Music Lyrics Generator

* COMP331 Data Challenge 2

Amy Lee
dept. of History
Occidental College
Los Angeles, United States
alee5@oxy.edu

Olivia Baldwin
dept. of Computer Science
Occidental College
Los Angeles, United States
obaldwingeil@oxy.edu

***Abstract*—Music is widely popular today and remains a huge part in people's lives. The industry was record to have a network of $19 million with more than a million songs released each year. Therefore, we are interested in seeing if we could train a machine that could potentially create a hit song of its own. In this paper, we will be discussing our N-gram model that was used to generate six lines of lyrics based on one of the ten genres chosen by the user. The different lyric outputs reflects how specific words and phrases are more attuned to specific genres as we used the top 100 songs from Metro Lyrics.**

I. Introduction

Technology is always improving and developing to mimic the brains function and even exceed it. For this project, we used the N-gram model to generate the next word, given a randomly selected word, until 6 lines of lyrics were outputted. We sought to test the ability of the N-gram model by examining to what extent a machine can generate lyrics cohesive and relevant enough to a specific genre. The goal of the model is to generate lyrics particular to the genre inputted by the user and grammatically coherent.

We used the top 100 songs per genre from MetroLyrics for our dataset. In total, we used roughly 1000 songs across 10 genres after manually cleaning the data set of non-English songs such as Korean (romanized and hangul), Arabic, Spanish, etc.

In the remainder of the paper, we first discuss related work and outline our approach and results. We conclude with discussion and future work.

II. Related Work

Other models have used N-gram for different prediction efforts. Su et al[1] used the N-gram model to predict a user's future request for a web page. They built their model based off a web-server log file L, which records each sessions' sequence of request to visit a web page, and on the occurrence frequency (Su et al., 2000). However, if the user's clicking sequence is not part of the prediction model, then the model will return "No Prediction." To assess their model, rather than using precision and recall, they used precision and applicability, which measures the portion of all requests that can be predicted, whether correctly or not, by the model. Similarly, we compare the different outputs with differing n values. However, unlike Su et al, we use the N-gram model not to predict the next set of lines given a certain line, but to generate a product.

Similar to the use of the N-gram model to predict future web requests, the N-gram model has been used to predict, given the beginning words of a sentence, the subsequent words (Bickel, Haider, and Scheffer, 2005). Bickel, Haider, and Scheffer adapted the N-gram to predict as much of the sentence the user currently intends to write but necessarily meaning the prediction had to end the sentence. For their model, they used precision and recall to assess the performance of their model. Precision was used to quantify the inverse risk of unnecessary distractions, meaning the time wasted by having the user look at the recommendation they do not select. Recall was used to quantify the rate of keystrokes saved by accurately prediction what the user was going to type. Their data was also separated into different categories: emails sent by the service center in reply to customer request, email correspondence between a management staff, daily weather reports, and cooking recipes. Likewise, we used the N-gram model to help generate words within a specific category. In contrast, their model was tested by how often the predicted phrase of words accurately predicted what the user intended to write while our model tested for its grammar and relevance.

III. Methodology

Our model is created by an implementation of the Bigram and Trigram language models. The model first reads in the data and organizes it into individual lists by genre. Depending on what genre is entered when running the program, that genre's data will be input into the model. Once the model is running, it will create unigram, bigram, and trigram lists for the specified data. The unigram list keeps a count of the number of time each individual word appears in the data while bigram does this every pair of words and trigram for each group of three words. An example section of each list is shown bellow:

After these list are created, we begin the process of generating our own lyrics. First, a starting word is chosen for each model by calling the functions getStartWord for the bigram model and getStartPair for the trigram model. These methods

. A special thanks to Dr. Celia Chen of the Computer Science of Occidental College, without whom this project would not have been possible.

1. .

```
Unigram:  {'And': 122, 'as': 28, 'it': 160, 'fell': 2, ',': 573,
'you': 368, 'rose': 1}

Bigram: [[('And', 'as'), 1], [('as', 'it'), 1], [('it', 'fell'), 1],
[('fell', ','), 1], [(',', 'you'), 14], [('you', 'rose'), 1]]

Trigram: [[('And', 'as', 'it'), 1], [('as', 'it', 'fell'), 1], [('it',
'fell', ','), 1], [('fell', ',', 'you'), 1], [(',', 'you', 'rose'),
1]]
```

generate a list of the words or pairs that immediately follow a 'newline' tag and that occur more than ten times in the data, then it randomly selects a word or pair from the list and this word or pair is sent into the getBigram or getNgram methods. The getBigram function contains a loop that runs until the model has printed six lines of lyrics. Each iteration of the loop prints the current word and then updates the current word to the output of the getBigramNextWord method. This method begins by creating a list of all the pairs that contain the input word in the first position, for example: [(I, stay), 4], [(I, am), 1], [(I, believe), 2]. Then, the method loops through each item in the pair list and assigns each pair a probability based on this equation:

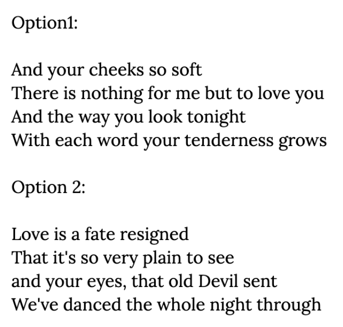
$$P(w2|w1) = \frac{C(w1, w2)}{C(w1)}$$

where w1 is the current word and w2 is the second word in the pair. The method retrieves the count of w1 from the unigram list. Now, the list of pairs and their probabilities is sorted from highest probability to lowest probability and a random selection is made from the top 10 percent of the list with the highest probabilities. The method then returns the second word in the chosen pair. The getNgram method words similarly to the getBigram method but instead of taking in a word, it takes in a pair of words. The method prints the first word of the pair, then assigns a new current word to the word returned by the getNgramNextWord method and a new pair is chosen from the bigram list which contains the word that was just printed in its first position and the new current word in its second position. The getNgramNextWord method is identical to the getBigramNextWord method aside from a few details. This method again takes in a pair rather than a word, then generates a list of trigrams which contain the pair in their first and second positions, for example: [(I, stay, at), 4], [(I, stay, here), 1], [(I, stay, gone), 2]. Then, the probability of each trigram is calculated by this equation:

$$P(w3|w1, w2) = \frac{C(w1, w2, w3)}{C(w1, w2)}$$

IV. Result and Analysis

To evaluate the accuracy of our model, we compiled a survey that asked participants to compare two lyric samples from each genre, one generated by our model and one taken from one of the text files in that genre, and judge which sample they believe was generated by a machine. After each comparison, the participant was asked to identify the genre they believe those lyric sets belong to out of the list of ten genres. Figure 1 is an example of two lyric sets participants were asked to distinguish.


Option1:

And your cheeks so soft
There is nothing for me but to love you
And the way you look tonight
With each word your tenderness grows

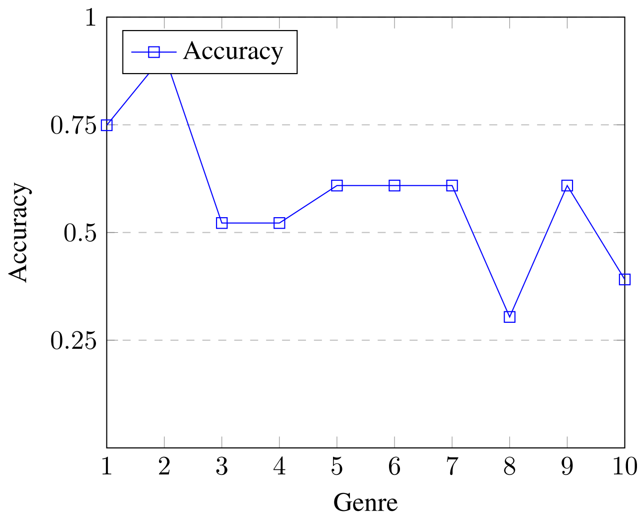
Option 2:

Love is a fate resigned
That it's so very plain to see
and your eyes, that old Devil sent
We've danced the whole night through

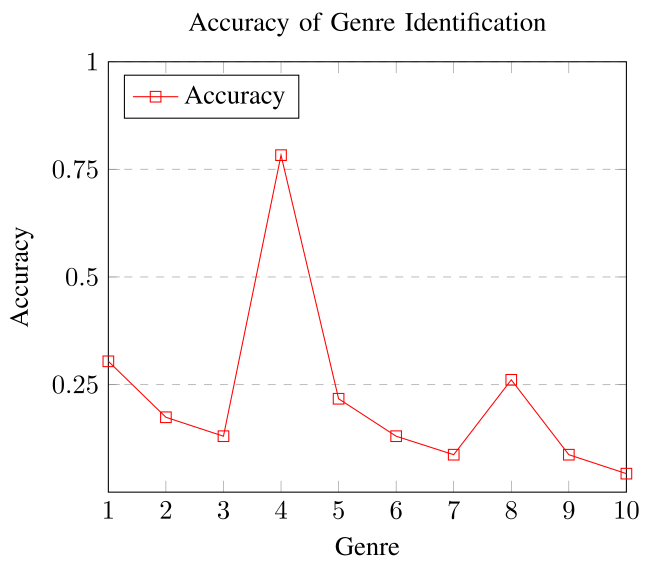
Fig. 1. Sample Lyric Comparison Question

Figure 2 depicts a table of the different accuracy percentages for each genre sample, or the percentage of participants who were able to accurately distinguish the model lyric set from the human lyric set. Each number on the x-axis represents a different genre: 1 is Country, 2 is Electronic, 3 is Folk, 4 is Hiphop, 5 is Jazz, 6 is Metal, 7 is Pop, 8 is RB, 9 is Rock, and 10 is Indie.

Figure 2: "Model or Human?" Participant Accuracy



As seen here, most accuracy levels fall between 50 and 60 percent. This indicates a high level of performance for the model as participants were unable to consistently distinguish between the model lyrics and the real lyrics. Figure 3 depicts the accuracy at which each genre was identified by participants. The numbers representing the same genres as in Figure 2.



Here we can see that only one specific genre had a high accuracy of identification, hiphop. We believe this to be due to the nature of the language involved in the genre and its uniqueness to hiphop. For example words like ”MC” typically only appear in the hiphop genre.

V. Threats to Validity

Threats to the validity of our model mostly revolved around our data. Firstly, we had to manually clean the data. We excluded non-English songs, files that were empty after scraping the data from MetroLyrics, and duplicates of songs. While examining the data, we questioned several songs placed under certain genres. This could have affected our model to skew the output; for instance, if a song that could be identified categorized as Country is placed in the Rock genre, then certain words, more associated with Country songs, would have a higher probability of being outputted.

Another problem with the data is the lack of uniformity in terms of format and language. Some of the songs included subtitles such as ”Verse,” ”Chorus,” or ”Bridge,” which are not part of the song lyrics. Other songs included sound effects, typically in parentheses, which would often be repeated multiple times, recorded with high frequency in our N-gram model, and subsequently then be generated. Furthermore, songs from the particularly in the Country genre would have some words in their colloquial form such as ”gettin’” or ”’cause.” Due to the different spelling, they are counted as a different word than then one they were related to such as ”getting” or ”because.” So, the probability of frequency does not accurately reflect the frequency of the words used in the lyrics. Lastly, some of the lyrics were not probably separated and were mashed together instead. For example, ”and the horses” were written as ”andthehorses,” which would be outputted in our model.

We did not manually change the categorization of songs nor the words used in the lyrics, however, to prevent imposing personal bias on the data.

In evaluating our results, a threat to our validity is how we selected which lyrics to use in our survey. We purposefully chose lyrics to make the distinction between non-generated and generated music lyrics difficult to discern by not choosing machine generated lyrics that had typos or long spans of repeated words. Furthermore, we chose to compare them to more ambiguous non-generated music lyrics, which could have affected our survey.

VI. Conclusion

In this paper, we proposed the N-gram model to generate music lyrics. From a survey, we can see that the model had produced lyrics relatively indistinguishable to real lyrics of the same genre. However, these lyrics were not distinct enough for surveyors to recognize the genre of the generated lyrics. Further work will include expanding the model to train with non-English songs as well as focus on the cleaning of the data for greater usability. We can also see future work towards generating entire songs, not just six lines of lyrics.

References

[1] Steffen Bickel, Peter Haider, and Tobias Scheffer. 2005. Predicting sentences using N-gram language models. In Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing (HLT ’05). Association for Computational Linguistics, USA, 193–200. DOI:https://doi.org/10.3115/1220575.1220600

[2] Su, Zhong Yang, Qiang Lu, Yiping Zhang, Hongjiang. (2000). WhatNext: a prediction system for Web requests using N-gramsequence models. 214-221 vol.1. 10.1109/WISE.2000.882395.