Jordan Van Duyne

Software Design Mini Project 3: Text Mining and Analysis

Project Overview

I used pattern's ability to download text from a webpage to obtain .txt files of books from Project Gutenberg. After obtaining the text for *The Adventures of Huckleberry Finn* by Mark Twain and *Great Expectations* by Charles Dickens, my program reads through the texts, performing Markov analyses, and creates a new, random text that is a combination of the results of the two analyses. I hoped to learn more about manipulating iterables and about reading and writing files. I created a new .txt file of a mash-up of the original texts that, due to its randomness, is quite comical.

Implementation

In my program, I have one global variable: a dictionary that maps from all "prefixes" to all possible "suffixes" for a given prefix; this dictionary's keys are every set of two words that appear in the analyzed texts and each value is a list of the next word after a prefix for every appearance of that prefix. My program reads through all text files that are provided as arguments, and for each file it performs a Markov analysis and adds to this global dictionary. I decided to make this dictionary a single global dictionary, instead of creating a new dictionary for each input text, because the two texts are likely to have some of the same "prefixes", so in order to create a truly random mash-up, the suffixes for each prefix that are to be used in the creation of the final text need to be from both texts, not from two separate dictionaries that each only contain words from their respective original texts. Furthermore, I decided to keep punctuation and capitalization as a part of word in the prefixes and suffixes to retain some sense of grammatical correctness in the final text.

In order to create the dictionary, I had to create a way to check every two words (every prefix) in a text file and the next word (the suffix for each prefix). Since the prefixes are the keys in the dictionary, they had to be of an immutable type, ruling out lists and leaving strings and tuples. In order to get through every possible prefix and suffix combination, my program saves the first two words of a file as a prefix, and takes the third word as the suffix and saves them to the dictionary. As it loops through every word in the file, it takes the next word as the new suffix, uses the previous word as the second word in the new prefix, and the previous second word as the prefix as the first word of the new prefix. As a result, each word in the variable containing the current prefix has to be easily accessible, which is more an attribute of a tuple of words rather than a string of words, resulting in the prefixes being tuples. On the other hand, the values in the dictionary have to be able to be modified (if a certain prefix appears more than once, that prefix maps to multiple suffixes), so lists are the optimal option for the value type since they are mutable.

After the dictionary is created, my program randomly selects a key (prefix) from the dictionary, writes it to a new file, writes a random corresponding suffix to the new file, uses the previous two words (the second word in the prefix and the word that is the suffix) as the next prefix, writes a random corresponding suffix to the file, and then repeats the last two steps until the file contains as many words as was specified by the arguments.

Results

My program outputs a file that contains a somewhat coherent combination of text from the input texts. The output file is “somewhat coherent” because, for the most part, it is grammatically correct since certain words tend to come after only certain other words (for example, prefixes ending in a possessive pronoun probably map to suffixes that are either adjectives or nouns, while prefixed ending in adjectives probably map to suffixes that nouns) and since punctuation is kept (meaning that words beginning with capital letters should come after periods), but the context of each sentence is completely missing due to the randomness of the words.

Although my program can be called using any number of text files greater than or equal to one for different ranges in the syntax of the output file, I decided to run my program using *The Adventures of Huckleberry Finn* and *Great Expectations* to create a combination that I like to call *The Expectations of Great Adventures.* A few excerpts from this thrilling novel include:

> "Well, then, how's he going to snake him out and shook me by the king he gets here." Well, the old man said he would a thought he lived in for a dog." "So 'd I. Well, it would take a man's clothes. So I says: "Why, dear me, I do." "My George! It's the best girl I ever see. Then the preacher he begged him not to be pretty dull. Buck and a half. Then I remembered. The river looked miles and miles across. The moon was so glad to git himself up for another spring. I laid there, and he warn't in no condition for me to save her life; and I reckon so, 'm. I don't want to know. And then I slipped down the river road as hard as I was fidgety. Miss Watson would take care of 'em in the snow.
>
> If you notice, most folks don't go about women in that way without anybody to mind about that way every night grieving. I got an old tin lantern, and judged that that was dead, I do. Dey's awluz at it, and they warn't real kings and dukes? It wouldn't strain myself if I was up in no time, but to be left free to work a camp-meeting with.
>
> It's the most noise, and slipped in behind the front door, nor a wooden one with a spelling-book. She worked me middling hard lot I was, and never thought about a quarter of a skull and crossbones on the plate and throwed it out in skiffs and trying to get the bed when you come to time. When you got to do it. Her face would give the pen and wrote. So then we went aboard, and Jim don't, either, I reckon. He told them he hugged and kissed as many years as we went by so thick that the talk got further and further off, and know it when he was mighty downhearted; so I took it.

Reflection

I believe that my process went well. I started with a function that could do a small, simple task, checking if it worked as expected, and then expanding that function by a small step to do something more towards my final goal and checking that addition, and then expanding and checking the function once more. For example, I began by inputting a small text file into a function that looped through and printed every word in the file to check if my looping worked. From there, I expanded my function to print every "prefix" as a tuple, and then, once my function had evolved to deal with the dictionary, I had the function print every key in the dictionary and then every value in the dictionary. However, I could probably improve my program by figuring out some way to possibly include a doctest. I had fun with my project due to the nonsensicalness of the output and believe that it was appropriately scoped; I usually have problems with dealing with strings, other iterables, and dealing with files, so being able to deal with something that I did not feel too comfortable with in a safe environment was a great experience. I can take my knowledge of string and tuple manipulation and file writing in future projects. I am also glad that I learned how to take text from a website. However, a skill that would have been nice to have had going into this project is more knowledge of tuple manipulation syntax.