Song Lyrics Generator - Kanye

Oscar O'Rahilly

Stanford University, California, United States of America

March 9, 2020

In this project, I analyze all of Kanye West's songs, looking at his semantic choices, syllabic choices, and rhyming structures to produce 8 line rap verses in a similar style to Kanye West. This project consisted of three main parts. The first was to analyze his semantic choices in order to create sentences that sounded similar to lines written by Kanye West. The second step was to determine Kanye's average number of words per line so I could extrapolate a rough metric structure. The final step was to work out Kanye's typical rhyming structure.

1 Semantic Choices

To create lines for my verse I implemented a technique called the N-gram Language model. This is model that approximates the probability of a word appearing next, given all the previous words by using only the conditional probability of the preceding $N-1$ words. For example, suppose I had a source file containing all the words from the Harry potter series, and I wanted to calculate the probability that in the Harry Potter books the word "Potter" would appear next in the sentence "My name is Harry".

$$P(\text{Potter} \mid \text{My name is Harry}) \tag{1}$$

Using an 2-Gram (uni-gran) model we could approximate the above probability by calculating

$$P(\text{Potter} \mid \text{Harry})$$

Similarly, using a 3-gram (bi-gram) model we could approximate probability in equation 1 by calculating

$$P(\text{Potter} \mid \text{is Harry})$$

In order to estimate these N-gram probabilities, I used maximum likelihood estimation (MLE). I obtained the MLE estimate for the parameters of an N-gram model by getting the counts from the source, which in my case was a txt file containing all of Kanye's lyrics, and normalizing the counts so that they lie between 0 and 1. For example, to calculate the 3-gram (tri-gram) probability of a word x given the previous two words y, z, we compute the number of occurrences of the contiguous words x, y, z, $Occurrences(xyz)$, over the number of occurrences of the two words x, y, $Occurrences(xy)$. To use our example above, this would look like:

$$P(\text{Potter} \mid \text{is Harry}) = \frac{Occurrence(\text{is Harry Potter})}{Occurrence(\text{is Harry})} \tag{2}$$

The result of this probability would return a probability in the range 0 to 1.

One question I had to answer when looking into the implementation of the N-gram model was to decide what value of N I was going to use. The positives to a larger value of N is that it will sound more like the source text because you are conditioning your probability on more things by considering more of the previous words. For example, suppose you had a source text and the sentence "I shower every" and wanted to calculate the probability of the next word being "day" in your chosen source text. Intuitively we see that a tri-gram model would be more accurate than a bi-gram model because given the words "shower every", we know that the word "day" is far more likely to follow than the word "decade" (unless the source was talking about Shrek!), whereas in a bi-gram model we would simply see the previous word, "every", and so have less of an idea what could come next. The same is true for predicting text with an algorithm.

Conditioning the probability on more words will make the prediction for the next word sound more like the original source. The trade-off, however, is that a higher value of N requires more words in the source text for it to be accurate, otherwise you end up generating sentences which are exactly the same as the source text and not "original". Thus in the end I found the 3-grams (tri-grams) seemed to yield text that sounded most like Kanye West, whilst also not being sentences directly drawn from his songs.

I implemented the tri-gram model by creating a dictionary where each key entry was every contiguous pair of words from all of Kanye's songs, (example of key entries shown below) and the values were another dictionary where each key entry was a word that followed the original two words and their values were the number of times that they appeared after.

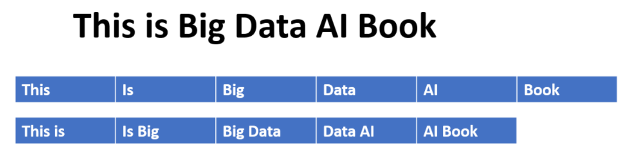


Figure 1: *bi-gram key entries & tri-gram key entries respectively using the text "This is Big Data AI Book"*

An example of an actual dictionary entry for the Kanye tri-grams is shown below. (total_count is simply a variable that tells me how many times 'i am' appeared in the source text).

{'i am' : {'total_count': 20, 'a': 5, 'one': 2, 'and': 1, 'bored': 1, 'lord': 1, 'limelight': 1, 'the': 6, 'never': 1, 'so': 1, 'all': 1}}

Using this complete dictionary, I can create a random sentence by picking a random key entry, such as "I am", and then choosing a random word in the value dictionary (weighted on its appearance e.g. "one" has a $\frac{2}{20}$ chance of being picked) and then repeating the same process for the resulting two words sequence. For example if I chose "one", i would look at the dictionary for the entry "am one" and choose the next word the same way I described above.

To determine the length of each sentence I calculated the mean number of words Kanye uses per line (8.3) and the standard deviation of the distribution (2.52). For my lines I chose random lengths (weighted on their frequencies) that were integers and within one standard deviation of the mean.

2 Syllable analysis

Despite being able to create sentences of any length using a tri-gram model, when I was reading out my 8 line verses, they did not flow like a Kanye West song

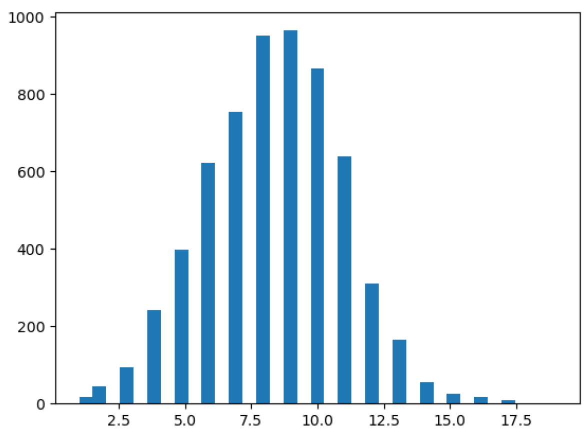


Figure 2: *number of words/line on x-axis and frequency on y-axis*

does. Thus to try and simulate this rhythmic quality found in Kanye's rap songs I read my source file line by line to determine the average number of syllables in each line and determine the mean (10.3) and standard deviation (2.98) number of syllables. For each line, similar to word count, I only used line that were within one standard deviation of the mean number of syllables.

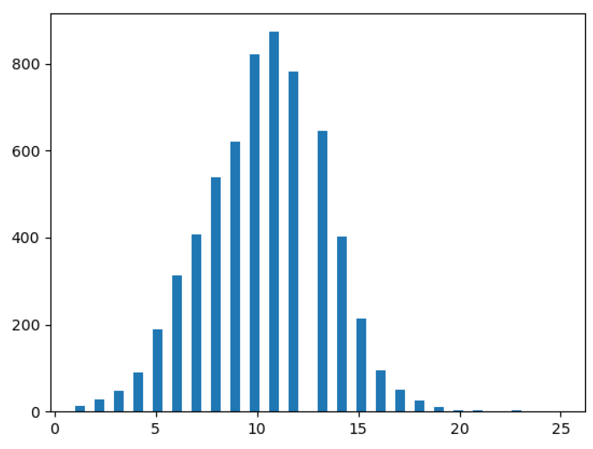
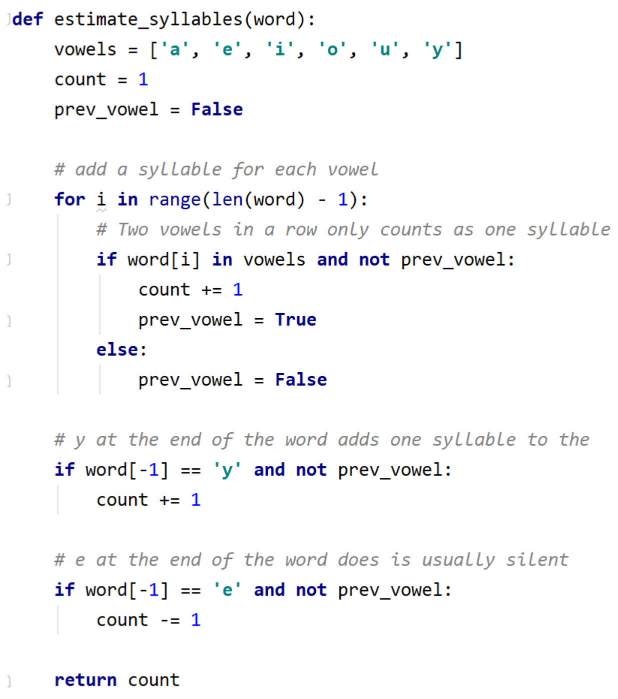


Figure 3: *number of syllables/line on x-axis and frequency on y-axis*

For the most of the words Kanye uses, I was able to use a Python library called Pronouncing which used the Carnegie Mellon University (CMU) pronouncing dictionary to determine the number of syllables in a word. Unfortunately, there are some words Kanye uses that are not in the CMU pronouncing dictionary, such as shortened words like *orderin'*. Thus, if a word could not be determined using the pronouncing library, I use an algorithm I wrote that estimates the number of syllables in a word using the number vowels of that appeared in that word.

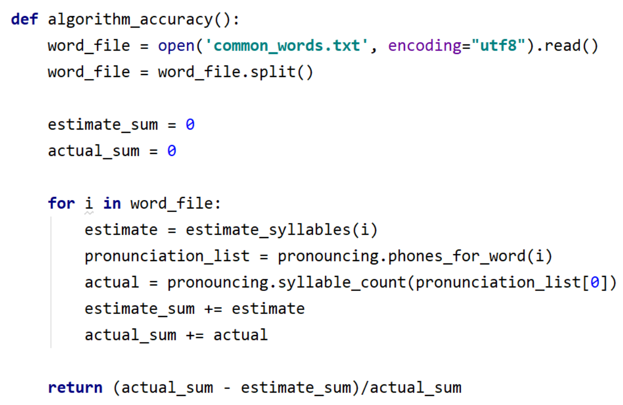
To make sure my estimation algorithm was accurate I wanted to find a way to compare it to the



```python
def estimate_syllables(word):
    vowels = ['a', 'e', 'i', 'o', 'u', 'y']
    count = 1
    prev_vowel = False

    # add a syllable for each vowel
    for i in range(len(word) - 1):
        # Two vowels in a row only counts as one syllable
        if word[i] in vowels and not prev_vowel:
            count += 1
            prev_vowel = True
        else:
            prev_vowel = False

    # y at the end of the word adds one syllable to the
    if word[-1] == 'y' and not prev_vowel:
        count += 1

    # e at the end of the word does is usually silent
    if word[-1] == 'e' and not prev_vowel:
        count -= 1

    return count
```

Figure 4: *syllable estimation algorithm*

pronouncing library's syllable counting method. To do this, I found a text file containing the 1000 most common words in the English language and ran my estimation algorithm and the built in function on every word, keeping variables that counted the total number of syllables for each algorithm. I then took the difference between the two and divided it by the built-in function's total (as this is the correct number) to find the relative error of my estimation function, which came out to 0.0247. The fact that this value was positive means that my algorithm tended to overestimate the number of syllables in a word.



```python
def algorithm_accuracy():
    word_file = open('common_words.txt', encoding="utf8").read()
    word_file = word_file.split()

    estimate_sum = 0
    actual_sum = 0

    for i in word_file:
        estimate = estimate_syllables(i)
        pronunciation_list = pronouncing.phones_for_word(i)
        actual = pronouncing.syllable_count(pronunciation_list[0])
        estimate_sum += estimate
        actual_sum += actual

    return (actual_sum - estimate_sum)/actual_sum
```

3 Rhyme Structure

Now that I had lines for my verses that used the similar semantic choices and rhythmically sounded like Kanye West lyrics, the final stage was to get my lines to rhyme in the same structure that Kanye rhymes. The first step in this was to parse my source file verse by verse, and for each verse return a list that represented the rhyming scheme for that verse. For example the following verse from Kanye West's *Welcome to Heartbreak*,

Dad cracked a joke, all the kids laughed
But I couldn't hear him all the way in first class
Chased the good life my whole life long
Look back on my life and my life gone
Where did I go wrong?

would return the array, ['a', 'a', 'b', 'b', 'b'], where each index represents a line and the value at that index the rhyme group it is in. This array means that the first two lines all rhyme with each other and the last three lines all rhyme with each other.

Determining if two words rhyme, however, is less straight forward than it sounds. Originally, I decided to tackle this by using a built in method in the Pronouncing Python library that returns a Boolean value telling you if two words rhyme or not. The problem with this is that Kanye very often rhymes words that don't actually rhyme by contorting the way he says the words. For example in this short extract taken from Kanye West's *Robocop*

Up late night, like she on patrol
Checkin' everything like I'm on parole
I told her it's some things she don't need to know

the words *patrol* and *know* don't officially rhyme, according to the New Oxford Rhyming Dictionary; however, when you listen to the song he manages to rhyme the two words. A common theme I noticed is that Kanye is mostly able to rhyme words if they have the same final vowel. Thus, I wrote a function that compares the pronunciation guide, according to the CMU pronunciation dictionary, and compares the final vowel sound in two words and returns True if they are the same and False otherwise. For example the words *patrol* and *know* return the following pronunciation guide, [['P', 'ER0', 'OW1', 'L']] and [['N', 'OW1']]. Here we see that the final vowel sounds in both are the same, *OW1*, and so my algorithm says that they rhyme.

Now that I was able to determine if two words rhyme, I went about finding Kanye's average rhyming scheme for the first eight lines of every verse. To do so, I initialized an empty array with length 8 and for each index, i, filled it with the modal letter of all the rhyming scheme arrays for index i. The result of this returned an average rhyming structure that Kanye uses for the first 8 lines of every verse, ['a', 'a', 'b', 'b', 'c', 'd', 'd', 'd'].
Using this rhyme scheme, I was then able to create an 8 line verse that rhymed in a style most similar to Kanye's raps.

4 Results

Putting all of these three techniques together, I was able to create 8 line verses which sounded semantically and rhythmically similar and used similar rhyming schemes to Kanye West. One of my favorite verses my algorithm created is shown below. Note that I have added punctuation to this verse manually.

Way y'all don't feel me man I'm gon' tape it and came
young world I'm the new Jesus, chains in Jesus' name
models we rap we don't care what people say you
gold Jesus piece man I've been down this road too.
Booed at Apollo ghosts use my best friends worry 'bout me
do no press but i don't give up now they
just see your girlfriends you need a rodman, jay
pg or atl back in the hood its a blessin', man I say.