle, but still Davey cried. Their mother rocked and gurgled and rushed, and then so did Kurt. "Okay, it works," Lyman said.

Which makes about as much sense as the original.

Here's how it works. The simplest form of analysis you can do is to record each word in the document and the number of times it appears. Then you can generate random words so that the proportion of each word in the generated text is the same as the proportion in the original text. Of course, the results are unlikely to make much sense.

The next step is to use a bit of context to model the correlations in the original text; that is, the probability that a given word will follow another. You can do that by keeping track, for each word, of all the words that might follow it (and their proportions). If you do that, you get text that seems a little more sensible, but not much.

So the next step is to generalize the previous technique so that the prior context can be more than one word. For example, you could keep track of all the two-word pairs that appear in the text, and for each two-word prefix, you could keep track of all the words that might follow (and their proportions).

Then, to generate random text, you can:

1. Start with any prefix.
2. Find the words that might follow the prefix and choose one.
3. Form a new prefix by removing the first word from the old prefix and adding the new word to the end.
4. Go to step 2.

Your job is to write a program that will read and analyze a given text, and then generate a few hundred words of random text with the same correlation structure as the original. If your program is fully general, you should be able to adjust the number of words in the prefix, which will adjust the amount of context that is used to choose each word.

The first challenge of this assignment is to figure out where to start, and to plan a development strategy. Here are some ideas about how to get started:

- Choose a text to work with (maybe from `gutenberg.net`, but also check out more recent books under a Creative Commons license at `http://creativecommons.org/text/`) and make a cleaned-up version that doesn't have the header (you know, the information you are not supposed to remove).
- Analysis of the full text might be slow, and generate more information than you can print and understand, so you might want to make a small file that contains, for example, just one chapter.
- We have worked with several programs that read through a file and break it into words. You might want to start with one of them and modify it.
- Count the words in the document and the number of times each one appears (pretty much the same as the previous assignment).
- Now think of ways to generate new words so that they have the same probabilities as in the original text. You might want to check out the `random` module at `http://www.python.org/doc/current/lib/module-random.html`. If you have not already bookmarked the Python documentation, you should!
- Next you might want to build a dictionary with each possible two-word prefix as a key, and a list of possible successors as the value.
- You might want to write code to print the contents of the dictionary in a more readable form. This might help with any debugging you have to do.
- Once you are convinced that the analysis phase is working, you can work on the generating phase.
- You might want to make your program take command-line arguments to specify the file to analyze and the number of words to generate.
- Finally, you can generalize your code to work with prefixes of any length. How does the random text change as the prefix length increases?

As always, you should start with a working program and try to make small, incremental changes. Make sure that each change produces some visible output so that you can confirm that it is doing what you think. In many cases, you will find that your changes work the first time, but if not, you will be able to find the error quickly.