Hong Kong University of Science and Technology
Computer Science & Engineering Department

MSBD 5002 Group 5: Rap Lyrics Generation

Name	Student ID
Shiu Stephen	20041373
Ko Wing Fung	20279217
Koo Tin Lok	20344775
Ng Pui Yan	20670370
Muhammad Adam Usmani	20729858
Qiwei Li	20720230

Introduction

Rapping emerged from a hobby of African American youth in the 1970's, it quickly evolved into a mainstream music genre with several artists frequenting Billboard top rankings. Writing high quality rap lyrics requires creativity, to be able to construct interesting-story-telling, and-also lyrical skills, which is the foundation behind forming rhythmic complexity. In our paper, we will be proposing several deep learning methods to tackle rap lyric-generation. Our goal is to capture the patterns and styles of multiple lyricists to generate new lyrics so that we could generate novel lyrics at the same time preserve artists' style. Rap lyrics possess a formal structure which is made up of very carefully crafted sound and rhyme. Our focus is to analyze this formal structure and develop models that are-able-to generate the same artistic style that is present in multiple rap songs.

Motivation

Creation has always been people's dream and being able to write a good rap lyric is a lot of people's expectations for the future. However, the creation of rap itself has a high pre-request, and many people are discouraged from it. Not only does it take time to consider the rhythm, but it also needs to spend time on the idea of lyrics. However, with the rapid development of neural network technology in recent years and the significant improvement of computer performance, there are more and more applications of using computers to create lyrics and rhythm. A large number of papers have discussed how to use neural networks to generate lyrics. In addition, the development of the Internet now enables us to easily obtain a large number of lyrics data without too much effort, which helps us to make the trained model more accurate. Based on these two points, we decided on the overall plan of this project-using neural networks to generate lyrics based on deep learning method. We believe that after using these technologies, people will be more able to access rap creation, and at the same time, it can greatly reduce the entry barrier for ordinary people in rap creation. We believe that in the near future, computer-created songs will become popular in people's daily lives.

Preprocessing

In order to generate the lyrics, we first need to prepare our own dataset. This will include web scraping, data cleaning and features extraction. In this section we will talk about the technique we used to collect the lyrics and the state of art system we built to clean up the dataset, and finally we will discuss how to extract the features from the dataset.

Scraping of the dataset

For the dataset, we gathered our own dataset. There are some public datasets on Kaggle, but their size is small, that dataset contains only about 500 songs. As a result, we decided to gather our own data by scrapping data from "OHHLA website". The website archives English rap lyrics from 3604 rappers starting since 1992.

The website is open to public and it organizes song lyrics in a structured hierarchy. Thus, it took us only a little effort to crawl all the lyrics. We managed to crawl 103682 webpages in an hour.

The raw data have a large size and there are some unnecessary information and meta data inside the lyric files such as [Verse], [Chorus], author information, song information, etc. We scan all the files and used regular express to perform a clean up to delete those meta data tags and result in plaintext files that contain only lyrics lines. We also filtered some lyrics files that contain only a few lines (less than 8 lines) that are generally useless to use. The final result are 90468 lyrics files. The whole data scraping process took us only an hour.

Lyrics Cleaning

In the Natural Language Processing (NLP) task, the ultimate goal for this is let the computer understand the context of the words (Vijayarani, 2015). The online lyrics we scrapped may have the problems (1) using the different grammatically form of words (Vijayarani, 2015), (2) mis-spelling words (Gupta, 2013), (3) including some non-English words and (4) not useful sentences. We will discuss each of these in details. If we don't do the text cleaning before the training, we could get some unexpected training results. Since we are dividing our groups into multiple sub-groups, we built a state of art system which can do all of these text preprocessing jobs in a pipeline system. Easy to extend and fast to run.

Transform different grammatically form of words

This problem exists a lot in the lyrics we have after we scrapped the data. Because these lyrics are from different person in different albums, so they use different representations of same word. For example, some people prefer word "till" instead of "until", and some people use word "ya" instead of "you". Even if people all use formal English in their lyrics, there are bunch of different dialects of English exists (List of dialects of English, 2021). We built a system which will handle this by mapping all of the words into a fixed form and output this transformed form of lyrics to the next step.

Correct misspelling words

Since all of our lyrics are coming from the web, it may contain some misspelling words. This is caused by the fact that these lyrics are coming from the people online and may not coming from the trustful sources like album publishers. To mitigate this problem, we can use a library called (TextBlob, n.d.). This library provides a simple API to utilize its machine learning model to correct misspelling words. However, currently this library can only use on English words and may have some constrains when facing some non-common errors such that after using this library, it still might have some words un-processed. We will handle this problem in the next step.

Non-English words

This problem is related to the natural property of the lyrics itself: artists coming from different culture background and may use other languages not in English as part of the song. This is very common in songs from the United States since there are more than 60,481,746 native speakers in United States (Wikipedia, n.d.). Other reasons might be some unprocessed mis-spelling words from previous stage, some symbols like "@" and "*" in the lyrics. And also, some self-created words might appear in the lyrics. To solve this problem, we use enumerate all words split by space from the lyrics and try to find a match in our English dictionary. If not found, then we will simply remove it from our dataset so that it won't become an outlier in training. Also, by using this method, we can guarantee that there is no non-English word exist in our training dataset.

Non-useful sentences

This issue exists in our original dataset. A lot of words like [Repeat] exist. The method we used to mitigate is trying to replace all words in form of "[*]" to a space. This method is fast and can guarantee to remove all the occurrences. However, in the actual text pre-processing stage, we need to put this step at the very beginning of the pipeline, otherwise, it will be replaced by other preprocessed steps to unwanted replacement. For example, [Repeat] now becomes Repeat and will be used in training.

System design

Since we want to make our preprocessor has a state of art design with three key functions. (1) Easy to extend, (2) easy to use, and (3) easy to debug. We finally created a preprocess pipeline class in Python. This class has multiple method to process an original string. Since it is a pipeline, we can easily add more methods to it or change the order of the pipeline. However, in the early experiment, we found that the method used to correct spelling takes a lot of time in processing. It takes roughly 2 days to process 10,000 records. To speed up this, we use a python library (nalepae, n.d.) which utilize the parallel processing method and will run our code in multi-core settings. After implementing this method into preprocessor, it speeds up 10x compare to previous experiment (Using testing computer with 12 cores available). And we used less than 2 days to process all 90,000 records. Since we have one common preprocessor to use, all members in the group can use the same preprocessed lyrics in their training which can provide a consistence results when doing the evaluation.

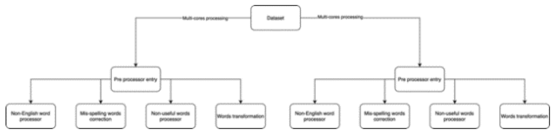

Dataset
Multi-cores processing
Multi-cores processing
Pre processor entry
Pre processor entry
Non-English word processor
Mis-spelling words correction
Non-useful words processor
Words transformation
Non-English word processor
Mis-spelling words correction
Non-useful words processor
Words transformation

Figure 1: pre-processor design

Features

In this project, information retrieval (IR) is a critical part for all the methods used for rap lyrics generation, including the word-by-word generation and line-by-line generation. Rap song differs from other music genres due to its strong rhythm and better flow to the preceding lyrics. In this sense, rap lyrics generation differs from text generation. The latter one focuses on correlation between words (e.g. the N-gram model) and the semantic meaning of overall sentence, while the former one emphasizes the pronunciation of words. From literature professor Adam Bradley, rap songs have formal structure of literary, compared to other popular lyrics [1].

Therefore, by extracting the unique structure of rap lyrics, the features are used in modelling to give the best prediction of the next line. Meanwhile, parts of them are appropriate assessment metrics to evaluate “how good is the generated lyrics” in a qualitative perspective.

The below features are computed based on the similarity between the potential next line l and the existing lyrics B consisting of n existing lines $(b_1, b_2, \dots, b_n)$. By having both as inputs, a vector of scores will then be computed for the model training. The mathematical expression is as follows:

$$\Phi(B, l) = X$$

$$, X \in R^n \ and\ n\ subjects\ to\ number\ of\ features$$

The features in this project can be categories into three aspects, capturing (i) rhyming, (ii) structural similarity and (iii) semantic similarity.

Rhyming

Phonetic transformation

The rationale behind for transforming every word to a phonetic transcription is for rhyme feature extractions. In this project, a collection of corpuses, namely “cmudict” built by Carnegie Mellon University at 1998 is used for the transformation. According to [2], this corpus contains mappings between 134,000 North American English words to pronunciation. Fundamentally, this mapping requires the target word to have an exact match and the conversion of a word to a correct spelling by a spell corrector based on Bayes Theorem [3]. Therefore, in additional to the preprocessing techniques mentioned, the text’s spell correction is exceptionally important for the transformation of a sentence to phonetic transcriptions representation.

Apart from “cmudict”’s authenticity, vowels are marked with stress level. It brings more information to bring higher accuracy of rhyme identification. For example,

“wireless” → ['W', 'AY1', 'R', 'L', 'IH0', 'S'] vector. In this representation, the phoneme ‘AY’ is marked as 1, meaning there is a primary stress on ‘i’ in the pronunciation while the phoneme ‘IH0’ indicates that the ‘e’ should not be stressed although it is a vowel. It can be deduced that if every word in the every lyrics is transformed to above vectors would eventually make the dimension of over 90000 songs huge. It would turn out make the process of machine learning building computationally expensive. For example, consider the below two sentences,

“Hong Kong University of Science of Technology is beautiful.

It is long.” →

```
[
    [
        [HH', 'AO1', 'NG'], ['K', 'AO1', 'NG'], ['Y', 'UW2', 'N', 'AH0', 'V', 'ER1', 'S', 'AH0', 'T', 'IY0'], ['AH1', 'V'], ['S', 'AY1', 'AH0', 'N', 'S'], ['AH0', 'N', 'D'], ['T', 'EH0', 'K', 'N', 'AA1', 'L', 'AH0', 'JH', 'IY0'], ['IH1', 'Z'], ['B', 'Y', 'UW1', 'T', 'AH0', 'F', 'AH0', 'L']
    ],
    [
        ['IH1', 'T'], ['IH1', 'Z'], ['L', 'AO1', 'NG']
    ]
]
```

Instead, the vector representation is simplified by

1. Omit all the non-vowels phonemes
2. Ignore all the spaces between words to flatten all the vowels’ phonemes in a line into a single vector. The reason is that only vowels contribute rhyming and the space only controls the speed of pronunciation rather than the pronunciation itself. The final phonetic representation of above example would be:

“Hong Kong University of Science of Technology is beautiful.

It is long.” →

```
[
    ['AO1', 'AO1', 'UW2', 'AH0', 'ER1', 'AH0', 'IY0', 'AH1', 'AY1', 'AH0', 'AH1', 'EH0', 'AA1', 'AH0', 'IY0', 'IH1', 'UW1', 'AH0', 'AH0'],

    ['IH1', 'IH1', 'AO1']
]
```

Due to the complexity for the transformation including the text correction, this step is parallelized by 4 cores which take less than 20 hours to complete the transformation of 90000 rap songs.

EndRhyme

There are various rhyme types, such as “perfect rhyme”, “alliteration” and “consonance” while “assonance” is the most common rhyme type nowadays. In particular, perfect rhyme highlights the words share exactly same end sound, while “assonance” highlights only the vowels sounds are shared. For example,

“spectacular” → ['S', 'P', 'EH0', 'K', 'T', 'AE1', 'K', 'Y', 'AH0', 'L', 'ER0'] → ['EH0', 'AE1', **'AH0', 'ER0'**]

“perpendicular” → ['P', 'ER2', 'P', 'AH0', 'N', 'D', 'IH1', 'K', 'Y', 'AH0', 'L', 'ER0'] → ['ER2', 'AH0', 'IH1', **'AH0', 'ER0'**]

In the phonetic representation, it can be seen that “spectacular” and “perpendicular” have the two common vowel phonemes while the common phoneme is a strong indication for rhyme. It is observed that the common parts are only at the last phonemes but not the other parts. In other words, the tail of vector is the first to look at when distinguishing the rhyme. The EndRhyme algorithm converts two lines, the last sentence in B, namely b_n and l, into the phonetic transcriptions and compare the last vowels phonemes. Lastly, the algorithm outputs the number of common phonemes at the tails. It is also known as the longest common prefix problem.

EndRhyme-1

The EndRhyme-1 is similar to EndRhyme except it compares the second last line in B, b_{n-1} to the target candidate l. Then, it counts the common phonemes at the tails. It captures the alternating rhyme patterns "abab".

Rhyme Density

Instead of capturing the rhyme of the last word in two lines, multisyllabic assonance rhyme does not only cover the end of line, but it spans multiple words. For example, consider two lines:

"This is a job – I get paid to **sling some raps**. What you made last year was less than my **income tax**, keep in mind"

From the pronunciations, it can be distinguished that the last words in the first line "sling some raps" and the second lines "income tax" have fairly similar sounds. To view it in phonetic transcription, here are the result.

The vowels phonetic representation of the first line is:

['IH', 'IH', 'AH', 'AA', 'AY', 'EH', 'EY', 'UW', **'IH', 'AH', 'AE'**]

The second line is:

['AH', 'UW', 'EY', 'AE', 'IH', 'AA', 'EH', 'AE', 'AY', **'IH', 'AH', 'AE'**, 'IY', 'IH', 'AY']

After the transformation, it can be seen that there are common vowels sequence in the middle on either lines. As proposed in [4], the calculation of rhyme density are as follows:

1. Compute the phonetic transcription of the lyrics and remove all but the vowel phonemes. For simplicity, the stress level is dropped in this part

2. Scan the lyrics word by word

3. For each word, scan back the past k phonemes by the given scanning range and find the longest matching vowel sequences of all words. It can be done by scanning back the phonemes, increment the count of the common vowel phonemes and terminate when the first different phonemes are hit

4. Among the longest matching sequences counts of all words, compute the average and the output is the rhyme density

Rhyme density is not just a feature to score the potential line of lyric by knowing it matches with the rhyme, it is also a good assessment metric to value the rap song. A rhyme density analysis is performed for all the rap songs in the dataset. Due to the $O(n)$ computational complexity, a multiprocessing is performed to compute the rhyme density of all 90000 songs. The distribution is as follows:

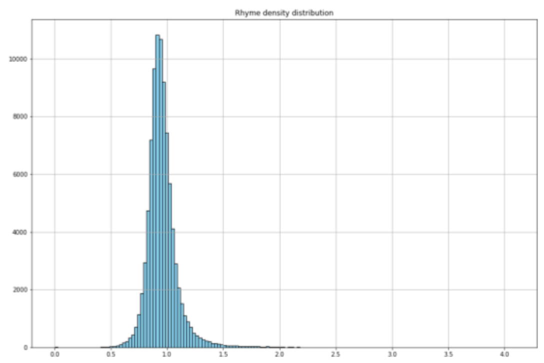


Figure 2 Rhyme Density distribution in the dataset

The mean μ of Rhyme Density is 0.96 and the standard deviation σ is 0.16. It means if the rhyme density of generated lyrics is equal or larger than ($\mu + 2\sigma$) = 1.28, the Rhyme Density is statistically significant and we consider it as a satisfactory result.

The top 10 rapper with highest Rhyme Density is as follow. Note that the Rhyme Density only considers the multisyllabic assonance rhyme mentioned without the consideration of other factors like the semantic meaning, rhythm and the background music. Therefore, it should not be taken as the only assessment of the quality.

Artist	Rhyme_density
Big_B	1.899563
Bloodraw	1.780627
DJ_Class	1.728140
Raheem_(the_Dream)	1.626794
Hkeem	1.448069
Lightborn	1.429022
DJ_Pied_Piper	1.404731
Big_Stan	1.390625
Jimmy_Spicer	1.386161
Juganot	1.381232

Table 1

Structure similarity

LineLength

As discussed in [4], the consecutive lines are typically having same length because they need be sung out within one musical bar of a fixed length. Therefore, LineLength is introduced to compare the similarity of line length of last line in B (b_n) and l. The computation is as follow:

$$1 - \frac{|len(l) - len(b_n)|}{max(len(l), len(b_n))}$$

Note that the LineLength is normalized by the expression above and it ranges from 0 to 1 for the least similar to the most similar.

Semantic similarity

JaccardDist-1

It is common that the lines of lyrics share similar words. To measure this, last line in B (b_n) is tokenized into a bag of words S_m and the potential line is tokenized into L. Then, the Jaccard similarity between the corresponding bags of words is computed as:

$$\frac{|S_m \cap L|}{|S_m \cup L|}$$

Note that the JaccardDist-1 is normalized and it ranges from 0 to 1 for $L \cap S_m = \emptyset$ to $L = S_m$

JaccardDist-5

JaccardDist-5 is similar to JaccardDist-1 except it tokenizes the last 5 lines in B ($b_{n-4}, b_{n-3}, b_{n-2}, b_{n-1}, b_n$) into bag of words and compare to L. The computation is as follow:

$$\frac{|(\bigcup_{j=m-5}^{m} S_j) \cap L|}{|(\bigcup_{j=m-5}^{m} S_j) \cup L|}$$

TfidfDist-5

The next line of lyrics is likely to match the theme of the preceding lines of lyrics. One evidence is that they share similar words with equal importance. Term Frequency –

Inverse Document Frequency (TF-IDF) measures the importance of a word by two principles. 1. Within a single document, if the term appears frequently, this term tends to be more important than the others. 2. Across different documents, if the term rarely appears, it tends to be important. TF-IDF is calculated as below:

$$w(t,d) = TF(t,d) \times IDF(t)$$

$$, where\ TF(t,d) = c(t,d)\ and\ IDF(t) = 1 + \log\left(\frac{N}{df(t)}\right)$$

In TfidfDist-5, the terms in the last 5 lines ($b_{n-4}, b_{n-3}, b_{n-2}, b_{n-1}, b_n$) in B are vectorized with TF-IDF. The candidate of next line is also vectorized with the corpus built. Tfidf-5 is computed as follows:

$$\frac{1}{5}\sum_{i=0}^{4} tfidf(l,d) \cdot tfidf(b_{n-i},d)$$

UseDist-5

Bags-of-word based JaccardDist and term frequency and document frequency based TfidfDist are not capable of handling synonymy nor polysemy [4]. In other words, both models may not be able to give a high score for a next line l with different words but similar meaning to the preceding lines B. As observed in the dataset, rapper tends to use a large variety of vocabularies with close meanings instead of the same words to give a high diversity. To capture the synonymy and polysemy, Universal Sentence Encoder (USE) which was developed by Google Research in 2018 is used to summarizes any given sentence to a high dimensional sentence embedding [5]. It was trained from data sources like Wikipedia, web news, QA pages and discussion forums.

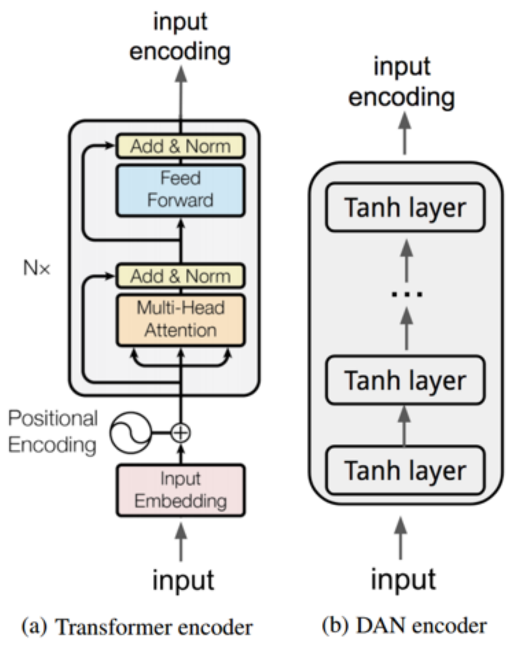
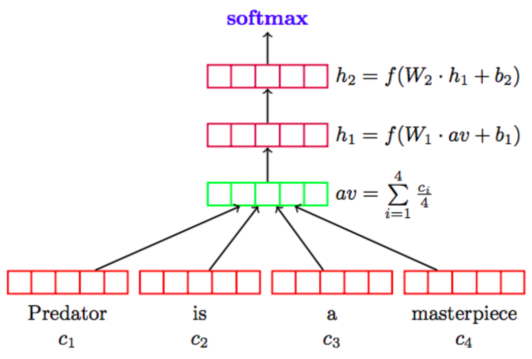


Figure 3 Transformer (left) and DAN (right) version of USE Embedding

There are two releases of USE, one is Transformer based while one is Deep Averaging Network (DAN) based. For Transformer version, it is composed of 6 layers. Each layer has sub-layers, one uses attention to compute the awareness of orders and identity of all words while the second one is just feedforward network. Without any convolutional or average pooling layers, it causes the transformer to have a high complexity of computation $O(n^2)$ and memory $O(n^2)$.

The DAN based model is much simpler. It takes in sentences and generate 512-dimensional sentence embedding with PTB tokenizer (Split the sentences according to specific regular expressions). The input embedding and bi-grams are averaged before passing to a DNN. The complexities are $O(n)$ for computation and $O(1)$ for memory.



softmax
$h_2 = f(W_2 \cdot h_1 + b_2)$
$h_1 = f(W_1 \cdot av + b_1)$
$av = \sum_{i=1}^{4} \frac{c_i}{4}$
Predator c_1
is c_2
a c_3
masterpiece c_4

Figure 4 DAN Encoder

The advantage of USE is that it is able to capture tricky synonyms, homonyms, typos and etc. As shown in below diagram, even the sentences are having different words, as long as their themes are close, USE can still give a high correlation between sentences based on semantic textual similarity. The drawback is that it requires searching through millions of vectors which is expensive compared to the classical keywords search.

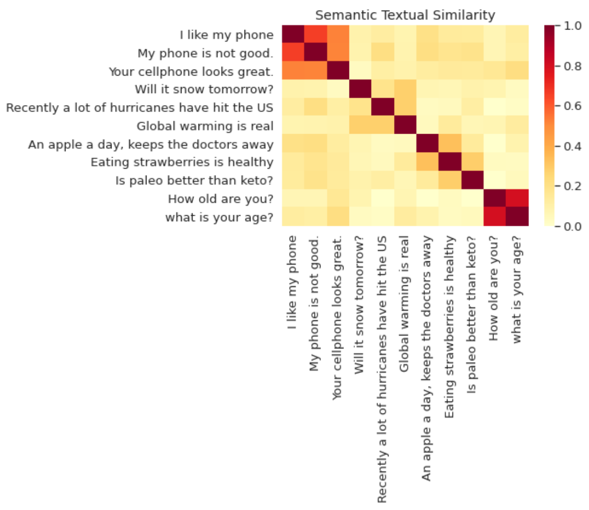


Figure 5 Semantic similarity heatmap based on USE

In this project, UseDist-5 is calculated from the dot products of embedded vector of L and the embedded vectors of the last 5 lines $(b_{n-4}, b_{n-3}, b_{n-2}, b_{n-1}, b_n)$ in B.

$$\frac{1}{5}\sum_{i=0}^{4} USE\_embed(l) \cdot USE\_embed(B_{last-i})$$

Models

Ghost Writer Models

Baseline Model

To compare with the results of the LSTM model, we created a Markov model for lyric generation. Our baseline simplifies to a basic n-gram model.

The model works by using the previous k words in the corpus as the current state, and then modelling the probabilities of the next token. This involves creating a vector for each distinct sequence of k words, having N components, where N is the total quantity of distinct words in the corpus. We then add 1 to the j-th component of the i-th vector, where i is the index of the i-th k-sequence of words, and j is the index of the next word. Finally, normalizing each word vector gives a probability distribution for the next word, given the previous k tokens.

We can frame this using a simple example, say we have a sentence 'This sentence has five words'. For this example, we use k=1 where k is the quantity of words our chain considers before sampling/predicting the next word. Our sentence has five distinct sequences of 1 word (one for each word). It is worth noting that duplicate words do not contribute to this number.

We start by initializing a 5×5 matrix of zeroes. We then add 1 to the column corresponding to ‘sentence’ on the row for ‘this’. We then add another 1 on the row for ‘sentence’, on the column for ‘has’. We continue this process until we’ve gone through the whole sentence.

Following this approach gives us the resulting matrix where the diagonal pattern comes from the ordering of the words.

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Since each word only appears once, this model would simply generate the same sentence over and over but adding more words can make this more interesting.

In our model, we consider only n-grams with k up to three since chains with a higher value for k increases the chance of obtaining deterministic results. This is because longer sequences are more likely to be unique.

LSTM Model

Previous work has shown that applying RNNs to language modeling can bring about incredible success. Essential to the success of RNN has been the use of the LSTM, a very special kind of recurrent neural network which is capable of learning long term dependencies, and which has proven to perform very well when it comes to sequence forecasting. In our attempt to generate rap lyrics, we hope that using an LSTM can help us capture the rhythmic style of an artist by learning rhyme and meter patterns.

We start off by carrying out some preprocessing. We apply a tokenizer to all the words in the data, this vectorizes each word into a sequence of integers and creates a dictionary. The next step is to take the list of tokens and create input sequences. This is essentially an input vector made up of sequences of each word in a sentence starting from the first word and followed by the proceeding word which is used for the purpose of prediction, it then eventually ends with the final word in the sentence.

An example is given below,

```
I love [1,200]
I love to [1,200,3]
I love to procratinate [1,200,3,504]
```

The final preprocessing step is to find the max-length of a sequence and pad all other sequences with 0s so that all sequences are of the same length.

The next step in setting up our model is to create the training data. We treat the first n-1 (where n is the number of words in the sequence) words of an input sequence as predictors and the nth word as the label. We then convert the labels to categorical labels by one hot encoding them and treating the total number of words in the dictionary as the total number of classes.

Finally, we are ready to build our model. We use the sequential method in Keras since it is the simplest one to create a stacking of layer architecture. We add embedding which is a special method specifically made for handling text data. After this, we add the bidirectional LSTM layer. We then add another LSTM layer, add the final dense layer which is the size of total words to predict. We added some dropouts in between to avoid overfitting the data. We finish up by using categorical cross-entropy as the loss function and Adam as the optimizer.

Our LSTM model looks something like this,

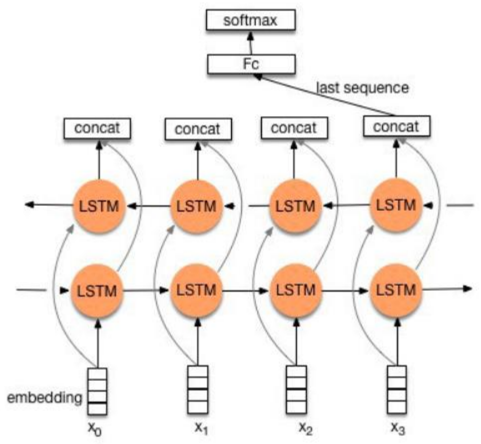


Figure 6 LSTM Model

Dope Learning Models

The dope learning algorithm proposed in [4] achieves the rap lyrics generation through line-by-line generation. It approaches the problem from Information Retrieval (IR). First, it treats the dataset containing 90000 rap songs as a large database. Then, it considers the last m lines in the partially constructed rap song as a query and identify the most relevant next line from the collection of candidates lines that is selected by a highly efficient algorithm. The candidates are further filtered by 1. Semantic scores 2. A trained deep learning model and 3. A RankSVM to select the best line.

Problem definition

Consider the lyrics of partially constructed rap song B formed by sequence of lines $(b_1, b_2, \ldots, b_n)$ are known as query. Treat the lines of lyrics in the dataset as a database D that is composed of 90000 rap songs. The goal is to 1. Identify a collection of k candidates C = $\{l_1, l_2, \ldots, l_k\}$from D and restricted that the true next line $b_{n+1} \notin C$. 2. Among the k candidates, choose the best one to be the next line $b_{n+1} = l_i$

Algorithm: Lyrics generation algorithm

```
Input: Seed a line $b_1$ and the number of lines n to generate
Output: Lyrics B = $(b_1, b_2, \ldots, b_n)$
B[1] ← $b_1$;
for $i$ ← 2 to n do
    C ← retrieve_candidates(B);
    C ← nn5(B);
    C ← top5_most_related_semantic_meaning(C);
    best_candidate = $\hat{c}$;
    foreach c ∈ C do
        if $relevance\_score(c, B) > relevance\_score(\hat{c}, B)$ & rhyme_ok(c, B) then
            $\hat{c}$ ← c;
    B[i] ← $\hat{c}$;
return B
```

Candidate retrieval

The goal of candidate retrieval is to find out a subset of lines which fulfil a good rap song criterion. Without inputting many lines into our model that is introduced later, candidate retrieval can largely reduce the computational resources. In this project, EndRhyme is used to be the first criteria to filter the candidates because it is a alone a good predictor. That is, to fetch k number of candidates which have the non-zero EndRhyme with B. As explained, phonetic transformation is needed to identify which lines rhyme with the last line in B. As a result, the following algorithm is performed in candidate retrieval.

1. Transform all the lines in the dataset into phonetic representations and remove all except the vowels
2. Reverse the each of the one-dimensional vector such that the last vowel phoneme is at the front
3. Perform Quicksort to sort all the lines according to the alphabetical order of vowels in phonetic form.
4. Given the last line in B, it performs binary search to find the line with longest common prefix, namely l_{long}
5. Not only return the l_{long}, but also return the k-1 lines around l_{long} which may share the common prefix with last line in B
6. From the all the candidates, if the pairwise normalized Levenshtein distance between candidates is lower than the threshold, one of the candidates will be dropped and the selection of a candidate will begin again until it has larger Levenshtein distance with the others.

In step 3 and 4, the Quicksort takes $O(\text{n}log\, n)$ to sort all the lines. In here, it only returns the indices that would sort the lines, instead of reorder the entire data. It is because we need to keep track of the source (artist, song) of the line. Once the sorting is finished, it only takes $O(log\, n)$ to find the candidates, and hence an efficient method to filter.

In step 6, it ensures the candidates are not similar to each other. As it is desired that a generated lyrics maintain a high diversity, but not having many repeated lines. Levenshtein distance is an algorithm to calculate the editing distance between two strings by the cost of insertion, deletion and updating. The time complexity of Levenshtein distance can further be optimized by dynamic programming.

Filter by semantic meaning

It may be essential that lines in the rap lyrics have similar theme. In this project, rather than having the JaccardDist and TfidfDist which will be used in the modelling part later, one more gate implemented by UseDist is used in advance to find out the top 5 most related lines among the candidates. It is used because it is considered the most powerful to consider the synonymy and polysemy compared to the other two.

The UseDist-5 which is based on the Universal Sentence Encoder is used to find the semantic correlation between the line with the last five lines in B. After that, top 5 lines with the closest meaning will be extracted from C.

Neural Network based language modelling

All the other features are algorithm based, meaning that they are not comprehensive. Deep learning feature extractor can be a powerful tool that handle the grammatical, semantic, rhymical and other not obvious features between the lyrics lines.

Therefore, we tried to train a feature extractor nn5. The feature extractor should be able to discover the relationship between lines, words, and the rhymes.

Training and evaluation method

The nn5 feature extractor is trained as follows. We fit the neural network that do classification task such that it would take a lyrics line (target) and 5 previous line and predict if the target line is the next line of the previous line. And then we use the last layer as the features. Training examples are also easy to generate, we randomly pick a lyric line $l_{a,b}$ and its 5 previous lines $l_{a,b-1:b-5}$ as truth data and we randomly pick another line $l_{c,d}$ with $l_{a,b-1:b-5}$ as fake data where

$a \in Set\ of\ Song,$

$b \in Set\ of\ lyric\ lines\ in\ Song\ a,$

and $(a,b) \neq (c,d).$

By this approach, we can also easily quantitative evaluate the nn5 feature extractor by using its classification accuracy as the metric.

Vectorization

The vectorization is a bit different than normal embedding layers. Because we would like the word vector can preserve some rhymes features. Thus, we used a Ma-transformation as mentioned in [P. Takala. Word embeddings for morphologically rich languages. In Proceedings of European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning, 2016.]. The word transformation algorithm embeds a word into a vector of 26 dimensions, where each dimension represents an English character. i.e., $\boldsymbol{w} = [w_a, w_b, w_c, \dots w_z]$

and for each character. $w_a = \sum \frac{(1-\alpha)^{c_a}}{Z}$ where c_a is the index of the character in the word, α is the hyperparameter in range $(0,1)$ set to 0.6 and Z is the length of the word.

Using this transformation, we can embed a word into a vector that including its spelling features, as suggested in the paper.

Network Architecture
The network should take the word, line, and the lines into consideration. We tried three network architectures.

I. NN5 original architecture (as in the paper)

The model needs fixed input shape, so we limit the number of words in a line to be 13, removing words to keep only the first 13 words or pad zeros at the end for short line. This is done for each vector. In addition to pad the line, we also need an input of the previous 5 lines. If there are not enough, we pad the line will all zeros. Therefore, the shape of input of the network should be $(Batch\ size, Number\ of\ lines\ = 6\ , Number\ of\ words\ in\ a\ line = 13, Word\ embedding\ dimension = 26)$

The model can be separated into 3 parts. The first part is to encode each word directly using 2 FC layers (500 neutrons each). Then we concatenate the words thus it is a line vector will high dimension that represent a lyric line. The line vector is encoded with another FC layer (256 neutrons). The encoded line vector is further concatenate into a vector representing the whole text, and then an FC layer (256 neutrons) is added to this layer and use softmax loss to determine the label. Each FC layer, except the last one, uses ReLU as activation function and 0.1 dropout after each layer.

II. Modified NN5

This model makes some modification by the original NN5 network.

The original model has some faults. First, the input features take the first 13 words and pad zeros at the end of line. For Hip Hop lyrics, the rhymes occur more likely at the end of line. Thus, instead of taking the first 13 words, we took the last 13 words and pad zeros at the beginning of line.

Also, we added a batch normalization layer after each FC layer.

Another modification is that using cross-entropy loss instead of softmax loss as it is a binary classification problem.

III. FC Network

The above two nn5 network are doing dimension expansion in the first layer (26 dimension to 500 dimension) and it is not making a lot of sense since decoder network is generally difficult to train and should not be having such a great contrast. And also the idea of using the same FC layers (shared layers) to encode the words and line was not behaving well because the words would be very noisy.

Also, the model is too deep, meaning it is harder to train.

The address the problems, we used 3 FC layers dense neural network directly on the training example and use Batch normalization and dropout is also applied to the 2 FC layer and we use cross-entropy loss as II.

<u>Relevance score calculation</u>
The relevance score is computed based on the RankSVM model. RankSVM is a variant of Support Vector Machine (SVM) model to solve the ranking problem. It is a learning retrieval function to employ pair-wise ranking methods to sort the results based on the relevance. In RankSVM, it

reduces a ranking problem to a classification problem by pairwise transformation. The pairwise transformation will first try to compute the difference between every pair in feature X of different labels as a new feature X_{new}. After that, it performs the classical SVM training on these transformed features and label. The rationale behind is to find a weight w such that

$$x_i \succ x_j \Leftrightarrow \langle w, x_i - x_j \rangle > 0$$

To train the RankSVM model, here are the steps.

1. Sample 10000 line samples from dataset
2. For each line, compute the feature using (LineLength, EndRhyme-1, EndRhyme-2, JaccardDist-1, JaccardDist-5 and TfidfDist-5) with 1. The next line and mark the label as true. 2. The randomly sampled line and mark the label as false
3. Perform the pairwise transformation
4. Shuffle the dataset and perform the train-test split for 8:2
5. Train the RankSVM model
6. Obtain the best candidate ĉ from the ranking. If the best candidate and the last line in B are not rhyme or ĉ is already repeated in B, choose the second best until it fulfils the requirement.

As the RankSVM involves the pairwise transformation which triggers $O(n^2)$ complexity, only subset in the data is selected for the model training. Once the pairwise transformation is solved, the remaining is just the normal SVM training that is more efficient.

Results and Evaluation

Ghost Writer Models

For the lyrics generation of n-gram and LSTM model, we have chosen two artists, Fabolous and Drake to generate lyrics separately to simulate two different rap styles. The reason of choosing them is that they have a reasonable training data size with 210 songs and 332 songs respectively. We then picked 8 lines of lyrics from each artist and use the first three words as the seed to generate lyrics. For each model, we predict the next 15 words to produce a line of lyrics. Then we compute the average scores of the rhyme density and similarity of all the lines to evaluate our model.

<u>Result of N-gram model</u>

We have created three N-Gram models, 1, 2 and 3-gram model as our baseline model to compare with the LSTM model. Below is a sample of lyrics of two artists generated by the n-gram model:

Sample 1: Seed "You can give" from Fabolous's lyrics

Fabolous	
Real Lyrics	You can give me brain when I'm free - scholarship. If you had me open on the head - hollow tip
1-gram model	you is how you can not the steering wheel one night it every night b.i.t.e. bitch
2-gram model	you can not do the kind of incident the black keys getting slicker get your hand on
3-gram model	you can give me brain when i am there, the hypno come out the place but you know

Sample 2: Seed "And she gives" from Drake's lyrics

Drake	
Real Lyrics	And she gives me all her trust and I'm abusing it right now with this money coming in it's just confusing shit right now

1-gram model	and we supposed to god uhh, wake up i would work work hard to go so
2-gram model	and she back back i am a let us get it then tell them from the free throw
3-gram model	and she gives me all her trust and i am feeling while you holding me begging to take

Intuitively from the results, we could see that the higher the value of n, the higher the similarity to the real lyrics. As the n-gram model predicts the occurrence of a word based on its previous n words, the larger the n is, the more likely the model generates lyrics that is similar to the real lyrics since the longer word sequence is unique. However, from the results, we could see that the n-gram model is hard to capture the rhyme feature. For a more in-depth analysis, we would discuss and compute the rhyme density and similarity scores of all the generated lyrics in the next section.

<u>Results of LSTM Model</u>

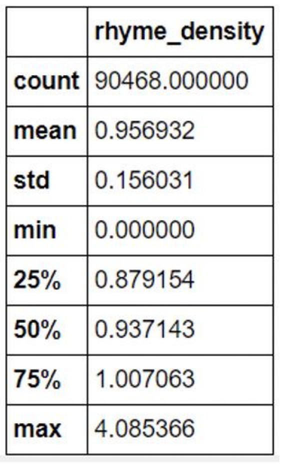
For LSTM model, we trained the LSTM model with 400 epochs and added early stopping so that model stops training when a monitored metric has stopped improving. As we would like to experiment with different LSTM models to see which approaches is better, we have built two LSTM models, one model has only 2 layers whereas the other has 3 layers with increased number of units in each layer.

Below shows a sample of the generated lyrics.

Sample 3: 8 generated lyrics of Fabolous

line 1: yeah we hop in the range she off top with the brain got the windows down broad day, line 2: you know i am at the studio man i got the fucking call it a condo one ay, line 3: they be at my neck like when i spray on cologne fields hoe sittin on the bannon grill, line 4: you can give me brain when i am free scholarship differ around broads what over here round me, line 5: i am not a fan of phony vibes and weird energy over top four three chains for me, line 6: you blowing on that bright side mean skies be dark grey anyday of never had someone but they, line 7: and i tell the cops this joint is to do things all speak you all look good gangster, line 8: feels better when you let it out do not it girl not look to me so hot she

The rhyme density and the similarity score of this verse is 0.7019 and 0.36 respectively. Among 90468 songs in our database, the mean and the 25 percentiles of the rhyme density are 0.9569 and 0.8791. In terms of rhyme density, the generated lyrics are not good enough for rapping.



	rhyme_density
count	90468.000000
mean	0.956932
std	0.156031
min	0.000000
25%	0.879154
50%	0.937143
75%	1.007063
max	4.085366

Table 2 distribution of rhyme density among all songs in our dataset

Simple and deeper LSTM models comparison

We wanted to train different models in different ways to compare the result and the training time of each model. One of them is a simple LSTM with regular preprocessing methods and another one is a deeper LSTM model. Here is the results:

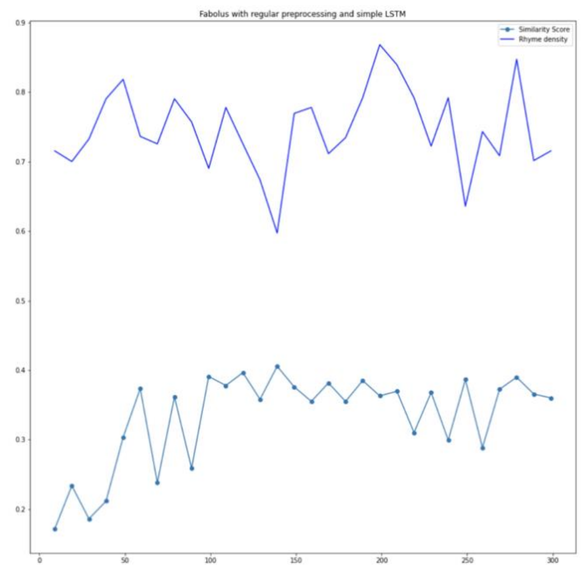


Figure 7 rhyme density and similarity versus number of epochs when training the simple LSTM model

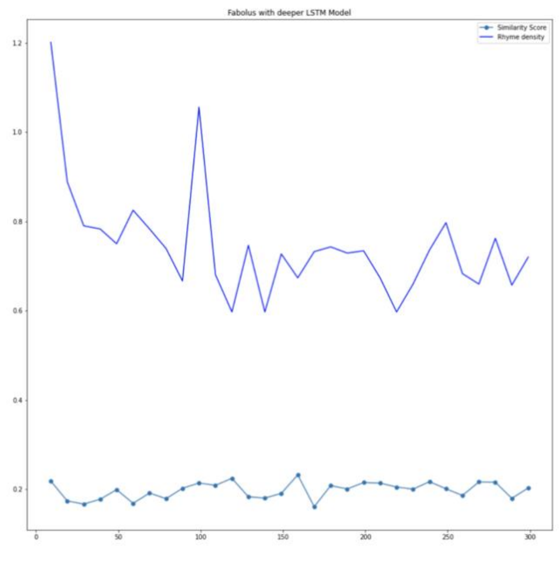


Figure 8 Rhyme density and similarity versus number of epochs when training the deeper LSTM model

From the above result, we can see there is no significant improvement in rhyme density of generated lyrics from deeper LSTM models. Although there is smaller similarity score which makes the generated lyrics novel, the training time of the deeper LSTM is extremely long and it is not worth it for spending this amount of time to have no significant improvement in lyrics generation. Therefore, we will only perform quantitative analysis for the results from n-grams models and simple LSTM models in next section.

<u>Quantitative Analysis</u>

Our goal of this project is to mirror an artist's style, at the same time providing original content. Although vocabulary and lyrical content are key components to evaluate an artist's style, this is satisfied by using words from the training data. As a result, in our project, our evaluation matrix would mainly be the rhyme density and the similarity as the key performance indicator for imitating an artist's style. A higher rhyme density is often better, while we hope to have a lower similarity score as it implies higher novelty. Therefore, we are trying to achieve a model with a high rhyme density, but a low max similarity score.

The below figures show the graph for rhyme density and similarity for n-gram and the LSTM models respectively. For the n-gram model, they are graphed dependent on n-gram value. For each n-gram value, we generate 8 lines of lyrics and compute the average value of the two metrics. From the two graphs of n-gram model, we could see that the similarity increases with the value of n of n-gram model increases. Thus, it is not a good choice when we have a higher value of n as our n-gram model in terms of similarity. While in terms of rhyme density, the rhyme density is similar among the 3 n-gram models, it may because n-gram model predicts words based on the probability given previous n words and it fails to capture the rhyme feature.

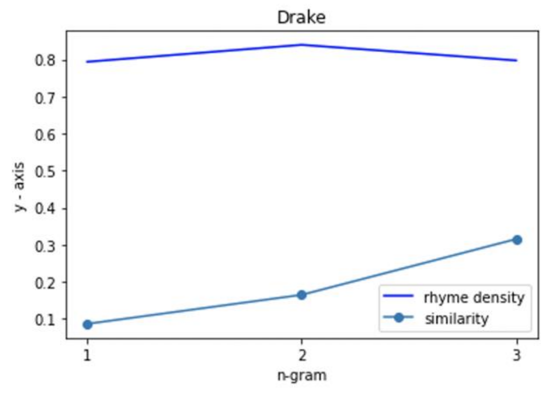


Figure 9 rhyme density and similarity versus Drake’s n-gram model

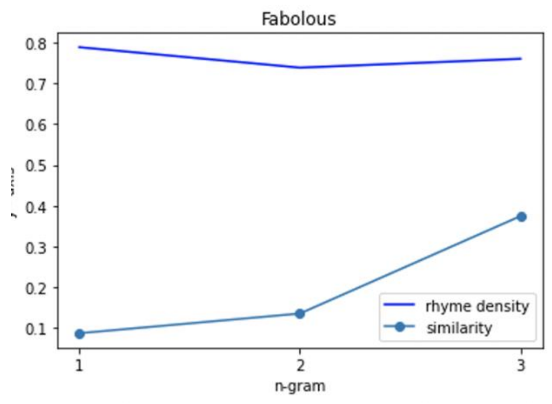


Figure 10 rhyme density and similarity versus Fabolous’s n-gram model

The below two graphs shows the results of LSTM model. The values are graphed compared to training iteration number. The optimal training iteration number would be around 180 to 200 epochs, as the correlation between the rhyme density and the similarity is lowest for both the artists. While when compared the LSTM model between Fabolous and Drake dataset, we could see that the similarity of Drake dataset is lower and thus giving a lower correlation between the rhyme density and the similarity. The reason behind could be the difference in the training data size.

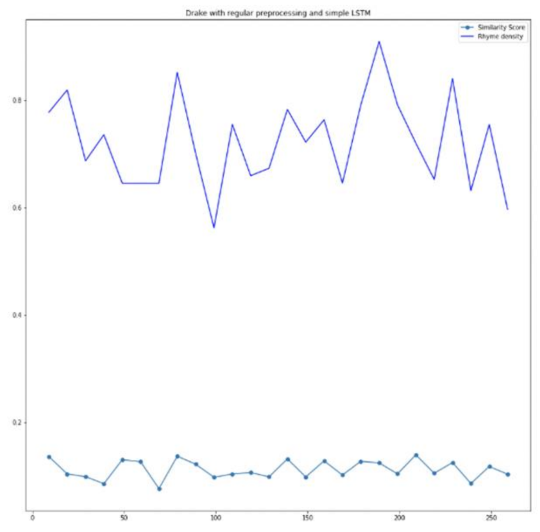


Figure 11 rhyme density and similarity versus number of epochs of Drake’s simple LSTM model

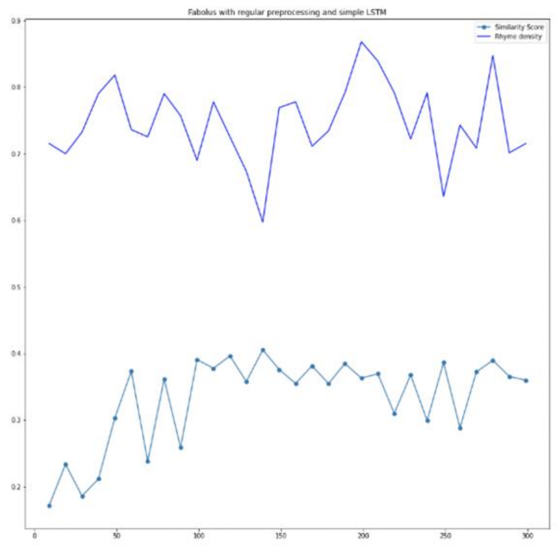


Figure 12 rhyme density and similarity versus number of epochs of Fabolous’s simple LSTM model

Dope Learning Model

Language Model accuracy

RankSVM’s accuracy

The RankSVM is tested by 2000 randomly samples and the test accuracy is about 75.2%. We consider it to be a satisfactory result which is better than the accuracy

(55%) of the original NN5 algorithms proposed in [4] tried in this project. It shows that the features used (LineLength, EndRhyme-1, EndRhyme-2, JaccardDist-1, JaccardDist-5 and TfidfDist-5) are sufficiently indicates the next line's fitness. However, due to the computational complexity by pairwise transformation of features spaces, a large number of candidates are dropped in the candidate retrieval section. Since the candidate retrieval section only relies on the EndRhyme criteria while a real rap song may not rhyme in every consecutive line, large amount of the candidates which are already dropped may have a better prediction in the RankSVM.

Apart from the accuracy by model's testing, samples are generated to assess the 1. Rhyme density 2. Semantic similarity

Sample 1: Seed "Professor Raymond is teaching data mining"

line	artist	source
Professor Raymond is teaching data mining	None	None
throw your cash that is cake and icing	DJ Felli Fel	Go dj thefiner.fel
all the greed they supplying	Non Phixion	Future suicide.non
we the base runners who beat out the throw they providin	Fabolous	Motivation
ya whole world is tremblin like aero planes collidin	Klashnekoff	Hesagas parrow.kla
sittin in my high chair throwin cake and cryin	Gym Class Heroes	Like Father, Like Son
the food came before i could take a bite in	Fabolous	I Miss My Love
heads always collided	Black Sheep	Rm bside onthwall.bsp
but you inspire me so i had to break the ice in	Troy Ave	BSB vol2 hey luv.ave
this rap race is no place to raise a child in	Murs	For pres erything.mrs
i am in that dropper with juve the groupies boppin and fiendin	Juvenile	Way I Be Leanin'
got nickels and diamonds we rollin and lightin	Denzel Curry	32 zel chieffor.cur
i came off from scammin and swipin	Lil Yachty	Nuthin 2 saintYSL.yac
young nigga i am trapping and grinding	Migos	nolabel2 ounces.mig
we gettin this paper let is keep the cheddar stacking and piling	Busta_ Rhymes	Party is Goin' On Over Here
out here just hustlin and grindin	Scarface	2 Real

Rhyme density: 1.621

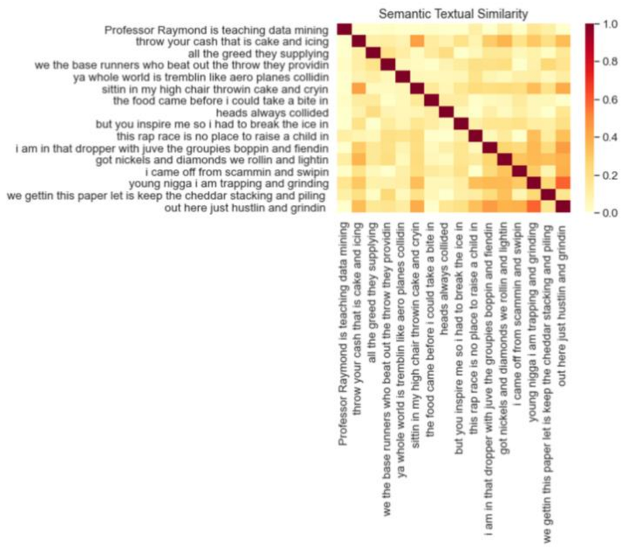


Fig :Semantic similarity heatmap of Sample 1

Average semantic correlation: 0.215

Sample 2: Seed “Rap lyrics generation is funny”

	line	artist	source
0	Rap lyrics generation is funny	None	None
1	could not pull away so easy seperation is ugly	Sen_Dog	Gangsta_Music
2	what is goin on in a dis country	Big_Daddy_Kane	W.G.O.N.R.S
3	had a bright idea now my bracelet is sunny	Chamillionaire	I'd_Rather_Get_Bread
4	the cards on the table gin rummy	Chief_Keef	Cut_the_Check
5	but once they get a lil money	Starlito	blksheep_no_TWO.str
6	he will get a lil money	Petey_Pablo	diary_of_test_of.pab
7	no black person is ugly	Lil_B	ultimate_no_black.lil
8	my cape is still tore up but i make a lil money	Redman	Soopaman_Luva_5_(Part_I)
9	money is over a bitch money	Migos	culture_brownbag.mig
10	in a blunt cuz a nigga had to stretch the figgas money	Koopsta_Knicca	Shut_Ya_Mouth,_Bh
11	when i am grindin niggas might see me as dressed a lil bummy	Busta_Rhymes	Like_Ohh
12	now she got a big tummy lookin like a big dummy	Poison_Clan	mental_all_they.cln
13	turnin into robots the devil is money	Lil_B	im_gay_unchain.lil
14	i told the bitch it is just money the road can get a lil bumpy	Lil_Wayne	I_Feel_Good
15	ohhh shorty said you want a playa with money	Akon	freedom_hollahol.akn

Rhyme density: 1.468

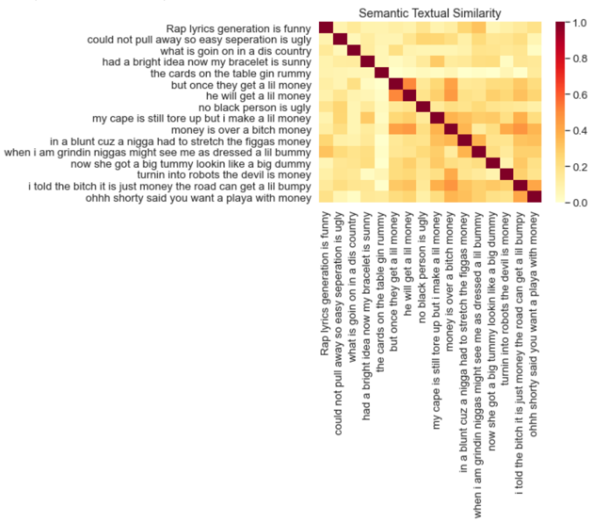


Figure 13 Semantic similarity heatmap of Sample 2

Average semantic correlation: 0.234

Sample 3: Seed "I go to school by bus"

	line	artist	source
0	I go to school by bus	None	None
1	i am in a bad place so who do i trust	Ill_Bill	www_bill_www_bill.ill
2	i prefer to do light drugs	Future	Damage
3	yeacoming to you live from	Lil_Wayne	Money_to_Blow
4	hot like curry food boo like chucky	King_Just	no_power_zero_tol.js
5	i give it to you like uhh	Nelly	Baby
6	i am sending you you you my love	Layzie_&_Bizzy_Bone	bonebro3_lock_rmx.bne
7	me bring the noise to you like chuck	KRS_One	Mad_Crew
8	aye aye you need to do my club	The_Grouch	p_sense_zip_it.grh
9	nigga this my party where do you live huh	Gucci_Mane	Block_Party
10	put you on punishment the same way i do to my son	Big_Punisher	Pina_Colada
11	but do you like drugs do you like drugs	Miguel	k_dream_do_you.mig
12	from c ward to you nigga uh	Chris_Ward	vet_rook_my_life.wrd
13	now who do i trust	Flow_Click	flwclick_indagame.flw
14	i used to look up to you nigga uh	Rick_Ross	Idols_Become_Rivals
15	do you like drugs do you like drugs	Miguel	k_dream_do_you.mig

Rhyme density: 1.984

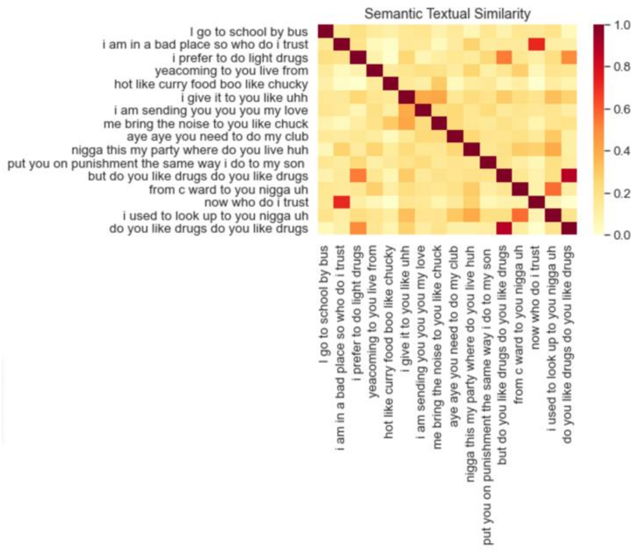


Figure 14 Semantic similarity heatmap of Sample 3

Average semantic correlation: 0.231

Rhyme density

From the above three examples, the rhyme densities are all larger than 2 standard deviations from the mean of rhyme density in the dataset. Therefore, the lyrics generated is considered to be capable to generate multisyllabic assonance rhyme lyrics which is unique characteristic in rap song. From the pronunciations, it can also be verified that the line generated are rhyme. It is expected because the EndRhyme criteria are considered in the candidate retrieval and the RankSVM model training. Not just the consecutive lines, sometimes it is discovered that the line even rhymes more with the lines before the past line. It can be explained by the features used in the RankSVM model. In the RankSVM model, features (EndRhyme-2, JaccardDist-5 and TfidfDist-5) are used and they capture the similarity of non-consecutive lines. As a result, these features tend to have positive impact for the line generation and the model tends to choose the best candidate from the line which is close to the last preceding k lines instead of just last line. However, feature LineLength does not contribute too much in the model as it can be observed that the lengths of lines generated has large variety.

Semantic meaning

It can be observed that occasionally the line generated is highly correlated to the preceding lines, but it does not always hold for every line. Therefore, the semantic correlation score is only about 0.23. In the dataset, it can be observed that the semantic correlation is generally not high and lays between 0.2 to 0.3. In fact, the candidate retrieval and NN5 is done in advance of the semantic correlation filtering. The former one emphasizes on the rhyming and the latter one focus on the semantic meaning. It becomes a trade-off that if the line generation algorithm stresses more on the rhyming, it may eventually lose the semantically correlated lines in the candidates. It can be solved by increasing the candidate size in the candidate retrieval stage so as to have a higher chance to have more semantically correlated candidates.

However, in the dataset, it is observed that the nowadays songs are generally having low semantic correlation between lines. For example, the semantic correlation matrix of rap song Sixteen Switches sung by rapper Currently has the average semantic correlation: 0.206. It reflects that semantic meaning may not be as important as the rhythm for a rap song.

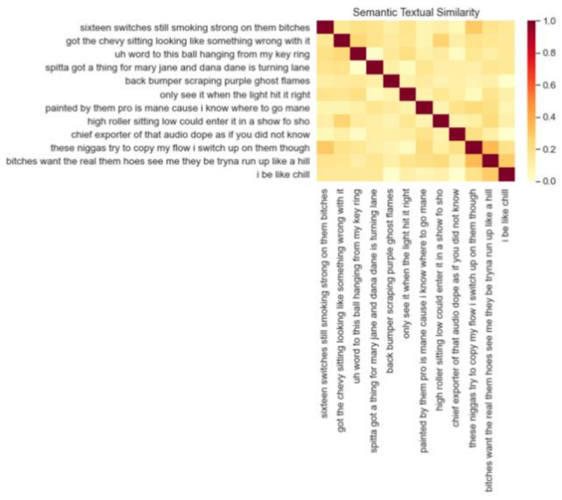


Fig :Semantic similarity heatmap of rap song Sixteen Switches by rapper Curren$y

Conclusion

In this project, we have tried three different approaches to generate lyrics. It is shown the effectiveness of an LSTM model of generating lyrics with novelty, that as an unsupervised non-template model, it is better to be able to produce novel lyrics when comparing to the n-gram model. While for dope learning algorithm, as it is using a candidate retrieval approach, it is difficult to produce original lyrics based on dope learning algorithm. Whereas, in terms of rhyme density, the dope learning algorithm outperforms the LSTM model as it captures the rhyme feature successfully in the RankSVM model.

Despite the good performance the models have, there are still limitations for each model that needs to be improved. For example, lots of preprocessing must be done for dope learning algorithm to capture the rhyme feature, and it is computational exhaustive for LSTM model.

Future works

In our future work, we may consider utilizing both the models to preserve the rhyme pattern at the same time generating new lyrics. In the paper by Chen, they mentioned that some modifications of current LSTM models may get 90.7% training accuracy and 87.5% testing accuracy compare to our 80% training accuracy (Chen). By using these kinds of LSTM model, we will simply improve our overall results. It is also possible pretrained model like BERT (Devlin) could be used to decrease the training time of the LSTM model. Currently, it took 12 hours to train on dataset with 1000 records on Nvidia P100 GPU. It is very time consuming on a larger dataset, for example 10000 records in it. In the future, we want to speed up our training as well as improve our overall performance so that the users of our app can really enjoy the output from our algorithm.

Contribution

Ko Wing Fung
In this project, he mainly implements the improvement work from the Dopelearning paper. He performs the features generation from the rap lyrics. The features include the EndRhyme,LineLength, JaccardDist, RhymeDensity, UseDist, TfidfDist. In each of them, tailored algorithms are implemented by his own code and some features are implemented in addition to the original paper. For example, TfidfDIst and UseDist which requires advanced embedding Universal Sentence Encoder to obtain the semantic meaning, compared to the ordinary LSA method which proposed by the paper does not work well. Meanwhile, the RhymeDensity is used to assess the lyrics generated by DopeLearning and Ghostwriter.
Furthermore, he also performs the phonetic transform and data processing to the big data set, with the achievement of both time efficiency by multiprocessing technique and space efficiency by dimension reduction. Phonetic vectors sorting and binary search are implemented for fast lyrics query among the big dataset. In addition, improvement from the original DopeLearning algorithm is also done by having the semantic meaning filtering and the Levenshtein distance comparison to enhance the diversity of rap lyrics generated. Apart from the algorithm implementation, he also involves in the model RankSVM's training, which requires extra research effort to understand the principle and the usage. Fine-tunning for the model parameter is done to optimize the test accuracy to 75.2%. He also performs the analysis for the result of generated lyrics like the semantic correlation and the RhymeDensity. Lastly, he performs the software integration with the nn5 algorithm developed by the other member to complete the entire DopeLearning algorithm.

Muhammad Adam Usmani
Adam carried out preprocessing steps for the GhostWriter models including lower casing the lyrics, converting all digits into text, cleaning the dataset by removing meaningless special characters and non-ascii characters and converting the words into more meaningful English words by creating a contraction map. He also carried out research into different variations of the LSTM and baseline models used in the GhostWriter paper. He trialed a range of different character level and word level models for the LSTM model and did the same for the baseline model by trialing different Markov models. Adam set up the LSTM and baseline models for the implementation task. Set up involved preprocessing, creating the training data and building the model for LSTM and doing likewise for the baseline model. Adam trained the LSTM model with the allocated data over 150 epochs. Adam also selected certain lines from the lyrics and used seeds from these lines to generate new lyrics for the artists of interest. In addition to these tasks, Adam organized weekly meetings with the team and helped to guide the path of the GhostWriter implementation workflow.

Koo Tin Lok
My major responsibilities are for finding dataset implementing the language model for Dope Learning. I explored different public dataset and decided to gather our own data, I scraped data from the lyric website and done the first part of filtering and cleansing. In addition, for the modelling part, I have implemented LSA feature extraction and NN5 feature extraction as mentioned in the Dope Learning paper. My major contribution is the NN5 feature extractor which includes word vectorization and the language modelling. I have tried to replicate the

result of the nn5 feature extractor as the paper did but fail to achieve their result. Thus, I tried to improve the result by using different word transformation methods and improving the model architecture.

Qiwei Li
In this project, I did some researches on how to improve the final LSTM accuracy and trained 4 different models with 400 epochs for each model. Also set up few notebooks on colab to accelerate the training process as well as can easily share the checkpoints generated by models to others. This better environment for teamwork. In order to analyze the relationship between the epochs with lyrics rhythm density and lyrics similarity similar to the original ghost writer paper, I generated similarity versus epoch graphs for all models. Finally, I wrote a preprocessing library in order to test different preprocessing techniques and their impacts on the final training results. I am very appreciating the works done by my teammates and that helps a lot when I am doing my training as well as the final evaluation part.

Shiu Stephen
Before the topic was finalized, I did some research on different kinds of machine learning projects like sentiment analysis on tweets, Taylor Swift's Song Lyrics Generator project on Kaggle. After our 1st group discussion, we confirmed our topic, and I was responsible for writing the preprocessing part of the proposal. At the early stage, I mainly focused on the feature extraction part of the Dope learning model such as calculating rhyme density of a verse, computing the similarity between sentences. After that, I focused on the evaluation part of the n-gram models and simple LSTM models. I was responsible to evaluate the generated lyrics from these models by different metrics such as rhyme density and similarity score. I computed the scores and plotted graphs using python which are shown in our final report. In the final report, I mainly focus on writing the evaluation part of n-gram models and simple LSTM models. In the whole project, we had weekly meetings to keep monitoring our progress and all of us did attend all meetings without absence.

Ng Pui Yan
In the beginning stages, we all did a research on lyrics generation and proposed various approaches on the lyrics generation. After we decided our project direction, we have studied the methodology of the papers for our implementation. As we use a large dataset of 90468 songs, we processed the data preprocessing in parallel to speed up the preprocessing task. I have also contributed on training the LSTM model for Drake's lyrics, and generated the results on multiple checkpoints for further evaluation process. Finally, I helped on evaluating the result for the models and drawing a conclusion on the models implemented.

Bibliography

[1] The Anthology of Rap, Adam Bradley: http://docshare01.docshare.tips/files/26185/261857217.pdf

[2] The CMU Pronouncing Dictionary: http://www.speech.cs.cmu.edu/cgi-bin/cmudict

[3] How to Write a Spelling Corrector: http://norvig.com/spell-correct.html

[4] DopeLearning: A Computational Approach to Rap Lyrics Generation: http://www.kdd.org/kdd2016/papers/files/adf0399-malmiA.pdf

[5] Universal Sentence Encoder: https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/46808.pdf

[6] Markov Chains Train Text Generation: https://www.kdnuggets.com/2019/11/markov-chains-train-text-generation.html

[1] Poetry Generation LSTM

https://github.com/agrawalp

arth08/poetry-generation-lstm/blob/master/Poetry_Generation.ipynb

Chen, Q. (n.d.). Enhanced LSTM for Natural Language Inference.

Devlin. (n.d.). Bert: Pre-training of deep bidirectional transformers for language understanding.

Gupta. (2013). A survey of common stemming techniques and existing stemmers for indian languages. *Journal of Emerging Technologies in Web Intelligence*.

Kamiran. (2012). Data preprocessing techniques for classification without discrimination. *Knowledge and Information Systems 33.1*.

List of dialects of English. (2021, 4 17). Retrieved 4 17, 2021

nalepae. (n.d.). *nalepae / pandarallel* . Retrieved 4 17, 2021, from https://github.com/nalepae/pandarallel

TextBlob. (n.d.). Retrieved 4 17, 2021, from https://textblob.readthedocs.io/en/dev/

Vijayarani, S. (2015). Preprocessing techniques for text mining-an overview. *International Journal of Computer Science & Communication Networks 5.1*.

Wikipedia. (n.d.). *Spanish language in the United States*. Retrieved 4 17, 2021, from https://en.wikipedia.org/wiki/Spanish_language_in_the_United_States