"I'm like yeah": An Exploration into Popular Rap Lyrics

Stephen Owen, stephen.owen1@richmond.edu

1 Introduction

Rap music has become one of the defining genres of the current generation, often transcending it's own genre space and becoming a mainstay in the modern "pop" sound. The rap genre is known for having stereotypes about its lyrical content, so I thought it would be interesting to explore some of the most common words and phrases in the 100 most popular rap songs of all time according to Genius and analyze them for any common trends. Upon researching how to use the tools required for use in this project, I found another project [1] where someone had already explored the idea of performing sentiment analysis on rap lyrics. I found this to be a very interesting idea so I also explored sentiment trends for the lyrics in my data set. I would not have been able to feasibly do this project without the "LyricsGenius"[2] library, which I will go further in depth on below.

2 Approach

2.1 Data set

For this project, I decided to use the website Genius (formerly Rap Genius) as my data source. Genius is unique in that crowd sources it's lyrics and lyric interpretations, so it is often one of the most accurate sources for modern and popular song lyrics. Among it's other benefits, Genius is also one of the more modern lyric repositories on the internet. There are loads of websites with simple page with nothing the title of the song and the lyrics on it, pulling lyrics from pages like these without building a full on web-scraping program would be a nightmare. Thankfully, Genius has a public API, so the task of pulling lyrics for my data set was much easier than it otherwise would have been. Genius has a feature where you can browse songs by popularity, so I was able to easily find the top 100 most popular rap songs of all time and store the title and artist of those songs.

2.1.1 LyricsGenius

While Genius does have a public API, as described above, it (like many other public APIs) is not very intuitive to use. Thankfully there is a Python library titled LyricsGenius which acts as a more intuitive front-end for the Genius API. It is specifically tailored to making it easy to search for songs, and save the metadata associated with it (in which the lyrics are contained). For example, If I was trying to find the lyrics to the hit classic "3 Hours of Glottal Stops" by rap icon Tom Bonfiglio [3], it would be as easy as:

```
search("3 Hours of Glottal Stops",
"Tom Bonfiglio")
```

[1] https://github.com/Hugo-Nattagh/2017-Hip-Hop
[2] https://github.com/johnwmillr/LyricsGenius
[3] Unfortunately, this song doesn't actually exist.

2.2 Python

For the purposes of this project, I chose to use the programming language Python as my tool of choice. Of the plethora of programming language options available, Python is not only my language of choice, but the language of choice for many others working on language processing tasks. Python has a multitude of pre-made libraries that drastically reduce the development time of some of the tasks I will describe later. On top of all of that, because of some of the design decisions of the Python language (the specifics of which are outside the scope of this paper), the code required for my analysis can be written much faster in Python, in comparison to say, something like Java or C++.

2.3 Cleaning

One of the shortcomings of the Genius API mentioned above is that it is limited to searching for songs by title/artist name only. Because of this I had to find an admittedly hack-y solution to getting automatically pulling the data (I did *not* want to individually write out 100 separate requests). I was able to pull all of the 100 artist/song titles directly from the website, dump them in a text file, and read them in through python. From there, I was able to use a handful of operations to clean out the other junk that was copied along with the information, and then cleanly format the artists and song names so their lyrics could all be pulled in one fell swoop.

2.4 NLTK

NLTK, which stands for Natural Language Tool Kit, is a Python library built for handling and processing text-based data. It is almost universally considered the state-of-the-art, and is the most common choice for any task involving textual analysis and language based development.

2.4.1 Tokens

Before a text based data set can be used in any meaningful capacity, it has to be what is called "tokenized." Tokenization is the process of breaking down a piece of text into individual parts (called tokens) based on some heuristic that accomplishes the desired goal. NLTK comes with a number of pre-built tokenizers available, and I mainly used "word tokenizer" and "sentence tokenizers" (which tokenize a corpus exactly as you would expect given their respective names). There are other, more powerful tokenizers, such as the regex tokenizer, but because of the fact that lyrics are typically uniformly formatted (no weird characters to parse, and rarely any punctuation outside of commas, periods, question marks, and explanation points), the simpler tokenizers are more than sufficient for the task at hand.

2.4.2 N-Grams

If tokens are the atoms (in the sense that they're fundamental building blocks) of natural language processing tasks, then n-grams are the elements. N-grams are chunks of words from the data set of some preset length n, where n is some positive integer. For example:

- Unigram: "This"
- Bigram: "This is"
- Trigram: "This is a"
- 4gram: "This is a 4gram"[4]

2.4.3 Stop Words

Stop words are a set of words that often necessary for a sentence to make sense to humans, but often don't tell us anything of any inherent value when analyzing a corpus. These are often words like "the", "a", "is", etc. In natural language processing, these kinds of words are often removed from the corpus prior to the analysis. For example, if we were, say, trying to find the most frequently used words and phrases from a corpus (something *quite* pertinent to this paper), it wouldn't be a very interesting result to see "the" as the most common word in the entire corpus. Removing them from the corpus prior to analysis yields much more interesting and significant results.

2.5 VADER

When conducting sentiment analysis, there are two commonly used approaches: model based and rule based. Model based approaches involve

[4] n-grams are commonly referenced using numbers instead of prefixes whenever n is greater than 3.

building a training set of text pieces (sentences, tweets, lyrics, etc.) which are paired with a predetermined sentiment score (ranging from -1 to 1, where -1 represents a strongly negative sentiment, 1 represents a strongly positive sentiment, and 0 represents a neutral sentiment). This set of text is then fed into a model, in which the model figures out what words correspond to positive/negative changes in sentiment. Rule based approaches, on the other hand, use a predetermined list of words with associated positive/negative weights and compare the input text to the set of predetermined "rules". Each approach has its own set of pros and cons. Model based approaches often offer greater accuracy, because they are able to extract features that are unique to the data set. In other words, they can consider how words that are unique to the data set affect the sentiment of a phrase. However, as mentioned above, they require a pre-built data set of phrases with associated sentiments, prior to the construction of the model. Constructing a useful model would likely require thousands of examples with associated sentiments, so creating this data set is often easier said than done. On the other hand, rule based rule based approaches offer greater flexibility, allowing the analysis to be performed on new data sets without being trained on some preconstructed one. However, this generality comes at the cost of decreased accuracy. Because of the lack of a robust pre-built data set for a task like this, I decided to go with a rule-based sentiment analysis tool called VADER (which stands for Valence Aware Dictionary for sEntiment Reasoning). When analyzing a phrase using VADER, it returns 4 scores: Positive, Neutral, Negative, and Compound. The first three of these return values ranging from 0 to 1, and indicate the level to which the inputted phrase matches VADER's pre-built dictionary of positive, neutral, and negative words, respectively. The compound score is a weighted sum of these values, and gets mapped to a value between -1 and 1, as I described above.

3 Experiments & Analysis

3.1 Common words and phrases

3.1.1 Length 1

As described above, with removed stop words, the 10 most common phrases of length 1 throughout the entirety of the corpus are as follows:[5]

Word	# of Occurances	% of Corpus
"i'm"	875	2.31%
"like"	669	1.77%
"yeah"	561	1.48%
"got"	481	1.27%
"know"	401	1.06%
"get"	385	1.02%
"bitch"	302	0.80%
"fuck"	268	0.71%
"n****"	264	0.70%
"one"	257	0.68%

The word choices here aren't very surprising for the most part. "I'm" is a very versatile word, so its presence is especially not surprising. Looking at some of the songs from the corpus, we see the word "i'm" being used in various contexts like:

- "**I'm** beginning to feel like a Rap God, Rap God"
- "Girl, **I'm** Kendrick Lamar (Mmm)"
- "Y'all know **I'm** a grown boy, your clique full of broke boys"[6]

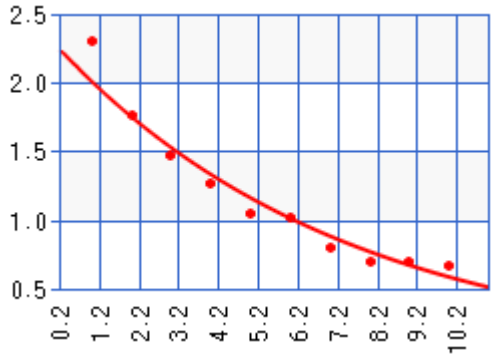
We have a similar story when we look at the second most common term "like". Like can be used in a lot of different contexts, and rap is historically fond of metaphors and similes, so the high placement of the word "like" comes as no surprise. The word "yeah", however, is different from the other two. Yeah is not necessarily "versatile" in the same way "like" and "I'm" are, and is usually used as an affirmative response to a prior statement. Looking at the data set, we see "yeah" being used as a unique "ad-lib" type word (often being used at the end / beginning of a phrase in order to transition to the next thought/idea), rather than one with any inherent meaning. We can see common usage of the word "yeah" in examples such as:

[5]The title of this paper comes from the top 3 most common words on this list.

[6]Examples taken from "Rap God" By Eminem, "Fuckin' Problems" by A$AP Rockey ft. Kendrick Lamar, 2 Chainz, & Drake, and "Love Sosa" by Chief Keef.

- "Might go down a G.O.D., **yeah**, wait"
- "Push me to the edge **(Yeah)**"
- "**Yeah**, this shit way too formal, y'all know I don't follow suit"[7]

As we expect, the frequency percentage goes down as the words get less and less common, so I felt inclined to graph the decay. This yielded a graph with a very clear negative exponential trend. So, I ran exponential regression on the data points and was provided with the following figure.



The results of the regression (which was of the form $y = A \times B^x$) yielded the coefficients $A = 2.302935625$ and $B = 0.8720104904$.[8] Analyzing this regression line gives us an R^2 value of -0.977592165. This represents an extremely strong negative correlation and allows us to make predictions about less common words. For example, using this equation, we can predict that the 20^{th} most common word takes up approximately 0.14% of the entire corpus.

3.1.2 Length 2

Immediately upon analyzing the common phrases longer than length 1, we find one of modern rap's most common characteristics: its repetitions.

Word	# of Occurances
"yeah yeah"	102
"hey hey"	71
"yeah I'm "	53
"Gucci gang"	52
"I'm one"	48
"gang Gucci"	44
"like I'm"	37
"feel like"	36
"clique clique"	36
"back back"	33

We can see from the above chart that 4 out of the 10 phrases are just the same word repeated twice. We also have the weird case where in the song "Gucci Gang" by Lil Pump, he says the phrase "Gucci Gang" so many times in succession (52, in the whole song), the bigram counter caught the case where both the phrase itself appears, and the shifted phrase appears. In other words, the bigram counter caught

- "[Gucci gang], [Gucci gang], [Gucci gang], [Gucci gang]"
- "Gucci [gang, Gucci] [gang, Gucci] [gang, Gucci] gang"

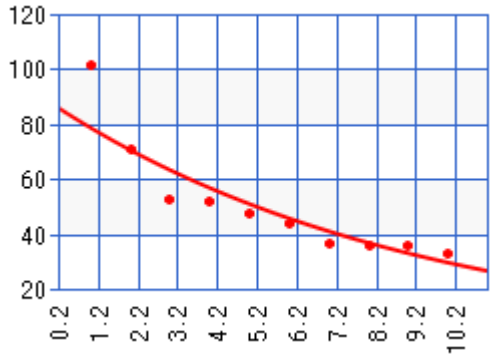
and both phrases occurred frequently enough (thanks in no small part to Lil Pump's *ingenious* writing) that they were both in the top 10.

Running the same sort of regression as we did with the phrases of length 1 yielded the following graph.

[7]Examples taken from "God's Plan" by Drake, "XO Tour Life3" by Lil Uzi Vert, and "SICKO MODE" by Travis Scott and Drake

[8]I decided to graph the percentages of the entire data set rather than the total counts because it kept the numbers smaller, while still capturing the overall trend.

[9]For the phrases of length 2 and 3, I plotted them using the counts, instead of the proportions. I did this to be consistent with the tables, which don't show the proportions due to the tables being to large to fit on the page if proportions were included.



The results of this regression gave us the coefficients $A = 87.5587341$ and $B = 0.896908871$.[9] The regression line here has an R^2 value of -0.93428769 which again, like the length 1 phrases, suggests a very strong negative correlation.

3.1.3 Length 3

We don't see anything wildly different when we analyze phrases of length 3.

Word	# of Occurances
"yeah yeah yeah"	52
"Gucci gang Gucci"	44
"gang Gucci gang"	43
"hey hey hey"	40
"clique clique clique"	27
"bitch kill vibe"	25
"fuckin' problem problem"	24
"panda panda panda"	22
"I'm one yeah"	21
"throwing away shot"	21

However, we do see a couple interesting points that are worth bring up. Namely, here we have "bitch kill vibe" and "throwing away shot". We can extrapolate that these are both cut down versions of the phrases "Bitch don't kill my vibe" from the aptly titled "Bitch don't kill my vibe" by Kendrick Lamar, and "I am not throwing away my shot" from "My Shot" from the Hamilton Soundtrack. This is interesting because we can assume they were cut down because of the removal of stop words (it makes sense that words like "I", "My", "am" would be considered stop words), yet because of their frequency in the data set, they were still picked up by the trigram counter.

Looking at the regression plot, we see a similar trend to the regression plots from the shorter phrases.

This regression line gave us the coefficients $A = 55.466706$ and $B = 0.8950288541$. The R^2 value associated with this line was -0.95774405 which, again, suggests a strong, negative correlation. Considering the three regression lines we've constructed, It shows us something interesting. At first glance, one might think "Ok so, less frequently used words are...well, less common". Although this is true, I would agree that this is a tremendously boring result. However, this becomes interesting when we consider the fact that not only was there a negative correlation, but the data strongly followed an exponential decay curve for each set of phrase lengths. In other words, not only are less frequently used words less common, they are *exponentially* less common. I believe this is indicative of some larger mathematical property that has to do with word frequencies and would be worth exploring further.

3.2 Sentiment

Earlier, I mentioned one of the downsides of rule based sentiment analysis was that, although easier to implement, it often offers decreased accuracy when compared to model based approaches. That statement is definitely seen in my analysis. I fed each line of each song through VADER, and collected the the 5[10] lines with the compound scores closest to 1, and the 5 lines with the compound scores closest to -1 (this is analogous to picking the 5 "most positive" and 5 "most negative" phrases). The top 5 positive lines in the corpus were:

[10] I could have collected any number of the top lines, but I limited my discussion, insofar as this paper is concerned, to 5.

1. "Offset, woo, woo, woo, woo, woo"
2. "You're such a fuckin' ho, I love it (Love it, love it)"
3. "Just love me, just love me, just love"
4. Just love me, just love me, just love me""
5. "My love, my love, my love"

And, the top 5 negative lines in the corpus were:

1. "Send me some mo' shit, you triflin' ho bitch (Bitch, bitch, bitch)"
2. "Bad thang, bad, bad, bad, bad thang, thang, thang, thang, thang"
3. "Ignore the hate, ignore the fake, ignore the funny shit (Ignore the funny shit)"
4. "I'm Attila, kill or be killed, I'm a killer bee, the vanilla gorilla"
5. "My bitch is bad and bougie (Bad, she bad)"

Above all else, I think this sheds some light into the weakness of rule based approaches to sentiment analysis when used in an application like this. Rap, as a genre, relies on it's slang as a means of communicating ideas, which makes creating robust, general-purpose rules for analysis is difficult. We can deduce, from looking at the positive statements, that the word "love" clearly has a positive affect on the perceived sentiment, as it shows up frequently in almost all of the statements.[11] However, where my previous point about slang comes into play is when we look at the negative statements. Based off what we see here, I think it would be fair to assume that somewhere in VADER's construction, the word "bad" was set to carry a negative connotation. However, in rap this isn't necessarily the case. In the 5^{th} negative line "My bitch is bad and bougie (Bad, she bad)", this is a line where the rapper Offset is praising someone. In recent slang, "bad" has become synonymous with "sexy", a positive adjective. As a result, statements that are arguably positive are getting classified as negative. This highlights some of the unique characteristics of rap's overall vernacular. This leads me to believe that something like VADER would perform much better on something like a data set of product reviews, as they tend to be written by an older demographic and thus use less slang.

4 Conclusions

I think there are interesting trends here that would be worth exploring in more detail. As mentioned at the end of my analysis of the 3 word phrases, I think there is some inherent property relating to the proportions of common words that I would like to research more. I also believe this project had a fair bit of shortcomings. Due to the nature of repetitions in rap lyrics, they accounted for most of the common words and phrases, which I see as a bit of a roadblock to getting more interesting results. In order to work past that, I would need to figure out some way to account for repeated words, without outright removing repeated words from the data set(as this would be inherently misrepresenting the lyrics). I think another potential area of improvement would be the usage of stemming. Stemming is a technique in which variants of a given root word are considered identical to the root word according to the program. For example, with stemming applied, the words "like", "liked", and "likes" would all be considered the same word.

4.1 Future Work

As mentioned above, these experiments would have been more robust had I implemented stemming, which I believe would have produced better results. There are many ways that this work could be extended. What comes to mind immediately to me would be to build a more robust data set. While using the top 100 songs on Genius provided a sufficient amount of text for some initial exploratory analysis, coming up with anything truly meaningful and robust would undoubtedly require a much larger corpus of song lyrics. Aside from that, given more time I think it would be interesting to potentially segment the lyrics by artist locale. With this, I could run similar experiments and see if, say, there is a difference in common words between west-coast and east-coast rappers, or rappers from different countries, and the like. Another interesting idea would be to to compare the common words on an artist-by-artist basis and

[11] Although, I will say, the first statement has me at a loss...

use that data to generate a clean graphic like the one seen in the repository marked in footnote 1. A less "useful", but fun application of a data set like this would be to feed the text into some kind of generative model (be it a language model, neural network, or what have you) and construct a program to generate rap lyrics.

5 Appendix

The code is available on github at:

`https://github.com/xbarjoe/GeniusAnalysis`

Thanks Dr. Bonfiglio for a great semester!