Logbook for Poem Generator

Egon Janssen (s1530100); Unmukt Deswal (s2310171) 04/11/2020

Implementation & Motivation

This is a logbook of our implementation where we describe a step-wise account of the tasks that we implemented as a part of this project. The first step of the process was to be able to create an Inspiration Set for our poem generator. We explored a bunch of resources for our Inspiration Set and finally settled on 'Project Gutenberg corpus'. The Project Gutenberg electronic text archive was famously the first free online book provider, which contains about 60,000 free electronic books. However, our motivation behind selecting this set was simple, as it comprised of a huge dataset which accounts for big sample space for experimentation. We also noted that it was possible to select a single author's work and use feed in our Inspiration Set. This gave us a sense of assurance, that consistency in writing style was somewhat retained. We weren't entirely sure about what to expect when we started with this project, and thus, a simple implementation of a text generator with a good Inspiration Set was the target. For this, we decided on implementing n-grams for our poem generator as it is the most popular way of sensible text generation. We experimented with these n-grams, but bi-grams and tri-grams gave the best results in terms of making coherent and sensible lines. Thus, we decided to use bi-grams and refer to them as couplets for our implementation, as they made most sense for poems which weren't particularly meant to be verbose. In order to do so we defined a context dictionary in the setupModel() function, which returns our Inspiration Set in the form of a dictionary. This dictionary comprises of the frequency of occurrence a particular n-gram in the data. In order to update our language model, we supply each individual sentence from a book, and update the dictionaries with information corresponding to our n-grams by using TF-IDF. However, as a result of successful implementation of this simple poem generator using bi-grams, we were challenged to think out of the box to add complexity and nuance. So, we decided to implement rhyme patterns in our generator to resemble true poetry. This was done by calling for the findRhyme() function that found rhyming words from the Inspiring Set. Finally, we added Capitalisation and use of characters such as the semi-colon(;) and the comma(,). All this collectively brings some structure to the generated poems. We discuss such strengths and also a few weaknesses of our generator in the concluding section.

Logs

The following is a simple log of the step-wise sequence of tasks which were carried out as a part of the overall implementation.

Date	Time	Person	Remark
28/10/2020	2 hrs	Egon	Researched resources and techniques to generate poems. Finalizing NLTK library as source.
29/10/2020	1 hr	Egon	Implemented n-grams of NLTK corpora from Gutenberg Selections corpus as our Inspiration Set.
29/10/2020	2 hrs	Egon	Implemented a line generator for our Inspiration set.
29/10/2020	3 hrs	Egon	Implemented rhyming a couplet(Bi-gram). Adding a rhythm of two rhyming lines to this couplet.
29/10/2020	1 hr	Egon	Added a title generator for poems.
29/10/2020	1.5 hrs	Egon	Initialized the 'Little book of generated poems' as an html webpage.
30/10/2020	2 hrs	Unmukt	Got up-to-speed with the code progress and implementation.
30/10/2020	0.5 hr	Egon	Re-structured and concreted the project.
30/10/2020	0.5 hr	Egon	Setup the README file.
30/10/2020	1 hr	Egon	Converted the python files into executables.
30/10/2020	1 hr	Egon	Cleaned up the code.
02/11/2020	1.5 hr	Egon	Update the output html.
03/11/2020	4 hrs	Unmukt	Setup Logbook.
03/11/2020	0.5 hr	Unmukt	Update and polish README file.

Discussion & Conclusions

We believe that this poem generator is a serious contender in the race to podium for poem generators (of course the ones made by the participants of this course). We believe our poem generator takes into account a few things that helps it qualify as a good one. Firstly, this implementation uses bi-grams in the basic text generation process, which ensures that sensible texts are produced. Secondly, the ability of the generator to use two rhymes in a rhythmic fashion adds some kind of a poetic character to the generated texts. Thirdly, the consideration of special characters and capitalisation makes it unique and add to the overall presentation of our poems. However, we cannot say that it could ever be match up to a human in this regard. Despite having a sense of context and having a good Inspiration Set, it must not be considered as truly capable of producing a creative outcome. This is to say that even though we could classify this poem generator as adopting a 'Creation' based generation odyssey, there is still much debatable about what how much this perception truly describes creativity in poetry.