Mini-Project 3: Text Mining

Emily Yeh

Project Overview

In this project, I analyzed text that I mined from Project Gutenberg's The Complete Works of William Shakespeare. Before working on this project, I completed the Word Frequency toolbox exercise. This exercise inspired me to try to make something related to the frequency of words in Shakespeare's texts. On a tangentially related note, I've also always liked dogs, memes, and the doge internet meme. So what happens when you mix Shakespeare with a popular internet meme? That's what I hoped to demonstrate in this project.

Implementation

First, I needed to get a list of all the words used in Shakespeare's texts. I used my work from the toolbox exercise to strip punctuation and white space from the text, then stored the resulting words in a list. Then, I wrote a function to get the top n words, storing the words as a dictionary whose keys are the words and whose values are the number of times those words occur in the text. Finally, I created a skeleton for the poem, retrieving words for the poem by referring to them by their indices in the word list.

At first, I wanted to make it possible to take the number of stanzas as an input and generate a poem with the corresponding number of stanzas by using a `for` loop. However, since I was retrieving words from the word list using their indices, I realized that I had no idea how to randomize the indices for every iteration of the `for` loop. As a result, I eventually decided just to hard-code the skeleton of the poem.

To make my poem a little fancier, I also added a function that finds words that rhyme with "wow" by generating a list of words that end in –ow. What I did not foresee is that some words that end in –ow don't actually rhyme with "wow" (in fact, a majority of the words in Shakespeare's texts that did end in –ow did not rhyme with "wow"). Doge poetry isn't supposed to be perfect, though, so I decided not to worry about it. (It's a feature, not a bug!)

Results

In the end, I created a fairly entertaining poem generator. It's far from perfect, but I think that it's still fun to use! The skeleton that I cobbled together for the poem systematically retrieves words from the list of most commonly used words by referring to them by their indices in the list. I found that in general, the words which had indices of 120 or greater were mostly nouns, while the words which had indices of between 0 and 120 were mostly personal pronouns, articles, and other extremely common words. For the purposes of this explanation, let N denote the words which had indices of 120 or greater and let X denote the words which had indices of between 0 and 120. For the fun of it, I threw in some prepositions (which I will denote as P) and words that rhyme with "wow" (which I will denote as W).

Therefore, I structured the poem like this:

```
[X] [N], [X] [N], [X] [N]

[P] [N], [N], [N]

[X] [N], [X] [N], [X] [N]

much [N], very [N], wow
```

For reference, here are the top 100 most common words. (Note how they are mostly extremely commonplace words, although they do not perfectly follow the patterns that I observed.)

```
['the', 'and', 'i', 'to', 'of', 'a', 'you', 'my', 'in',
'that', 'is', 'not', 'with', 'me', 'it', 'for', 'his', 'be',
'your', 'this', 'but', 'he', 'have', 'as', 'thou', 'him',
'so', 'will', 'what', 'thy', 'all', 'her', 'no', 'do', 'by',
'shall', 'if', 'are', 'we', 'thee', 'our', 'lord', 'on',
'king', 'good', 'now', 'sir', 'from', 'o', 'come', 'at',
'they', 'well', 'or', 'which', 'would', 'more', 'was',
'then', 'she', 'am', 'how', 'here', 'let', 'enter', 'their',
'love', 'when', 'them', 'ill', 'hath', 'than', 'man', 'an',
'there', 'one', 'go', 'upon', 'like', 'say', 'know', 'may',
'make', 'did', 'us', 'were', 'should', 'yet', 'must', 'why',
'see', 'had', 'tis', 'such', 'out', 'some', 'give', 'these',
'where', 'too']
```

Here are some examples of poems that I generated. They don't always make sense and they don't always sound that great, but I hope you'll find that they're still pretty entertaining to read!

```
mini-project 3: doge writes poetry!

title: the maw's wailing skies

the sinister, the prophetic, the rescud
at desirous, net, narrow
should interpret, were buckled, us drowned
much daws, very grease, wow

and hoop, and reprieve, and thrusts
after appeared, anchor, willow
did ix, make cools, may theatre
so arbour, such venison, wow

that skip, that string, that olive
across agony, murthers, borrow
know checkd, say bespeak, like bathe
much seconds, VERY broker, wow
```

```
mini-project 3: doge writes poetry!

title: the concealed's revolve melts

the petter, the disdaining, the restless
of brightest, whetstone, allow
tis hired, had strut, see feelt
much unfolded, very confusions, wow

and bringst, and ordered, and venerable
past e, denyt, crow
why allowed, must pulling, yet masham
so usurpers, such mordake, wow

that gallery, that multiplying, that marchioness
beyond strumpets, smoking, snow
should cometh, were distant, us bottoms
much rote, VERY hurly, wow
```

```
mini-project 3: doge writes poetry!

title: the bravd's mistresses raised

the deliberate, the glittering, the retort
beneath bane, ducat, shallow
up deepest, duke agincourt, th headlong
much digestion, very watchmen, wow

and dispositions, and ruinous, and cart
under stained, enchanting, follow
first ruler, mine loo, speak complexions
so syria, such swallowing, wow

that usual, that unwillingly, that puritan
upon lacking, wingd, fellow
most professd, take berries, who quell
much lordly, VERY playfellow, wow
```

Reflection

I had a lot of fun working on this project! It was super exciting to figure out how to use Python to find the most frequently used words in a text, and even more exciting to figure out how to take this process one step further and generate my own poetry. In the future, I think it'd be pretty cool to learn how to use Python to generate poems that actually make sense—or even build my own sonnets based on the stresses and numbers of syllables in words. (This was my original plan for this project, but I quickly realized that counting syllables and stresses per word is a lot more complicated than I had thought, since there are so many different cases.) In general, I'm pretty satisfied with my work for this project. Going forward, I hope to use what I learned in completing this project to make even cooler text-mining projects happen in the future!