David Papp

Text Mining Project

Project Overview

My goal was to do Markov text synthesis not only by words as the problem suggested, but by various degrees, and compare how accurate each degree is. I used Pride and Prejudice because it's sufficiently long and thus the Markov chains were interesting enough. I did not care so much for generating interesting text as I did for learning about how the number of degrees affect the accuracy of the text synthesizer.

Implementation

The first step is to read in a file with PRAW and store it in a pickle file. In all future runs, the data is read in from the pickle file rather than from the original online source in order to speed up the program. The string is then stripped of all punctuation such as commas, periods, and line breaks.

Next, we iterate through each word of the stored string and store each unique word as a key in a dictionary. The next word is then stored in another dictionary within the previous word's key (so a nested dictionary), except this time the word key maps to the number of times that word occurred after the previous one. Once the string is iterated through, we create a copy of the dictionary in which the keys map to the probabilities of each word occurring after a previous word, rather than the number of occurrences. Note that I also created a function that doesn't look at full words, but rather a specified length of characters, to generate the next character. This often produces non-existent words but nevertheless produces interesting results.

Now that we have our markov values, we first generate a starting word. We then generate a random number between 0-1 and choose the next word by using the previous word as a key and finding the word within the key's dictionary that corresponds to our random number. This is then repeated the number of times as specified by the user.

Results

Generating 100 characters at different degrees: (ignore first words)

Degree: 2
obleat precriver the fore diculd come of ithe whommot ad hin herre the els of ther fresis my once ity
Degree: 3
gh fort of they a dance and invictly perfor was posed now whith othe civingley were that their londolen
Degree: 4
m equated fashion his emotive hereven on poor familiest enjoymention of sat had with lord and the back
Degree: 5
ant as would such a mode of the society as think i have been minuteness of her direction unanswer she was
Degree: 6
th he behaviour hoping appear which so many worked our sisters were it will that expressing her singing m
Degree: 10
led before mrs gardiner that he had done and anxious lizzy said he one day your sister must make them his prin
Degree: 15
hs mind and she would have felt almost secure and with reason for charlotte had been tolerably encouraging he was c
Degree: 50
 me pain to meet him but i have no reason for avoiding him but what i might proclaim before all the world a sense of very great illusage and most pain

Generating 50 words:

allowed to those strong effort to those strong effort for none of particular to throw off in requiring an instance which convinced that lay oxford blenheim warwick kenilworth birmingham etc etc and made himself were affected your own doubts fearing lest they knew would advise you belong to prevail on how

With character degree based Markov text synthesis, a minimum of degree 5 is required in order for most generated words to be actual English words. After 5 degrees, there seem to be marginal diminishing returns. At very high degrees, it is essentially reciting the text.
With word based Markov text synthesis, there seems to be little grammatical structure of coherence. In fact, it does not seem any more coherent than a generating text by the 6th degree. An interesting pattern that is apparent is in the middle of the second line, where a number of towns are listed. These words appear in the text so few times that they were basically guaranteed to follow each other. This raises an interesting question: would a longer or shorter text generate more meaningful text?

Reflection

I found the programming part fairly straightforward but also valuable. It actually raised a few interesting questions for me: how could we improve the code such that it looks at a variable number of words before generating the next one? For example, if the last word is a conjunction, it will be much less accurate in predicting what the next word should be than if it was a noun or a verb. The project was appropriately scoped and documented. I could have done a better job creating unit tests, but since many of my functions relied on random variables, it would have been futile for some of them.