DATS6312 Group Project: Rap Lyric Generation

Jesse Borg, Alex Cohen, Sarah Gates, Jonathan Giguere
Spring 2020

Section 1: Introduction

Rappers have certain sounds, styles, and vocabulary that they combine to make their art. You can hear a lyric, or read a verse, and you might be able to identify the artist who wrote it. Our project aims to emulate this "style" that a given rapper has. Our group plans to use a rapper's entire collection of songs as training data to train a model. This model will then take a word or a phrase as input from the user and generate new lyrics around this word or phrase in the style of the rapper whose songs were used as training data. Finally, to evaluate our model's performance, we will compare the generated lyrics for a given word or phrase and compare to the artist's actual lyrics for stylistic similarities.

Section 1.1: Desired Outcomes

For this project, we aim to develop a web scraper that can gather lyrics from a specified artist available on the internet and store them in a csv file. We will then tokenize the lyrics to make them easier to work with and so that the model can extract as much meaning as possible to generate lyrics accurate to the artists' style.

Once the lyrics are gathered and tokenized, they will be used to train a model which will take these lyrics and learn them. Once learned, the model will take a random string of the artist's actual lyrics and return a sequence of generated lyrics in the artists' style.

Finally, we will use similarity metrics to compare how accurate the model is when trying to emulate a specific artists' style. The newly generated lyrics will be compared against the scraped lyrics and a similarity score will be determined.

Section 1.2: Github Link

The code for our project can be found in this GitHub repository:
https://github.com/sarahjeangates/NLP-Project-Lyric-Generation

Section 2: Data

We developed a web scraper to gather lyrics data for a defined artist from Metrolyrics.com. As there are about 16,000 artists available on the website, this gives us access to over 1 million songs (with varying numbers of lyrics each). Drake was our primary artist of focus when developing our models, but this could also be expanded to other artists as well. This process will provide us with flexibility when choosing the artists for our training data.

Section 3: Model Building

This section will explain the process we carried out in getting our final product of generated lyrics. It will explain the step-by-step process on how we started with lyrics readily available online, to putting them in an easy to read form in which a model could learn the style of a chosen rapper and recreate it.

Section 3.1: Lyric Web Scraper

As mentioned in the Data section, the web scraper was developed using the urllib, re, beautiful soup, and unidecode libraries to scrape lyrics from up to 75 songs for a defined artist from Metrolyrics.com. The first step is to define an artist in the “lyric_scrape.py” code file. Then, simply run all markdown cells and a csv file containing the song, artist name, and lyrics will export. For example, if “Drake” is defined as the artist, the exported file will be called “drake_scraped_lyrics.csv”. This can then be plugged in to the next step in the pre-processing step.

Section 3.2: Lyric Tokenizer

For data pre-processing, the lyrics were tokenized and formed into one continuous corpus with all lowercase lettering. This made it more efficient to teach the model. We felt that capitalization and punctuation were not necessary to teach the model the style of a rapper. The tokenizer was kept as simple as possible in order to reserve time and computing power in teaching the model, which is the most important part of this project. With more time, the impact of using different tokenization methods, grammar and sentences could be evaluated.

Section 3.3: Lyric Generator

In order to construct the lyric generator, the pre-processed data needed to be first converted to a series of sequences, represented by a series of characters/words with one final item. This will be important during model training, as the models will be trained by taking a series of values as inputs, and a single value as the predicted output. The model will then be trained to estimate this new item based on a series of items passed as inputs. For example, the sentence: ‘He went to the store’ could be entered as the following (with spaces as underscores):

H-e-_-w-e-n-t-_-t-o-_-t-h-e-_-s-t-o-r-e
H-e-_-w-e-n-t-_-t-o-_-t-h-e-_-s-t-o-r-e
H-e-_-w-e-n-t-_-t-o- -t-h-e-_-s-t-o-r-e
H-e-_-w-e-n-t-_-t-o- -t-h-e-_-s-t-o-r-e
H-e-_-w-e-n-t-_-t-o- -t-h-e-_-s-t-o-r-e
H-e-_-w-e-n-t-_-t-o- -t-h-e-_-s-t-o-r-e
H-e-_-w-e-n-t-_-t-o- -t-h-e-_-s-t-o-r-e

This can be done on both a character by character level (as shown above), as well as a word by word level. The above assumes an input length of 13, which is a hyperparameter that can be tuned during training. The character-stride was 40 for the character model, and the word-stride was 15 for the word model.

The sequences of characters/words were decomposed in this manner, then converted to an index. This was done by taking all of the unique values across the corpus (either letters or words) and ordering them, then converting the item to its respective place in the order. For example, ordering characters in the character model could represent the letter 'A' as a 1 followed by 26 zeros (an extra character for spaces). This creates two dictionaries that each take the form of an embedding, converting a letter or word to a sparse vector of zeros with a single 1 in the column representing the item. This format is much easier for the model to use as an input when compared to a sequence of letters and numbers.

Two models were developed for this project: one used to estimate the next character in a sequence, and one to estimate the next word in the sequence. Both models were developed using the encoding strategy above, and trained using Keras. Keras was selected due to the ease of use, as well as the ease of implementing LSTM layers, which were the core mechanism used in both models. The model architecture is below:

Input → LSTM(128) → LSTM(64) → **ReLU(Dense(64)) → Dense(len(embedding)) → Softmax*

* The number of neurons in this layer was 128 for the word model
** Layer not included on the word model

Each model was trained for 15 epochs using categorical cross-entropy as the loss function. The character model was trained using Adam (with a learning rate of 0.01) as the optimizer and a batchsize of 512. The word model was trained with RMSProp (with a learning rate of 0.01) as the optimizer and a batchsize of 256. These differences were due to the differing number of input sequences that could be constructed given the corpus of lyrics. These models were used to generate new lyrics that emulate the style of the artist's lyrics used for training.

Section 4: Model Evaluation

For model evaluation, we focused on the character model. Two procedures were performed to evaluate the performance of this model: an edit distance calculation and a comparison of generated words to words the artist has used before.

Section 4.1: Edit Distance Metric

Edit distance is a measure of dissimilarity between two strings. It gives the minimum number of operations needed to make the strings identical. In our character model, if the artist's lyrics were learned perfectly, the edit distance value between the generated lyrics and the lyrics that

actually come after the sequence would be 0. A lower edit distance value is better as it indicates the model output is exactly the same as what the artist would say (the model learned the training data well).

When applying the metric to our model we generated new lyrics and calculated the edit distance for each line generated by comparing it to the characters that actually come next in the song. After calculating the edit distance values for many iterations, we got an average edit distance as the overall evaluation metric. For one test using Drake lyrics, the model generated 100 lines of new text with an average edit distance value of 148.56. The model's parameters were set to generate 200 new characters so we can conclude that the model is generating dissimilar output. For future tweaking of the model, we can use this value as a benchmark to evaluate the effectiveness of our changes.

Section 4.2: Artist Vocabulary Metric

Musical artist's have very different vocabularies. We want to be able to see if the words generated by the model are words that the artist has used before in his or her songs. The character model predicts one character at a time, so sometimes the words generated are not even of the English language. This metric attempts to capture what proportion of the generated words belong to the artist's overall vocabulary. We want to give the model a higher score if it uses the artist's vocabulary, and penalize it if it generates non-English words or words the artist has never said in any of his or her songs used for training.

To apply the metric to our character model, we generated 100 lines of new text, flattened the lines into a list of individual generated words, and extracted the unique ones. Once, we had the unique generated words, we did a similar procedure to get the artist's true vocabulary from the corpus of their songs. Having the unique generated words and the artist's unique words, we found the common terms and divided by the count of unique generated words. For one test, the model produced a value of 0.58 for this metric. This can be interpreted as, 58% of the words the model generated were words the artist would use in his or her songs. This score indicates the character model could improve and provides a baseline for additional experimentation.

Section 5: Outcome

This project has provided our group members with a solid introduction to natural language generation tasks. Although the model's performance could be improved, our team has accomplished the following: created a web-scraper that provides a basis for experimenting with lyrics from a multitude of artists, created word and character models with LSTM layers that effectively emulate a rapper's style to some degree, and created two model evaluation techniques for improving further experiments. We have gained a base knowledge of natural language generation and will be much better suited to tackle similar tasks in the future.

Section 6: Challenges and Further Work

One aspect of future work that could be undertaken is refinement of the character and word level models, further tuning the number of layers, layer sizes, and other model hyperparameters. Additionally, different training approaches could be experimented with if we had more time.

Aside from improving the models, we would also like to introduce more established evaluation metrics for natural language generation tasks like BLEU (Bilingual Evaluation Understudy) and ROUGE (Recall Oriented Understudy for Gisting Evaluation). These metrics are similar to the ones we have already used in that they compare model generated text to the actual reference text. The difference is, that they are more sophisticated with respect to recall and precision. Using these metrics would make our results easier to interpret for other natural language generation researchers.

In the future, our group would like to use lyrics from different rappers for training to see what makes a rapper easier or more difficult to emulate. We would also like to sample artists from a variety of genres to train our models. Perhaps a genre like pop, that is less lyrically complex would be an easier candidate for natural language generation. The research questions that could be explored are plentiful and thought provoking.

References

Kaggle Official Website Available at: https://www.kaggle.com/
(Accessed on March 10, 2020)

Metrolyrics Official Website Available at: https://www.metrolyrics.com/
(Accessed on March 11, 2020)