Rhyme Generator: using Markov and LSTM for creating a rhyming response to rap lyrics

Yuval Medina, Frank Li, Steven Zhang, Ruiyang Chen

Abstract

In Natural Language Processing, the distinct nature of lyrics is often overlooked. However, the significance of lyrics is highlighted by how they make songs more appealing, especially in rap songs, since the lyrics need to consider rhymes and rhythms. This paper proposes a system to generate rhymed words for incomplete rap lyrics. CMU Pronouncing Dictionary and pos tags from NLTK are used for handling the dataset from the MLDB dataset and Poetry Foundation. Then, we propose a new method of generating the lyrics combining Markov and LSTM models in training and predicting the words. Both statistic evaluation and human evaluation have verified some advantages of our model in generating rap lyrics. Our code can be found on GitHub.

1 Introduction

Music reflects the emotions of the composers and may affect the audience in various ways. It is obvious that lyrics are a major component in determining what a good song is. Council on Communications and Media has suggested that lyrics can be significant to children's development, while parents are unaware of it (Aho and Ullman, 2009).

There have been previous works on rap lyrics generation that have drawn attention since the progress of artificial intelligence (Potash et al., 2015; Nikolov et al., 2020; Malmi et al., 2016). However, most of them failed to consider both rhymes and rhythms, until DeepRapper produced an innovative system, where they used a N-gram model and trained it in a reverse order (Xue et al., 2021). They also added a special [BEAT] mark after words that align with beats. However, we want to train our model and automatically generate rap lyrics, so we figure out a new approach.

In this paper, we develop Rhyme Generator for creating rhymed rap lyrics. To build the system, we choose MLDB (Music Lyrics Dataset) and Poetry Foundation as our dataset. The reason is quite simple. Because we want to make our model more precise in generation, a large amount of data is required. However, at the same time, the efforts on processing and evaluating also grow simultaneously. We apply different approaches when obtaining the data. We used Genius API and Kaggle for data lyrics; for the Poetry Foundation, we access the data through Kaggle (HALLMAN, 2018). Multiple factors are considered while handling the data, like syllabus and phonemes. After annotating, we then use the data for training and predicting. Inspired by Poetry Generator (MOONEY, 2017), we figure out an innovative way of combining Markov and LSTM, by feeding every possible rhyme to a Markov model and merging them into a larger model. Finally, LSTM predicts the last word and decides to correct it or not. Both statistic evaluation and human evaluation show that the lyrics generated by our model make sense to viewers.

Our major contributions can be concluded as: 1) To engineer more elegant data and take rhymes into consideration, we build a special annotator that facilitates our work. 2) To generate better rap lyrics, we figure out a special approach——building a special Markov model and then using LSTM to predict.

2 Backgrounds and Related Works

In this section, we will discuss some backgrounds of rap lyrics generation and present some related works.

Let's first look at lyrics generation. In a general picture, lyrics generation can cover rap lyrics generation, poetry lyrics generation, song lyrics generation, and so on. But what's the difference or challenge of rap lyrics generation?

It should be noted that rap lyrics have two characteristics: 1) In order to form a natural flow of rap songs, they usually contain complex rhyme patterns between consecutive sentences. 2) Since rap songs come with rhythmic accompaniment,

rap lyrics should correspond to the singing beat (Potash et al., 2015). Therefore, inspired by DeepRapper (Potash et al., 2015) and Poetry Generator (MOONEY, 2017), we introduce a novel way of rap lyrics generation considering both rhyme and rhythms, and combining Markov and LSTM.

Then we can look at some work done by previous people.

Recent works have proved that RNN, which stands for Recurrent Neural Networks, performs effectively in text generation (Sutskever et al., 2011; Graves, 2013). In his paper, RNN was used for creating language models at a character level. The results were that the model successfully learned some language rules and quite amount of English words. Besides, in Graves's paper, he used LSTM for text generation, where he claimed that LSTM is "an RNN architecture designed to be better at storing and accessing information than standard RNNs" (Graves, 2013). This proves the advancement of LSTM. Although it is used in a different field than ours, we have chosen LSTM as our tool for rap lyrics generation.

Going further, lyrics generation appears different than text generation. In "Lyrics Information Processing: Analysis, Generation, and Applications", Watanabe and Goto provided a brief introduction to the distinct nature of lyric generation compared to prose text generation (Watanabe and Goto, 2020). As lyrics are used in songs to convey emotions through audio and words, the author proposed to set lyrics information processing as a research field by itself. To be more specific, lyrics are distinct in their nature from 3 perspectives:

(1) Their sentence structure as a whole, and more importantly, oftentimes lyrics have rhyme schemes that should be followed from previous sentences. It might also follow different segmentation as a whole sentence might be segmented into three sentences of lyrics in a song.

(2) The semantics of lyrics should also be analyzed as each sentence contains words that convey the same sentiments, or the entire song might be used to tell a story. As a result, generating lyrics could also mean generating a short article of texts that has unique rhyme schemes and is capable of conveying messages and emotions through texts.

(3) The most distinct nature of lyrics comes from the fact that the text is intertwined with the audio. This feature is also discussed by Xue et al (Xue et al., 2021). Watanabe and Goto provided a few methods for lyrics processing and generation, which include ghostwriting (following a certain writing structure or rhyme scheme) with a DNN-based language model.

In addition, the challenge of rap lyrics generation has also been researched. Addanki, Saers, and Wu have discussed in "Modeling Hip Hop Challenge-Response Lyrics as Machine Translation", which is one of the earliest papers focusing on rap lyrics generation. They approached rap lyrics generation as a translation problem (Addanki et al., 2013). The author refered to the generation of hip hop lyric responses as a "translation problem" because they were drawing an analogy between the process of generating a response in a hip hop battle (which involves creating lines that rhyme and correspond in meaning or theme to the challenge presented by an opponent) and the process of translating text from one language to another. In both cases, there is a source input (a hip hop challenge or a text in the source language) that needs to be converted into a different but related output (a rhyming response or a text in the target language) that maintains some aspects of the original—such as meaning, rhythm, or rhyme in the case of hip hop, and meaning and grammatical correctness in the case of language translation.

What about using LSTM in rap lyrics generation? Potash, Romanov, and Rumshisky developed a system called GhostWriter that takes an artist's lyrics and generates "similar yet unique" lyrics (Potash et al., 2015). To generate the lyrics as well as the structure, they added "<endLine>" and "<endVerse>" tokens to the lyrics, therefore, the system is able to self-generate the lyrics and define when they should end. In our system, we also have an "<end>" tag to separate the generated lyrics. Compared to a normal n-gram model, they concluded that LSTM performs better in producing "novel lyrics that also fit the rhyming style of the target artist" (Potash et al., 2015). Their work has proved that we have made the right choice.

3 System Building

3.1 Dataset

The LSTM model needs a large amount of data to be trained in order to generate perfect lyrics, so we acquire a big dataset from two sources. We collected 100,000 songs from MLDB using Genius API and over 3 million songs from the existing dataset on Kaggle (HALLMAN, 2018).

Data is the foundation for all machine learning projects, and we have spent a lot of time generating different versions of datasets for training. Unlike prose text generation with LSTM, we cannot use all the text we have to train, since that will make it a regular text generation model. To make an LSTM that can potentially predict a rhyming word, we have proposed several methods to partition our data.

Seed Text:

```
first_line: “walking in New York”
Next_line: “I dont eat pork”
```

3.1.1 First Method

Combine paired sentences by placing the <LINEBREAKER> tag at the end of each sentence for each pair of lyrics.

Ex: [“walking in New York LINEBREAKER i dont eat pork LINEBREAKER”]

Our assumption is that the LSTM model will learn the transition probability between the last word of each paired sentence and generate a rhyming word corresponding to the <LINEBREAKER> tag.

3.1.2 Second Method

We can get the Cartesian products of all paired sentences that share the same rhyming syllables, so that we have an enormous amount of data to study. However, the data size becomes huge and thus it is impossible for us to train.

3.1.3 Third Method

Use the last word from the first_line in paired lyrics, and use all but the last word from the second sentence to train. The intention is that we want to use all the words to predict the last word, and want to use the last word from the first line for rhyming purposes.

3.1.4 Selected Method

We choose to use the last word from the first line and the second to last word from the second line to predict the last word of the response line.

Ex: X = [‘york’, ‘eat’], y = [‘pork’]

The intuition behind this is that we want to use the first word as the word to rhyme with and the second to last word as another thing that is related to the last word of the response line. This is the best method for us as our LSTM model solely focuses on generating rhyming words.

3.2 Annotator

This is one of the most important parts of our system, where we annotate the raw lyrics data, consider syllabus and stress, and decide whether the two words are rhymed or not. Data from the two sources are written into two .csv files separately. Annotator is a Python class that we define, in which we create multiple functions. Next, I will describe some of the essential functions in this class.

3.2.1 Create Phoneme Table

To be specific, whenever the annotator class is called, A phoneme table is created using the CMU Pronouncing Dictionary, which will be used in further determinations. A brief description of the phoneme table in our system is as follows: the phoneme table is a python dictionary indexed by the word and the value is a list of phonemes. The elements in the outer list correspond to alternate pronunciations of the word, and the inner list is the sequence of phonemes corresponding to that word in the CMU Pronouncing Dictionary.

3.2.2 Find Rhyme Pairs

This function requires the input to be a list of sentences, so that it can search for rhyming pairs within. This function will return a list of tuples corresponding to lines that rhyme together.

3.2.3 Determining Rhyme or not

This function takes three inputs: word, otherword, and ignore_stress. If ignore_stress equals False, it finds the last syllable in the word with primary or secondary stress and returns true if this syllable and the ones following it match those in the otherword if ignore_stress equals True, it only finds the last syllable regardless of stress and matches it to the otherword’s. A special list called "include_stress" is defined to hold stress indicators ("1" and "2") in the phonetic representation. An additional stress ’0’ is included in the list when ignore_stress is True.

The function first converts the input words to uppercase and strips any punctuation from them. It checks if the two input words are the same. If they are identical, the function returns False immediately since a word cannot rhyme with itself.

Then two phoneme lists called "wordphonemes_list" and "otherwordphonemes_list" are created by calling the get_phonemes_list() function 1 for words and otherwords. If a word is not found in the CMU dictionary, it returns False.

```
def get_phonemes_list(self, word):
    return self.__phoneme_table[word.upper()]
```

Listing 1: get_phonemes_list function

Last, the function iterates through the phoneme lists for both words. For each pronunciation of "word", it checks against the pronunciation of "otherword" to find rhyming patterns. Syllables are identified by checking for stress indicators ("1" or "2") (or the additional stress '0') at the end of the phoneme representation. If the ending phonemes of "word" with the ending phonemes of "otherword" match, the function returns True, if no rhyme is found among any of the phonetic representations, False will be returned.

Many other minor functions are not described here, for example, the annotate_POS_tags() function adds POS tags from nltk to lyrics. All the codes are available on GitHub.

3.3 System Logic

As we have mentioned in the former context, the logic behind our system is that we combine Markov and LSTM together (Markov + Bidirectional(LSTM) + Output Layer). Next, we will explain it in two parts.

3.3.1 Markov model

Our Markov model is different from others in that we group all the rhyming text first and feed the same rhyming groups to the independent Markov model. After that, those Markov models are combined together to form a robust Markov model. We notice that the Markov model is seldom used in other papers, while some have only used it for minor purposes. For example, Addanki, Saerss, and Wu train a hidden Markov model (HMM) to generate a verse of hip hop lyrics, serving as their rhyme scheme detection model (Addanki et al., 2013). However, our robust Markov model can generate more precise rap lyrics.

3.3.2 LSTM

The principle behind LSTM is that it uses sequential input data to predict the next data. For example, as is shown below, LSTM may use the input [I, love] to predict the next word to be "New".

$$[I, love] \rightarrow New$$

Furthermore, "I", "love", and "New" will be used to predict "York".

$$[I, love, New] \rightarrow York$$

Our system utilizes this principle and we figure out two factors in predicting the next word. Before predicting, we will first let Markov generate a sample of lyrics according to the input. Then, the LSTM model uses the last word of the input sentences and the second to last word from the generated sentences; or, it will use the bigram of the last word of the input sentences and the third to last word from the generated sentences. The result is that the last word of the response sentence is generated by LSTM and compared to that of the original sentence generated by the Markov model. If the two are equal, then the generated sentence passes the test; otherwise, the last word is replaced by the LSTM one. In the previous example, if we have the result as follows:

```
Input: "I love New York."
Markov: "This is a dork."
```

LSTM functions as follows:

$$[York, a] \rightarrow pork$$

Then, "dork" will be replaced by "pork".

We believe that through this double-check system, the rap lyrics we generate can be more precise and natural.

4 Evaluation

Our evaluation takes three stages, in the first stage, we detect rhyming accuracy. Then, we evaluate using these two metrics: Perplexity (PPL) and Flesch-Kincaid Grade Level (FKGL), to assess the readability, complexity, and the possibility of the sentence appearing in rap songs. In the last stage, we conduct a human evaluation process to assess whether the responses make sense or not. The two metrics are combined since FKGL solely is not convincing enough, as Tanprasert and Kauchak argued, FKGL should no longer be used as a text simplification evaluation metric, and other related statistics should be used to help understand what different systems are doing (Tanprasert and Kauchak, 2021).

4.1 Perplexity

Perplexity (PPL) is a standard metric in evaluating the quality of a language model. To be specific, it measures how likely the model is to predict the response by using a GPT2 model to calculate some probabilities, how likely the next word is, and provides a score. Although Kuribayashi et al suggested

the report that says the lower PPL an LM has, the more human-like the LM is is not always human-like, their research focused on cross-language evaluation (Kuribayashi et al., 2021). Therefore, PPL is still a convincing metric for evaluation.

4.2 Flesch-Kincaid Grade Level

FleschKincaid Grade Level (FKGL) was first proposed in the 1940s (Flesch, 1948), though it was initially used in the medical domain. FKGL requires two inputs to calculate the score: the average number of syllables per word and the average number of words per sentence. The formula is as follows:

$$\text{FKGL} = 0.39 \left(\frac{\text{total words}}{\text{total sentences}} \right) + 11.8 \left(\frac{\text{total syllables}}{\text{total words}} \right) - 15.59$$

The score of FKGL indicates the approximate grade level necessary to understand a piece of text, through which, we believe, the readability of the rap lyrics can be effectively measured.

4.3 Human Evaluation

We choose 500 paired randomly selected lyrics from our generation and put them in a survey. Three different online graders are asked to assess whether the two paired lyrics rhyme or not and whether they make sense.

```
Input: comin out the darkness angels were callin its all right
Output: you know i get white light
```

```
Input: billy cook you know i'm a superstar
Output: dont be smoking on my guitar
```

Figure 1: example of generated lyrics

5 Experiment Results

We use two thousand unseen lyrics from 2000 different songs, as Figure 1 shows, and we generate outputs to first inspect whether they rhyme or not. The rhyming accuracy is around 26.8%. We then post 500 paired lyrics (randomly selected) as a survey and have three different online graders for assessment. The result for rhyming accuracy is similar with 28.8%, 25.3%, and 23.2% from the three different graders we have.

	perplexity	fk_grade
count	2000.000000	2000.000000
mean	1376.226971	-0.290850
std	2637.435953	2.150757
min	19.737740	-2.300000
25%	272.410469	-1.900000
50%	652.458649	-1.500000
75%	1268.365326	0.900000
max	29513.203125	9.600000

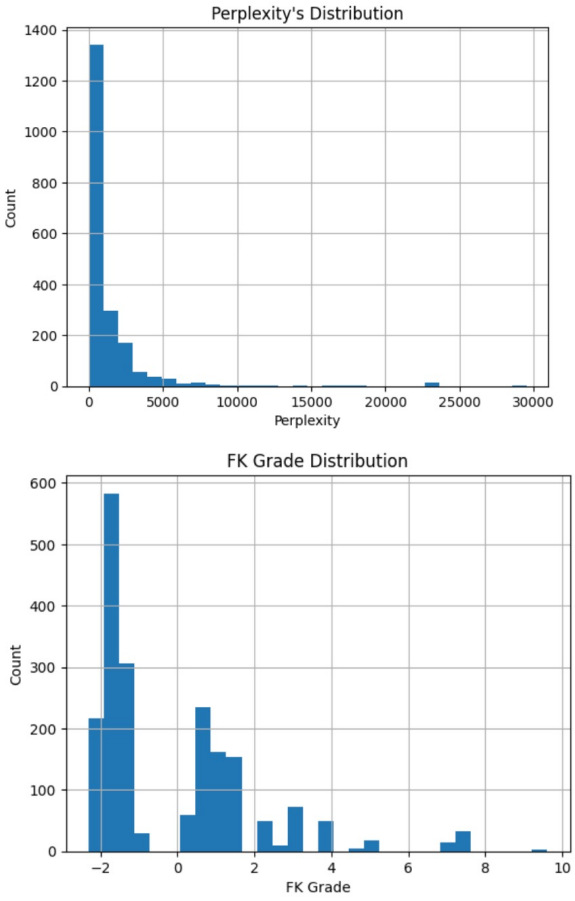

Figure 2: perplexity and FKGL score table

5.1 Perplexity and FKGL

For perplexity, the median is around 652, signaling that most of our responses are most likely not generated by humans or are not likely to be in our day-to-day conversation, which we believe the reason is due to the nature of lyrics since artists are not trying to have a conversation most of the time but rather just trying to use the right words to convey messages. The mean FK grade is around 0, and a low FK grade signals that our sentences are essentially not as complicated as perplexity suggested.

5.2 Human Evaluation

From our human evaluation results, 65.2%, 57.6%, and 53.6% of the sentences make sense to the graders, which is surprising since the perplexity is pretty high, but as lyrics, we do have to take into account that they are not regular sentences and making sense is more subjective than the objective goal of detecting rhymes.

5.3 Error Analysis

One of the profound findings is that if the second to last word from the output sentence generated from Markov is a determiner, the rhyming accuracy is significantly higher compared to the accuracy of the response generated from a verb as the word before the last word. The rhyming accuracy with the second to last word being a determiner is 49.4% (determiners used "the", "a", or "an"). Moreover, we see that the model does not do as well as expected on verbs as the second to last word of the output and if the rhyming word and second to last word are unseen.

5.4 Future Improvements

Using Hadoop and MapReduce will significantly increase the speed of processing big data, we would have used 80% of our lyrics data (3 million songs) for training, but due to funding and inability to process big data with our devices for deep learning, we didn't have the opportunity to train with more data. Moreover, with better data, we believe generating text using LSTM then using LSTM to rhyme would be a better way of generating lyrics with input; however, with deep learning the data and parameter required is huge, which is something we have learned later in the stage. Moreover, domain knowledge is certainly key to natural language projects. After interviewing a rapper and a free-style battle rapper recently, we have thought about making models imitating how humans think about generating rhymes. For example, the person would sometimes think of the word that rhymes first, before thinking about the entire next sentence. Occassionally, lyrics itself don't make as much sense when combined together. We tried to reverse the second rhyming paired lyric to train the model, but the response did not make much sense. To incorporate the rapper's method into our model, we need to figure out a more elegant solution to split our data. Most importantly, the data in the deep learning model plays a vital role in this project, and the quality of the data determines the quality of the model. Using rule-based models with huge data, POS tagging, audio feature, and Hidden Markov Model should be a great way for future version of rhyming lyrics generator. Additionally, another quality of the data that could be considered in the future is to using the cartesian product of all the dirty rhymes, which will almost resembled actual lyrics.

6 Discussion

In this part, we will discuss a model we have tried to develop. Due to the limitation of equipment, but we list below as plan for future improvements.

We have attempted using two LSTM models, one for generating the output, and one for correcting the last word. The structure is similar to our current one. The first LSTM uses a bidirectional method to generate lyrics, and the second one will generate the last word by looking at the last word of the input sentence and the second last word of the generated one.

For example, if the input and the result are as follows:

> Input: This is a pork.
> LSTM1: I don't use knife.

Then, LSTM2 will do the following:

> Input: [pork, use]
> LSTM2: [fork]
> Corrected result: I don't use fork.

However, we have tested that the results are not so well. It is likely that there can be improvements to this model, but we will leave it as future work.

7 Conclusion

In this paper, we develop Rhyme Generator, that uses Markov and LSTM for generating rhymed

rap lyrics. We carefully process the data and use the last word from the first line and the second to last word from the next line to predict the last word. With the double-check model of Markov and LSTM, our results are much more positive. Both statistic evaluation and human evaluation are conducted to ensure the quality of our model. Though PPL score suggests that lyrics generated by our model may be uncommon in daily life, human evaluations prove our assumption and that the lyrics actually make sense. We believe that using some of our methods, other systems can also be improved, such as DeepRapper (Xue et al., 2021). We plan to do this in our future development.

References

Karteek Addanki, Markus Saers, and Dekai Wu. 2013. Modeling hip hop challenge-response lyrics as machine translation. In *Proceedings of Machine Translation Summit XIV: Papers*, Nice, France.

Alfred V. Aho and Jeffrey D. Ullman. 2009. From the american academy of pediatrics: Policy statement–impact of music, music lyrics, and music videos on children and youth. *Pediatrics*, 124(5):1488–94.

Rudolph Flesch. 1948. A new readability yardstick. *Journal of applied psychology*, 32(3):221.

Alex Graves. 2013. Generating sequences with recurrent neural networks. *ArXiv*, abs/1308.0850.

JOHN HALLMAN. 2018. Complete poetryfoundation.org dataset. Https://www.kaggle.com/datasets/johnhallman/complete-poetryfoundationorg-dataset.

Tatsuki Kuribayashi, Yohei Oseki, Takumi Ito, Ryo Yoshida, Masayuki Asahara, and Kentaro Inui. 2021. Lower perplexity is not always human-like. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 5203–5217, Online. Association for Computational Linguistics.

Eric Malmi, Pyry Takala, Hannu Toivonen, Tapani Raiko, and Aristides Gionis. 2016. Dopelearning: A computational approach to rap lyrics generation. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

PAUL MOONEY. 2017. Poetry generator (rnn markov). Https://www.kaggle.com/code/paultimothymooney/poetry-generator-rnn-markov.

Nikola I. Nikolov, Eric Malmi, Curtis Northcutt, and Loreto Parisi. 2020. Rapformer: Conditional rap lyrics generation with denoising autoencoders. In *Proceedings of the 13th International Conference on Natural Language Generation*, pages 360–373, Dublin, Ireland. Association for Computational Linguistics.

Peter Potash, Alexey Romanov, and Anna Rumshisky. 2015. GhostWriter: Using an LSTM for automatic rap lyric generation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1919–1924, Lisbon, Portugal. Association for Computational Linguistics.

Ilya Sutskever, James Martens, and Geoffrey E Hinton. 2011. Generating text with recurrent neural networks. *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 1017–1024.

Teerapaun Tanprasert and David Kauchak. 2021. Flesch-kincaid is not a text simplification evaluation metric. In *Proceedings of the 1st Workshop on Natural Language Generation, Evaluation, and Metrics (GEM 2021)*, pages 1–14, Online. Association for Computational Linguistics.

Kento Watanabe and Masataka Goto. 2020. Lyrics information processing: Analysis, generation, and applications. In *Proceedings of the 1st Workshop on NLP for Music and Audio (NLP4MusA)*, pages 6–12, Online. Association for Computational Linguistics.

Lanqing Xue, Kaitao Song, Duocai Wu, Xu Tan, Nevin L. Zhang, Tao Qin, Wei-Qiang Zhang, and Tie-Yan Liu. 2021. DeepRapper: Neural rap generation with rhyme and rhythm modeling. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 69–81, Online. Association for Computational Linguistics.