**Project 3: Generative Text (Documentation)**

The generated poems and their titles are in the “my\_poems.txt” file.

**Background:**

The only coding class I have successfully taken taught me how to use natural language processing toolkits in python for low-level, textual analysis. So, my background in computationally manipulating text is mostly based in analysis methods as opposed to synthesis methods. However, I wanted to try my hand at using some of these NLP toolkits to synthesize text—as such, I used python for this project.

All the ideas below use a corpus of all of Ben Lerner’s poetry (3 poetry books: Angle of Yaw, Lichtenberg Figure, and Mean Free Path), which I gathered myself by de-DRMing kindle files of his three books (using Calibre).

Idea 1 (unsuccessful): I wanted to first generate a madlibs-like template using an open-source, machine learning algorithm (char\_rnn), which I have already implemented for a previous project. To do this, I thought that I could simply Part-of-Speech (POS)-tag each of Lerner’s poems, and then feed only the POS tag (formatted like the original poem) into the neural network. This idea didn’t get very far, as I soon realized that I didn’t know how to modify the neural network to “see” the POS tags *as* POS tags instead of as individual characters composing a word. Thus, I moved onto Idea 2.

Idea 2 (also unsuccessful): Instead of using a neural net, I thought I would try writing my own Markov chain in python to generate text from all of his poems. I read all the documentation you provided us to make Markov chain models, but I wasn’t able to get keep track of all the words following each word in my corpus (to define the pseudo-statistics of my model):

I was able to write a script that returned a dictionary with keys (as all the words in my corpus), and values (a list) (I had no idea you could do that with python dictionaries). However, I wasn’t able to add more words to that list (which would contain the words following the key-word). At this point, time was running out, so I had to move on to Idea 3.

Idea 3 (success!): Since I was really set on using a Markov chain, I searched around the internet and found Markovify, an extensible, Markov-chain generator for python (maybe not only python? I can’t tell). I imported it into my working environment and read through the documentation to implement it, and it worked! I then used a more “advanced” implementation to generate the main body of my text: instead of using only one Markov model trained on either: all of his poems or just the poems from one book, I used the markovify.combine method to create a combined model of three Markov models, each trained on only one of his books. I then specified the “weight” they each had in the combined model. In terms of composition and structure, the poems in Angle of Yaw stray the most, so I gave the model trained on those poems a lower “weight.” (for more details see the .md file with my code). Since I didn’t code my own Markov chain, I thought I would at least create titles for each of my generated poems. To do this, I used NLTK’s BiGramCollocateFinder, which generated a list of pretty evocative bigrams. I then used this list to create titles for the poems, which I wrote to a file before each new poem was generate by the combine Markovify models.

**Reflection**:

I would ideally like to host this Lerner-like poem generator online, but I wasn’t sure how to implement a .json file with the combined Markov model generated by Markovify. There is a method that allows you to export any generated model, but when I looked at the .json file, it contained (numerical) statistical data that I wasn’t sure how to access in javascript (the examples we had in class relied on statistical information presented as duplicates of certain data)

Despite the fact that I borrowed a lot of code to do the grunt work, I learned a lot about the limits of my coding knowledge (something that isn’t always apparent to me when I first conceive of an idea). Honestly, I proud that I even got Markovify to work, since I haven’t really had much experience implementing other people’s scripts.