Natural Language Generation Report - Lil A.I. *

Mike Nelhams
oa18502@bristol.ac.uk

Joshua Ukairo Stevens
tx18073@bristol.ac.uk

Matthew Forster
zc18711@bristol.ac.uk

James Groeneweg
kr18544@bristol.ac.uk

April 6, 2022

1 Introduction to the Problem

Originating from 1970s Hip-Hop culture in the Bronx, New York [1], rap music has become an extremely popular and established genre of music. In essence, rap may be described as music that assimilates *"rhyme, rhythmic speech, and street vernacular"*[2]. Thus one may describe rap as being a form of 'street poetry'. The rhythmic structure found in rap makes the creation of new rap lyrics an interesting task and distinguishes it from other genres of music.

Our intrigue in the task stemmed from a group interest in investigating the possibilities of natural language generation using machine learning. As such, artificially generating rap lyrics seemed a natural fit.

1.1 Project Definition

Our major objective is to create a convincing rap lyric generator. Two of the main elements to rap lyrics may be thought of as the meaning and the sound of the text.

The two main methods explored in order to create rap lyric generators are achieved through the utilisation of Markov Chains and Recurrent Neural Networks. Each method has been explored and used.

The only necessary input to the systems that will be developed is a vocabulary for the systems to use. One of the clearest ways in which to create a vocabulary for rap lyrics is to create a vocabulary from rap lyrics themselves. This does raise the issue that words of offensive nature may be inserted into the system. For this pipelines will be required in order to 'clean' the vocabulary before or after use. As a practical matter, punctuation and capital letters are needed to be removed and made into lowercase respectively in order to be able to use each word in the vocabulary throughout each lyric.

The output for the systems will be a string of text of the rap lyric generated. In order to test these outputs multiple methods of classifiers have been created in order to determine how successful each generator is.

1.2 Desiderata

As requested, we shall attempt to define 'fun' and persuade you that our problem is, in fact, fun.

This will be done by presenting you with a rap, generated by one of our generators:

'As I kiss your bum goodnight,
You nasty boy you nasty,
Squeeze your clip hit the right one pass that sandwhiches I got to light one,
Is the bad bad boy,
Stepped to police when I proceed'

*The Github repository for this paper: https://github.com/EMAT31530/ai-group-project-Team-JMJM

2 Literature Review

2.1 DopeLearning

The report *DopeLearning: A Computational Approach to Rap Lyrics Generation* [3] encapsulates what is defined to be two of the main features of rap: complex rhyme patterns and meaningful lyrics. The writers develop an algorithm `DeepBeat`, which in essence, is a rap lyric generator in which these outlined features are achieved.

The algorithm `DeepBeat` explored in the paper uses an *information retrieval* approach. From a database of over 104 different rap artists lyrics, accounting to over half a million lines, the algorithm finds what is termed a *relevant next line*. In doing so it is assumed that each lyric line from the database of over half a million can be pieced in a way that gives a relatively meaningful lyric structure. This involves the supposition that each line of a rap song's lyrics will contain some partial meaning. If two lines do not necessarily match up with continuous meaning, then this can also lead to what may be deemed as *computational creativity* or novelty.

The rhyme type utilised within `DeepBeat` is the *assonance* rhyme, which may be defined as: *"resemblance of sound between syllables of nearby words, arising particularly from the rhyming of two or more stressed vowels, but not consonants (e.g. sonnet, porridge), but also from the use of identical consonants with different vowels (e.g. killed, cold, culled)"* [4]. For implementing this, a rhyme density measure is used in which consonants and spaces are ignored and vowel sequences are found. This method is built as a deep learning neural network model in which after each line is classified and they are ranked using the RankSVM algorithm to choose the next line of the rap.

The paper explores an algorithm that produces convincing rap lyrics with assonance rhyme structure and often a conception of meaning. As assonance rhyme is the most commonly used rhyming method within rap [5] it is unlikely that the only use of rhyme within the lyrics being assonance rhyme will be a large limitation.

The method used, however, may lack in some senses for having true novelty. As the database used counts over half a million lines of lyrics, one should not notice this, however the algorithm does not come up with truly original lines of lyrics. In order to improve the method explored, further work implementing a similar method to each word in the lyrics could be utilised.

2.2 The Case of Rap Lyrics Ghostwriting

Evaluating Creative Language Generation: The Case of Rap Lyric Ghostwriting [6] has the goal of being an artificial 'ghostwriter'. Ghostwriting has always been prevalent in the creative industry and music is no exception. For years, major artists have had their songs written for them, but the goal is to create lyrics that the target artist would write, whilst also remaining unique.

NLG has grown over the past few years, but the difficulty has always been in evaluating the generated text because there is no 'perfect' metric for doing so (see Chris van der Lee et al. 2019 [7]). Using a wide range of different rappers and styles, the main goal of this paper is to develop a way to evaluate the generated lyrics, through fluency, coherence and style matching, ensuring that generated verses are similar in areas such as rhyme frequency and style. This is done so using both manual and automatic methods. At line level, annotators are given the task of measuring level of fluency (how fluent is the line), and coherence (how well does the line match the previous line), with different levels of fluency and coherence given different weightings. In the automated sense, generated verses are analysed against training verses to check for novelty.

The paper definitely has its strengths. The evaluation methods are very original and offer what so many NLP projects lack, a robust evaluation method that can be used in many different contexts. The authors realise the need for some way to analyse the creativity and distinctiveness of the generated text, rather than just analysing how accurate the generated text is. The wide range of different rappers also mean that there is some ability to be distinguished between the generated verses, that are in turn based off certain metrics.

However, there are certainly some drawbacks to the method. Not much consideration is given to the length of verse or line, which can be key to certain rappers. Furthermore, there could be an argument to be had about age/experience. No rapper keeps the same stylistic patterns throughout their career so there could be some way to break this up and weight certain verses. Some of the more recent rappers, verging popularity, also use *adlibs* and this could be an interesting idea to add into a similar project. In this sense, common adlibs could be singled out and be implemented into generated verses for each artist.

2.3 Automatic Neural Lyrics and Melody Composition

Automatic Neural Lyrics and Melody Composition [8] looks at the process of generating both melody and lyrics with a neural network model consisting of a lyric generator, a lyric encoder and a melody decoder. They trained several of these encoder-decoder models to find correlations between lyrics and melodies. Overall, their model can generate lyrics and corresponding melodies from a seed lyric inputted by the user, or can generate a melody for complete lyrics the user inputs.

The proposed model included a lyrics generator which worked by predicting the next lyrical token given the previous lyrical token, starting with a seed lyric inputted by the user. A recurrent neural network (RNN) was used to do this, and a non-linear function was applied to map the network states over time. This function can be a simple sigmoid function or a more advanced function such as a long short-term memory (LSTM) unit. The function is trained on a dataset of 12,023 MIDI files of "popular English songs" and looks at the relationship between the lyrics and the melody on the syllable, word and sentence level.

The paper uses a fairly basic method of lyric inference. At each step of lyric generation the paper uses a softmax probability distribution to obtain the most probable next lyric given the previous lyrics (a user inputted token is used as the first lyric). Furthermore, a freezing function is applied to the possible lyric tokens using a controllable parameter - this introduces noise to predict more robust and diverse music and is something we could consider in our modelling.

Perhaps the biggest problem with this model is that there is no attention paid to the structure of the song - there are no choruses/verses etc. generated, instead just a string of words and sentences. This also means that the piece of music as a whole makes little musical sense, as the notes are tied to the syllables and words are generated or inputted with no attention paid to either the structure or the key of the piece. Finally, only 11 responses to the model were gathered, which is unlikely to be representative of the wider populace.

2.4 Lyric Generation with Style

In the report *Lyric Generation with Style* [9], the aim is to produce lyrics that match some style and some topic given some input using a GAN. A combination of a generator, an encoder and a discriminator are used to produce lyrics, embed the lyrics, and ensure the generated lyrics are similar to the real ones.

The dataset is composed of 380,000 lyrics with a large number of styles and topics. This was preprocessed in a standard way, with abbreviations and punctuation removed and the data was split into a 80%, 10%, 10% training, cross-validation and testing split. Important to note, is the roughly equal split in genres in the dataset. word2vec was used in order to generate a mean vector over every word in a topic, so that generated lines could be compared to the supposed topic so it could be seen how close they were to this topic.

A hierachical structure is used in both the generator and the encoder, with two GRU RNN's, a line-level decoder and a word-level decoder being utilised. This hierarchical structure is useful in that line-to-line relations are captured as well as relations between words. Furthermore, loss was looked at on a word level and on a line level to measure the difference between the generation and the ground truths.

In terms of evaluation, two different approaches were used. This was a *Discriminator Approach*, measuring the 'realness' of generated lyrics with the trained discriminator, and a *Classification Measurement*, to check if the generated lyric matches

the style given.

As for the drawbacks, based off the lyrics produced, it appears that more hyper-tuning could have been done to produce more human-like lyrics. This is a large part of any model and a lot of time is required to try and find the best combination of parameters. The generality of the model also leads to more basic, general lyrics. This is opposed to a deeper look into a specific genre, where one could have made more realistic lyrics for the genre. Most genres do not match exactly in lyrical style.

2.5 Everybody Sign Now

Everybody Sign Now: Translating Spoken Language to Photo Realistic Sign Language Video [10] is a recent paper which presents '`SignGAN`', a Sign Language Production (SLP) model that outputs realistic images of sign language from an input of spoken language. Whilst this paper focuses on a different problem to what we explore in our paper, it does provide some interesting ideas for what could be considered as future work.

`SignGAN` uses a Mixture Density Network (MDN) in order to model the different styles of sign language. Skeletal pose sign language is generated from this method. Loss functions are also utilised in order to restrict any motion blur issues.

Whilst this has no direct applications to our work, methods such as this can be implemented on our rap lyrics. As our lyrics are in text form, this will decrease one of the tasks in the algorithm, however, we may implement some form of text-to-speech algorithm such as *Google Cloud, Text-to-Speech* [11] in order to have our model 'rap' and possibly sign!

3 Method

3.1 Data Preprocessing

3.1.1 Collecting rap lyric data

The initial procedure is to collect and preprocess royalty-free rap lyrics. The ideal workable format for the input data, is the standard '.txt' format. All of the rap lyrics have been collected from The Original Hip-Hop Archives [12] and another small resource from Kaggle [13]. Since the majority of the punctuation is meaningless towards the generated text, this is all preprocessed from the original data. **Show a small labelled example of punctuation in rap**

Censoring the data, whilst maintaining the semantics and pragmatism of the source content is necessary for publishing generated lyrics. To conserve the information, the output predictions are post processed using plain asterisk replacements, as seen in **EXAMPLE IN DESID**. Finally, a Github API was implemented in order to keep our training data updated in contrast to the corresponding repository data.

3.1.2 Censoring the lyrics

3.2 Markov Model

For our first model, we created a rap lyric generator using Markov Chains and the rhyming scheme described in our proposal. The model has three basic elements: a text generator, a rhyme ranker and a choice function.

Within Natural Language Processing, a Markov Chain contains a stochastic Markov process satisfying the Markov property whereby the probability of a word w_i is dependent on uniquely the immediately preceding word [14] [15]. Utilising this allows a model to form convincing sentences, however the absence of memory within the model restricts the amount of meaning each sentence will have.

The text generator creates a 'line' of rap lyrics using a Markov Chain trained on our vocabulary. The Markov Chain creates a dictionary of the distinct words of our vocabulary. For each unique word within our vocabulary (and so in our dictionary) the value of each word in the dictionary is a list of the words that are used immediately succeeding the key in our vocabulary. Thus the model learns the words that commonly succeed one another. After an initial random choice of first word for the line, the model will continue to choose the next word using the Markov Chain until it has reached the lines length. Here the length of each of these lines is defined as a random choice between 6 and 12.

The rhyme ranker used in the model is based on assonance rhyme (one of the most common types

of rhyme [3]) as described in our proposal. The mission statement of our rhyme ranker is from an input of two lines of rap lyric, outputting a numerical score that can be compared with other pairings of lines. Of the different rhyme rankers created and considered, the chosen function uses a counter which counts the number of matching syllables from the end of the line until the syllables at the same position from the end of each line do not match.

The choice function is the final step to the model. User input requires a choice of the number of lines of lyric required and the number of lines that are required to be created (this having to be greater than or equal to the number of lines required). Out of all of the possible lines generated, the choice function randomly selects a first line and then uses the rhyme ranker to choose the next line of the available generated lines that have not been used. In order to prevent every line rhyming with the end of the first line, the rhyme ranker resets each modulo 2.

This model generates amusing and fairly convincing results. It mimics the style of rap, albeit, without the text having any real meaning. As the Markov Chain lacks a memory property, caused by choosing each word using only knowledge of the previous word, no real substance is included in the rap. Alongside each line generated by the Markov Chain not carrying any meaning, the choice function purely chooses the next line by the amount of assonance rhyme found at the end of each line.

Algorithm 1: Markov rap generator

```
Result: Generate a line of rap, returns line
insert vocabulary;
index = 1;
create a dictionary called chain;
set count (number of words for the line) to
  be a random integer between 6 and 12;
for each word in the vocabulary do
    set key as previous word;
    if key is in chain then
        add word to key's dictionary;
    else
        add key to chain with word in key's
          dictionary;
    end
end
create line with a randomly chosen word
  from vocabulary while Number of words
  in line is less than count do
    next word is random choice of word
      from chain;
    add words to line;
end
```

Algorithm 2: Rhyme Ranker

```
Result: Score (counter) of assonance
        rhyme at end of each line
insert two generated lines of rap;
counter = 0;
i = 0;
Extract syllables of each line and reverse
  order;
Count is True;
while Count is True do
    if i is less than the minimum number of
      syllables for each line and the first
      syllable for each reversed line is equal
      then
        counter +1;
        i + 1;
    else
        Count is False;
    end
end
```

3.3 Syllable Counting

In practice, it may be useful for rappers to specify the number of syllables they want a line to be.

This allows for more precise lyrics generated to fit music. Generally rappers rap to semiquavers in standard time (that is, 16 syllables to a bar) [16] but for specific music they may want a specific number of syllables.

The text generator within the Markov model was edited so that it generated only the specified number of syllables per line. This was done by generating words as shown in Algorithm 1 until the specified number of syllables is reached or exceeded - if reached exactly, the line is used as normal and if exceeded, the last word is discarded and re-generated until the line is the correct number of syllables. If a word cannot be generated to make up the correct number of syllables, the entire line is re-generated until it fits the required number of syllables.

3 methods were investigated to predict the number of syllables per line. The first method used a syllable predicting algorithm found online [17]. The second used the Pyphen module [18] to hyphenate words and calculate the number of hyphenations, which is the number of syllables. Finally, a neural network was created with Tensorflow to predict the number of syllables of a given word. 2 text files containing over 40,000 words broken into their constituent syllables was found online [19]. This data was pre-processed by merging the files, counting the number of syllables each word has and creating a CSV file of words and their syllables count. Next, this file was split into training, testing and validation sets, in a 60-20-20 percentage split. Every word was broken down into a list of integers that represent its letters, so that the words could be represented as Numpy arrays. As the neural network required every list of letters to be of equal length, empty spaces were added to each word to bring them all to the same length - the length of the longest word, which was 16 characters. The neural network started with an Embedding layer and had 1 hidden layer. The network ended with a Dense layer, with a softmax activation function and 8 outputs, meaning that the network classifies words as being between 1 and 8 syllables. As the longest word in the data was 7 syllables this was reasonable, although in edge cases the model could fail to predict words with more syllables. The learning rate was set at 0.2 after some trial and error, and the number of epochs was set at 80. Gradient descent was used to optimise the model, with sparse categorical cross-entropy loss.

All of these syllable predictors were integrated into the Markov model and tested. They were tested on the validation dataset to avoid bias toward the neural network and to have a large sample size. The first model had an accuracy of 72%, the Pyphen model an accuracy of 63% and the neural network had an accuracy of 75%. This implies that the neural network is the best model, although there is just a small difference between the first method and the neural network. The neural network also has disadvantages over the other models. Firstly, it is much slower to predict syllable count and therefore takes much longer to generate lyrics. It will also to fail to accurately classify words with more than 8 syllables, due to its 8 output nodes, although these words are very rare, and cannot classify words with more than 16 characters accurately. The lyrics were censored after generation with this model, so that the model would categorise the words accurately. To ensure that the number of syllables were preserved after censoring the lyrics, the lyrics were censored to words that maintained the same number of syllables.

Using this model choruses and verses can be defined more clearly in a song, by defining the number of syllables per line based on the music. In conjunction with repeating the chorus where necessary, this allows a more musically accurate rap song to be generated.

3.4 Recurrent Neural Networks

In this subsection, we will discuss our rap lyric generator using bidirectional LSTM layers implemented via Tensorflow. In practise, LSTM models should provide more coherent, meaningful lyrics as opposed to Markov chains. The latter suffers from the vanishing gradient problem, meaning it has short-term memory. As a result, key information can be lost and this becomes more of a problem for large sequences. To combat this, LSTM's have gates which enable the model to retain key information, exempting them from suffering from short-term memory loss.

In order to create the model, words need to be converted into unique integers so that the model can train on the data, a process known as tokenization.

Hence, each unique word was given its own integer such each bar can be converted into a sequence of integers. This was done using two dictionaries, to both encode and decode bars. Next, these integer sequences were formed into n-grams of increasing length. Each vector needed to be of the same length, hence each sequence was pre-padded by 0s, so each sequence is now the length of the bar of largest length. In addition to n-gram vectorisations, each bar itself was padded with a start and end character, each with unique token IDs, so the model itself is able to predict when sequences should finish. The use of a Bi-Directional LSTM is also used [20]. This type of LSTM trains two LSTMs, one in the forward direction and one in the reverse direction. This improves the context as we have information about past and future states.

In order to train the model, the data was split such that for a sequence of length n, the prediction is a probability vector representation of one word from the vocabulary list. Following the input layer of the model is an embedding layer, which masks the padding 0s, since the additional padding provides no useful information. An embedding layer is used opposed to one-hot vector encoding, because 'the main benefit of the dense representations [word-embedding] is in generalization power' [20]. Embedding the n-gram inputs allows the model to better generalize to inputs and adapt to data similarities and patterns. With all of the models, an embedding layer was first used to convert from the n-grams into tensors for the model input. Lastly, the model was mini-batch trained using a number of different architectures and layers to fine-tune the model's different hyper-parameters.

Designing the model architecture and selecting its hyper-parameters was the most time-consuming process, because the model architecture and hyper-parameters affects whether the model creates somewhat meaningful lyrics that didn't overfit by always copying previous lyrics or underfit by repeating the same words. For testing the model hyper-parameters, the training/validation/testing split is 0.75/0.15/0.1, with the n-gram inputs shuffled uniformly.

The most simple decision for modelling was the choice of optimizer. Adam [21] is a fast, popular and semi-reliable optimizer that generally requires little hyper-parameter fine-tuning. A more stable and effective look-ahead adaptation of Adam known as Ranger [22] exists, however this optimizer is slower and requires over twice the memory, since it tracks more than twice the variables compared to Adam and Adam proved to be sufficiently stable.

It is understood that stacking LSTM layers can be risky if implemented incorrectly, and a heavy trade-off between time complexity, maximum validation accuracy and overfitting is incurred [20]. The idea behind deep (stacked) models, opposed to small, but wide layers is that the models can learn more advanced patterns with the same training time and with a lower risk of overfitting. The principle idea is that the model should consist of at least the following, shown below in Figure 1.

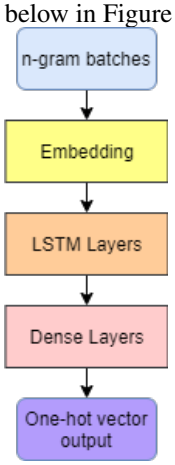

Figure 1: The most basic LSTM architecture.

3.5 Regularisation

The issue concerning generative recurrent neural networks is that the dataset input is always imbalanced. The frequencies of each word differs across the input data and this will always be the case for all rap lyrics. Tokenization models may struggle when attempting to predict words with low frequencies and if do they successfully predict uncommon words, this is often an indicator of overfitting. To combat overfitting, dropout layers where included after every LSTM layer. A dropout layer regularises the model, by decreasing the importance on specific weights and biases, by randomly ignoring them with likelihood of p per operation [23]. Dropout can decrease the dependency on individual neurons and slows down the learning process, which helps

prevent overfitting. **SHOW WITH AND WITHOUT DROPOUT**

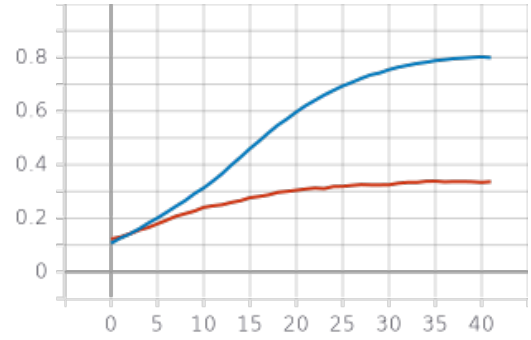

Figure 2: A simple model with an embedding size of 512, LSTM1 size 128 neurons, LSTM2 size 256 neurons and a single dense layer of the vocab length. No regularisation has been applied and it is clear that the model has begun converging early and the validation loss is capped at around 30%, but it is possible to perform better.

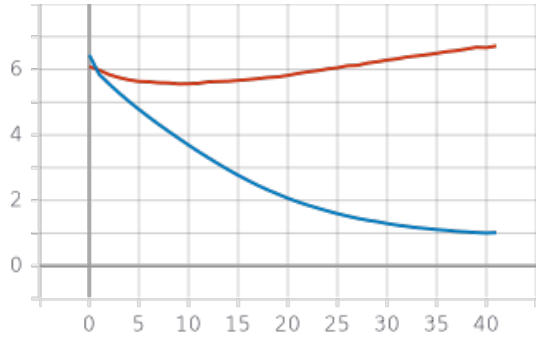

Figure 3: A simple model with an embedding size of 512, LSTM1 size 128 neurons, LSTM2 size 256 neurons and a single dense layer of the vocab length. No regularisation has been applied and it is clear that the model began overfitting, as early as epoch 8.

Another two forms of regularisation are L_1 and L_2, lasso and ridge regularisation respectively. Lasso and ridge regularisation are defined as the following for weights matrix $\mathbf{W}$:

$$L_1 = \lambda \sum_{j=0}^{M} |W_j| \tag{1}$$

$$L_2 = \lambda \sum_{j=0}^{M} W_j^2 \tag{2}$$

Lasso regularisation is great at zeroing out ineffective weights and biases, whereas ridge regularisation effectively reduces weights from overfitting, essentially lowering the chances that weights tend to infinity, which is another attributing factor to overfitting.

SHOW WITH AND WITHOUT REGULARISATION

DISTRIBUTION CURVE. Overfitting still occurs as shown in Figure wWQEqEQWEP, and rebalancing the data by word frequency cannot solve this problem, since the phrase-frequency will still cause imbalance. Furthermore, if words are more common, they should appear more commonly in the generated text and the opposite should remain true for more infrequent words and patterns. The actual issue with recurrent neural networks is that when they are trained on tokenized imbalanced data using solely categorical cross-entropy, they will quickly overfit the data and will struggle to generalize, indicated by the validation loss and accuracy diverging from the training loss and accuracy.

4 Analysing generated natural language

4.1 Lyric Genre Classifiers

In order to test the results of our lyric generation models we have considered multiple possible methods. The main method being creating a test lyric genre classifier.

Our analysis involves accuracy with the classifiers alongside their F_1 score which is defined as

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}.$$

The F_1 score is traditionally used for binary classifiers and so in order to use it on the multiclass classifiers, macro F_1 scores are used. This extension is simply the mean average of the F_1 score for each class.

4.1.1 Data Preprocessing

In order to classify the genre of lyrics, a classifier will need to be trained on lyric data other than just rap lyrics. Lyric data for rock, country and pop has been gathered from Kaggle [12] and the Genius

API [24] in order to have 4 genres our models are able to categorise input lyrics to.

Each classifier uses samples of 'lines' estimated at 10 words each. These are fed into the classifer for training and allow us to predict each line of our generated rap. Using the time taken to complete the first epoch, each classifier has been trained on the estimated number of epochs needed to have a training time of 1.5 hours.

For our lyrics that have been generated we have classified 400 lines for each of the main branches of the generators.

4.1.2 Multinomial Bayes Theorem Classifier

As our first classifier, we created a lyric genre classifier utilising multinomial naïve Bayes theorem. Within the classification model, input data is translated into a matrix of token counts and then is classified using a naive Bayes model. This is done using CountVectorizer and MultinomialNB from scikit-learn.

This model has negligible training time and gives quick results. The method estimates the likelihood of each word within the vocabulary for each genre. Our results for this classifier are shown below:

	Bayes	CNN	RNN
Number of Epochs	val	val	val
Total Training Run Time	val	val	val
Final Training Epoch Accuracy	val	val	val
Test Set Accuracy	val	val	val
Test Set Macro F1-Score	val	val	val
Markov Generator Correct Fraction	val	val	val
RNN Generator Correct Fraction	val	val	val

Multinomial Bayes Theorem Classifier	
Test Set Accuracy	0.3844
Test Set Macro F1-Score	0.3827
Markov Generator Fraction	0.6075
RNN Generator Fraction	0.5450

As can be seen, the test accuracy and macro F1 score are both less than 50%, whilst this classifier is usefull as a start, more advanced methods should be explored.

4.1.3 Convolutional Neural Network Classifier

Our CNN classifier was created using a convolutional neural network using preprocessing and layer packages from Keras.

Due to the short run time of the epochs, 1080 epochs have been run in order to have a similar training time to the RNN classifier. The model did, however, plateau in accuracy at around the 277'th epoch. This model achieved good accuracies, rating a majority of each generated lyrics as rap. Our results for this classifier are shown below:

CNN Classifier	
Epochs	1080
Total Training Run Time	4353s
Final Training Epoch Accuracy	0.9385
Test Set Accuracy	0.7333
Test Set Macro F1-Score	0.6499
Markov Generator Fraction	0.8025
RNN Generator Fraction	0.8857

4.1.4 Recurrent Neural Network Classifier

Our RNN classifier was created using a recurrent neural network with long short-term memory using preprocessing and layer packages from Keras.

Each epoch for this classifier required longer times to complete and as such, far fewer have been run. Each generator performed well when classified. Our results for this classifier are shown below:

RNN Classifier	
Epochs	26
Total Training Run Time	4835s
Final Training Epoch Accuracy	0.9007
Test Set Accuracy	0.7117
Test Set Macro F1-Score	0.6221
Markov Generator Fraction	0.8725
RNN Generator Fraction	0.7800

The different validation accuracies for different training sessions are listed below.

4.2 Performance metrics

Different performance metrics have been implemented as a more rigorous form for testing the generated text. The simplest feature to observe is the word frequency, since the majority of rap uses a wide vocabulary, with the best artists having a high variance of words. Figure **PUT FIGURE IN** shows that... Next, rhyming towards the end of sentences is considered standard amongst most rap songs **SOURCE** and the generated lyrics should be no different. The above 'rhyme ranker' metric is capable of measuring the reverse word similarity and the end of lines. However, this proposed metric biases lengthier words and doesn't account for phonemes opposed to vowel-based syllables **Show two words that should rhyme but don't with the old metric**. Subsequently, these disadvantages can be downsized through an improved rhyme metric, defined as **MIKE'S METRIC**.

5 Discussion

The goal of a 'ghostwriter' is to produce lyrics that relate to the content of the writer, whilst remaining unique and not copying previous lyrics. This was a large problem we found, in that if we overfit the data just a little bit, lyrics used in the dataset would crop up in our generated lyrics. On the other hand, if we underfit it was clear there was no fluency or meaning to the lyrics at all. After much fine-tuning, we were able to find a model that produced original, rhythmic lyrics.

However, using the BERT Model [25], it would be possible to create a more coherent set of lyrics. BERT is known to be highly effective in 'Next Sequence Prediction' and so by producing a number of lyrics based off a seed text, using BERT we could have chosen the generated lyric which matched the seeded text best. BERT uses masked language models to predict certain words in a given text, using Transformer to take into account the different contexts a word is used in. The use of the Transformer removes the use of the 'left to right approach', in favour of a bidirectional approach, meaning the model can learn the context of a word based on all words around it instead of just the words to the left.

Perhaps another problem is the unpredictability of rap lyrics, and how we could match this unpredictability in our lyrics. The task of language generation is certainly very complex and there are so many intricacies that need to be looked at to create the best model, such as grammar, semantics, and rhyme and to do this would have required a lot of work to perfect.

More could have been done to improve the efficiency of our algorithm. From a sustainability point of view, the amount of time training the model must of accumulated to days, and this no doubt would have produced a lot of co2 emissions. In the future it is important that more is done to improve efficiency and perhaps looking for an increase in 0.5% accuracy is not worth the time and resources. Overall, there are definitely inefficiencies with our model that could be looked at.

When we set out on this project, we wanted to compare and create models that could create original lyrics given a large dataset of lyrics of all kinds of rap artists, whilst maintaining some degree of rhyme that is present in rap music [26]. It was clear that it would be hard to replicate the complex rhyme patterns of some of the current skilled rappers such as Kendrick Lamar or Lil Pump, but we succeeded in creating a model that worked specific to rap music. We can also compare both the results of all the models to see how each one performed in comparison **WRITE A BIT MORE ABOUT THE COMPARISON EITHER HERE OR WHEREVER IT FITS IN PAPER**

6 Conclusion

7 Future Work

A natural extension to this project would be the generation of music alongside the rap lyrics, allowing a full piece of rap music to be generated with AI. This would involve scraping MIDI files from rap artists and using this data to train a neural network to produce similar music. The music would then be matched to lyrics generated using the methods explored above. The neural network created to automatically classify the number of syllables a word has would be valuable here, as it would allow words to be easily matched to music, with a sentence matching the music generated by bar or by

phrase. Looking at the ethics of this, there would be questions asked about who owns the music. Is it the creator of the model, or the AI itself that would make money from the music produced?

In rap music and any music for that matter, there is usually some sense of storytelling of the rappers past. With the use of a model such as ours, there would be a question over the truth of the stories being told. Perhaps information and stories from the rapper from the past could be scraped from online and fed into a model like ours to produce lyrics which have some relation to the rapper themselves. Additionally, we could look at the theme and semantics of lyrics separately, so that we could create lyrics that mimic a given theme, such as love or money.

Other extensions such as creating sign language interpretations for generated raps as described in Subsection 2.5 may also be considered.

References

[1] M. Dyson. *Know What I Mean? Reflections on Hip-Hop*, chapter Introduction. Basic Civitas Books, 2007.

[2] C. Lynette Keyes. *Rap Music and Street Consciousness*, chapter Introduction. University of Illinois Press, 2004.

[3] E. Malmi; P. Takala; H. Toivonen; T. Raiko and A. Gionis. Dopelearning: A computational approach to rap lyrics generation [online]. *KDD'16*, June 2016.

[4] Michael Proffitt. *The Oxford English Dictionary*. Oxford University Press, 2020.

[5] P. Edwards. *How to Rap: The Art and Science of the Hip-Hop MC*. Chicago Review Press, 2009.

[6] P. Potash; A. Romanov; A. Rumshisky. Evaluating creative language generation: The case of rap lyric ghostwriting [online]. Available from: https://arxiv.org/pdf/1612.03205.pdf?fbclid=IwAR3K-vFwMnwVGnov2C4VyZHS-fwnYJMfGbLMKmp-0x4SKuPdD6TsuO3ce0k, December 2016.

[7] C. van der Lee; A. Gatt; E. van Miltenburg S. Wubben; E. Krahmer. Best practices for the human evaluation of automatically generated text [online]. Available from: https://www.inlg2019.com/assets/papers/98_Paper.pdf, 2019.

[8] G. Reddy M; Y. Yu; F. Harscoët; S. Canales; S. Tang. Automatic neural lyrics and melody composition [online]. Available from: https://arxiv.org/pdf/2011.06380.pdf, November 2020.

[9] Y. Feng; B. Xu. Lyric generation with style [online]. Available from: http://www.bicheng-xu.com/files/Lyric_Generation.pdf.

[10] Ben Saunders, Necati Cihan Camgoz, and Richard Bowden. Everybody sign now: Translating spoken language to photo realistic sign language video, 2020.

[11] Google Cloud. Text-to-speech. Available from: https://cloud.google.com/text-to-speech. [Accessed (27/03/2021)].

[12] Paul Mooney. Poetry and lyrics [online]. Available from: https://www.kaggle.com/paultimothymooney/poetry. [Accessed (11/12/2020)].

[13] ohhla. The original hip-hop archives [online]. Available from: https://www.ohhla.com/. [Accessed (11/12/2020)].

[14] S. Russell P. Norvig. *Artificial Intelligence: A Modern Approach*, chapter Natural Language Processing. Pearson, 2016.

[15] S. Asmussen. *Applied Probability and Queues*, chapter Markov Chains. Springer Science and Business Media, 2003.

[16] colo mize. Colomize studios [online]. Available from: https://colemizestudios.com/how-to-rap-structuring-lyrics/. [Accessed (18/04/2021)].

[17] Michael Holtzscher. Syllable counting algorithm [online]. Available from: https://medium.com/@mholtzscher/programmatically-counting-syllables-ca760435fab4. [Accessed (01/03/2021)].

[18] Kozea. Pyphen source code [online]. Available from: https://github.com/Kozea/Pyphen. [Accessed (14/02/2021)].

[19] Gary Darby. Delphiforfun [online]. Available from: http://www.delphiforfun.org/programs/Syllables.htm. [Accessed (10/03/2021)].

[20] Yoav Goldberg. A primer on neural network models for natural language processing, 2015.

[21] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[22] Michael R. Zhang, James Lucas, Geoffrey Hinton, and Jimmy Ba. Lookahead optimizer: k steps forward, 1 step back, 2019.

[23] Pierre Baldi and Peter J Sadowski. Understanding dropout. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013.

[24] Multiple. Genius api [online]. Available from: https://docs.genius.com/#web_pages-h2. [Accessed (01/03/2021)].

[25] Devlin ; M. Chang; K. Lee; K. Toutanova J. Bert: Pre-training of deep bidirectional transformers for language understanding. Available from: https://arxiv.org/pdf/1810.04805.pdf, 2018.

[26] H. Hirjee ; D. Brown. Using automated rhyme detection to characterize rhyming style in rap music. Available from: https://kb.osu.edu/bitstream/handle/1811/48548/EMR000091a-Hirjee_Brown.pdf?sequence=1&isAllowed=y, 2010. [Accessed (08/04/2021)].

2