CS221 Project Milestone: “The Sixteen Machine: Generating Intelligent Rap Lyrics”

PROBLEM STATEMENT:

We are basically creating an AI agent that produces rap lyrics based on other artists lyrics. We are using a modified n-grams model for our language model, and we have a modified cost function that takes in account making sure lines rhyme and they match rhythmically. More details about this are explained below.

MODEL:

Our algorithm will choose the next word in a rap lyric based on the cost of the produced rap lyric according to a language model. Our language model is a modified n-gram model. We have modified the basic n-gram model to incorporate costs based on lines rhyming and costs based on rhythm (same syllable count in lines). Using this model, we can turn this problem into a state-based search problem.

STATE-BASED SEARCH PROBLEM:

States = ([*n-gram*], line#)
-- *n-gram*: last n-words in the current line, represented in a vector
-- *line#*: the current line that these words are in, represented as an int

Start State = ([‘START-LINE’] * n , 1)
-- our start state is a vector of n ‘START-LINE’ tokens.

End State = any state where line# = MAX_NUM_OF_LINES **AND** the last-word in the n-gram is an ‘END-LINE’ token
-- MAX_NUM_OF_LINES is a constant that specifies how many lines the generated rap song should be
-- When the line# in a state is at the MAX_NUM_OF_LINES, we know we are at the end of that line if the last word in the n-grams is the ‘END-LINE’ token.

Actions
The actions you can take in a state are:
--choose the next word to be in the lyric (sometimes choosing next word can be choosing END-LINE token)
-- start a new line

Successor States:
--if you choose a next word to be in the lyric, then the successor state is the next n-gram (the original n-gram with the first word dropped, and next word chosen appended) and the same line#
--if you start a new line, the successor state is the n-gram of START-LINE tokens, and the line#+1 i.e. ([‘START-LINE’] * n , line#+1)

Costs:

The cost for each action is broken down into three heuristics:

1) The n-gram words broken down into the respective bigrams, and calculating the bigram cost of the sequence of n-words (measure general fluency of language)

   Now the next 2 heuristics are only accounted for if the last word in the n-gram is the 'END-LINE' token and the line# is even (meaning we want this line to rhyme with the previous line, and also match it rhythmically in syllables)

2) If the ending word in this n-gram rhymes with the ending word in the previous line then we make this cost 0 or very low. If it does not rhyme, then we make this cost very high (100).
   a) We judge if two words rhyme because we make a bigram map of ending words in couplets from the lyrics in the dataset. If two words have this bigram mapping, meaning they were used as rhyming end words in the artists lyrics, then we consider them rhyming.
3) If total syllable count in the current (even) line# matches the total syllable count in the previous (odd) line#-1, then we make this cost 0 or very low. If the total syllable count differs greatly between lines then we make this cost very high. We can make the cost proportional to how greatly the syllable count differs (number of different syllables * 10).
   a) This is our heuristic for rhythm, which is an important feature of any rapper.

ALGORITHM:

We are using **Uniform Cost Search** on the Search-State problem as defined above.
The solution to this Search-State problem, as we have defined the problem, will then be rap lyrics that make up MAX_NUM_OF_LINES lines (16, as this is the standard number of bars in a rap verse), and these rap lyrics are the ones that produces the minimum cost (therefore meaning they are the most fluent, best rhyming, and best rhythmic lyrics that are close to the original artists lyrics).

DATA COLLECTION:

Because for some strange reason song lyrics are not available using the RapGenius API, we only use the API to search an artist and get the titles of their most popular songs. From there, we search the artist and each of their song titles to find the actual lyrics page for the song and scrape the lyrics from the site. Each song is then pruned of extraneous headers (the brackets denoting the start of a section on RapGenius like [Hook]) as well as any blank lines.

FEATURE EXTRACTION:

Features are especially important in rap, as some of them represent the very rules that allow the genre in the first place. The most important feature of rap is probably rhyme. Now there are currently no 100% effective ways to look at a word and see what words it rhymes with, we did find a way to learn what words rhymed. Because of the nature of rap, we can assume that every pair of lines rhymes (i.e. if we have lines 1, 2, 3, and 4, 1 rhymes with 2, and 3 rhymes with 4).

We then take the closing word of each pair and make another bigram with it. Words that appear more often in the list of rhyming words are more likely to be selected in the current model, and have a lower cost in the state search model.

Rhythm follows closely behind rhyme in terms of importance in Hip-Hop. Fortunately, we actually are able to count a word's syllables using the natural language toolkit. We can then use this prioritize words that would make their lines have a similar syllable count to the previous lines. In the state search model, we can represent this cost as the delta of current syllable count to the average number of syllables per line of the verse so far.

CONCRETE EXAMPLE:
We start in the start state: ([‘START-LINE’ * n] , 1). From there, we look through all of the possible next states, ([‘START-LINE’ * (n-1), chosenWord] , 1) and assign them transition costs, which is some function of ngramCost, syllableCost, and rhymeCosts (the syllableCost and rhymeCosts are only applicable if the line number is even and the last word in the n-gram is the ‘END-LINE’ token). We proceed like normal, incrementing the second number in the tuple whenever the last word in our ngram is ‘START_LINE’. This continues until we get to the end of the line.

Consider this example of ending an odd-numbered line. If we are in the state: (‘respect’, ‘my’, ‘power’, ‘END_LINE’, 5), we know we need to increment the line number by one, and bring us to (‘‘my’, ‘power’, ‘END_LINE’, ‘START_LINE’, 6). Because we are doing rhyming couplets, the odd numbers will start off the rhyme scheme, and the even numbers will finish them. This means there is no cost associated with not rhyming.

Consider this example of ending an even-numbered line (which we DO need to make sure rhymes with the previous line and matches it syllabically). If we want to get the cost of the state: (‘the’, ‘final’, ‘hours’, ‘END_LINE’, 6), then we check if the last word in this n-gram (‘hours’) has a bigram mapping from the last word in the previous line (‘power’). Since it does have a bigram mapping (power->hours), as seen in our dataset, then we add a cost of 0, since it rhymes. Then we get the syllables in all of line 6, (‘these the final hours’ = 5), and the syllables in all of line 5 (‘please respect my power’ = 6). The syllable difference between these is 1, so we add a cost of 1*10 = 10. So for the total cost of (‘the’, ‘final’, ‘hours’, ‘END_LINE’, 6), it is the bigram cost of “the final hours END_LINE” + 0 (rhymeCost) + 10 (syllableCost).

The final state will be when the number of lines = 16 (MAX_NUM_OF_LINES), and the last word in the ngram is ‘END_LINE’. An example of an endstate would be (‘respect’, ‘my’, ‘power’, ‘END_LINE’, 16).

PROGRESS:
While not monumental, our progress between the report and the milestone has laid out all of the groundwork for the future of our project. We have an actual way to query data, no matter the

artist, dynamically. The proposal was written up using manually extracted data. We have implemented all of the relevant feature extractors; important ones like rhyme and syllable count, as well as some others. And we have created a better verse. But still not as good as they oracle--yet.

EXAMPLE OUTPUT:

I recognize your mother be
Then you sell your defeat
If you eating the blood spilled on your life
Kendrick, A.K.A. "Compton's human sacrifice"
Playstation and greed; what's my drink, the other side may come up, fill up (Drank)
Never ever know you hate my head shot (Drank)
See, my life like I can help me finish this, love them, I recognize that want to diss but a job, boy with a swimming pool full of thirst, dying of a ghetto is beaming
I froze as it off the homies
The parade music we trip how the weed to suck my city, everybody's screamin' "Compton!"
How can be honest
Reeking the world on the message
And I feel you come soon, and weave
Then Usher Raymond "Let It got the gold
All the dollar in a motherfuckin' baller
Family history books overlook the blunt in my face and shit
These days of tumbling, tired of these motherfuckers, talking about my future

DIFFICULTIES:
The main difficulty we are encountering is that the instances of an artist not rhyming words because they aren't using rhyming couplets quickly overpower the ones where they are. This problem should be solved by adding in more artists and more songs, and if things get too terrible we can go in and try to prune skits and hooks to just focus on the verse.

FURTHER STEPS:
Now that we have gotten a basic n-grams going, the next step is to present our algorithm as a search state problem.

In addition to adding in n-grams costs, rhyme costs, and syllable costs, we will be looking for other heuristics we can add in as costs. For example, one heuristic we thought up of is how many slant rhymes/full rhymes to use. We have found a basic way to figure out some rhymes using nltk, and if we can figure out if words rhyme, then the rhyme is good enough to be considered a full rhyme. If we did this, we would have to put in some more work to the rhyme detection, and our cost would be a factor of the delta between the proportion of full rhymes and

our current proportion of full rhymes. Another feature we thought about is the amount of profanity in rap. We could once again find the optimal amount of profane words, and have costs be a factor of the delta between the current count of profane words and the optimal count.

From there we can also modify our code to take in more than one rapper, increasing the size of the corpus. What may be lost in personalization of the raps will be gained in improved performance to rapping in general.