Teddy Morris-Knower and Santiago Seira

CS221 Project Progress

2Pac Resurrected (actually though mixtape drops December 11th 2015)

The progress report will be comprised of three sections:

1. Current Model + Algorithm, where we're at right now
2. Results from Current Model
3. Future Iterations of the Model, what additions + adjustments we will make to model

1. Current Model + Algorithm

The initial model that we used to moderately improved on the baseline of randomly generated sentences is comprised of two parts, an n-gram based model for generating coherent sentences based on the work of the artist inputted to the model, and a model that takes "theme" words (words the user wishes to be the central theme of the rap) from the user and then changes the weights in the n-gram model to more heavily favor those words that are relevant to the theme words.

Data Extraction:

First we scraped data from allthelyrics.com using the BeautifulSoup python library to parse http responses. We took a per-artist approach which consisted of first finding a specific artist's page on allthelyrics.com. These artists' pages contained links to the lyrics of every song of the respective artist. Our scrapper first visited the url of the artist's page and extracted all of the links to the lyrics of each song. It then iterated through all of the links, visiting each one and traversed the DOM to extract the song title, album name and lyrics of the song which it stored as a song object and appended it to a song object list specific to the respective artist. We ran this algorithm for 5 artists and stored the data in JSON files available in our GitHub.

N-Gram model:

The n-gram model is loosely based off the n-gram program from CS106B. Given a list of words and a number n, create a map that maps n-1 consecutive words in the artist's work to the word that immediately follows those n-1 words. E.g. for a song "Happy christmas to all and to all a good night," and an n of 3, you would have a map of {{"Happy", "christmas" : "to"}, {"christmas", "to": "all"}, ...}. Note that if there are two of the same keys, then you simply append the new value to the end of the list of possible ending words. In the previous example this would be demonstrated with {"to", "all" : "and", "a"}. Once this map is generated, the sentences are generated by randomly selecting and initial key, from that initial key randomly selecting one of the possible values, and then updating the key by removing the first word and appending the value you just selected. An additional note is that the words in the value list are weighted by how many times they appear, so a word that shows up more often will more likely be chosen in the sentence generation. The words are also weighted by “song relevance,” which is described in the following section.

Model for song relevance

The goal of this feature is for the user to be able to input several keywords that will influence the n-gram freestyle generator in a way that the raps created follow a theme representative of the keywords. We are calling this feature song relevance and it works in the following way: First the user inputs a series of keywords which are grouped and treated as a song, we will refer to this song as *Keyword Song*. This *Keyword Song* is then compared with every song in our database to pick the songs which contain the greatest number of appearances of the keywords, we call these songs *Ideal Songs* (NUM_IDEAL_SONGS = 4). These *Ideal Songs* are then compared with every other song to compute a *Song Relevance* with regards to the *Ideal Songs*. The higher the *Song Relevance,* the more relevant the song is to the keywords and the more heavily we wish to weight the words of this song in our n-gram model. We accomplish this by using the *Song Relevance* of each song as a multiplier for its words when they are incorporated in our n-gram model. This results in the n-gram model having an inherent preference towards words that come from songs that are relevant to the keywords input by the user.

Algorithm for song relevance:

The algorithm for computing *Song Relevance* is dependent on a formula that computes the distance between two songs (sequences of words). We compute this distance in the following way:

First, we create feature vectors for every song. These feature vectors contain words of the song as keys and the number of appearances of the respective word as a value.

Second, we take the dot product of the feature vectors of the two songs being compared which results in a scalar that represents how relevant these two songs are to each other. The higher the dot product, the higher the relevance.

We utilize this approach for computing the relevance between two songs in our Song Relevance algorithm in the following way:

We first create a feature vector from our *Keyword Song* and then compute the dot product of the *Keyword Song’s* feature vector with every other song’s feature vector and choose the *Ideal Songs* based on the songs that produced the highest dot product with the *Keyword Song's* feature vector. We follow a similar approach with our *Ideal Songs* to compute the *Song Relevance* of all other candidate songs. For every candidate song, we compute its *Song Relevance* by calculating the sum of the resulting dot product with each of the *Ideal Songs.* We then normalize these values to have a range from [1:2] so as not to overpower n-gram model. Overall, it's similar to the approach the k-means clustering uses but instead of clustering all songs we rank songs based on the proximity to the *Ideal Songs* and this ranking (*Song Relevance*) is used to influence the n-gram model.

<u>2. Results from current model</u>

We will show a sample of the output (currently 32 words) with a given artist and "theme word"

Jay-Z, "roc": You can't touch me Jigga Kelly, not guilty Jigga Kelly not guilty. [Jay-z]: Uhh yea what is is (he's back). If you ain't got to be till one night well that's if you're.

2Pac: "compton": Over you got your head up and screamed. Fuck the world! They tryin to come find you ass fry don't ask me why Days go past and as a thug nigga though I walk.

Kanye West: "cash" Kids you give me Tribe Called Quest flashbacks and let's not even gonna go out with Me. Brains Power and Muscle like Dame Puffy and Russell. Your boy back on my arm I

Eminem: "kill" Zany enough to understand a little moonshine inside a ditch. There ain't gonna stop but you can't get by all your other Slim Shadys are just bad. South Park is gonna be able to

Lil' Wayne: "young" Fucking with the Phantom my niggas copin. Spurs my hands in the ocean I wear tha fuck out the ocean. I'm gonna save it. The south is so dirty bitch you can bath it.

From our metrics these are much better than random generation, both in terms of theme and in terms of sentence coherence. The next step is to improve overall sentence structure, and have actual "bars" with a rhyming scheme and syllable limits.

<u>3. Future iterations of the model</u>

The main things that we will have for the future model are adjustments to the song relevance model, and framing the rap generation not as randomly generating words, but as a search problem with states, actions and costs. We will also have some formatting changes, such as converting the Unicode strings that we have now back to a more readable Ascii version. We will also do some work with the song relevance to be able to get all forms of the base word given, i.e. if the input is "boat," be able to look for "boats," "boating" etc.

<u>Adjustments to the song relevance model:</u>

Create feature vectors based on the root of a word instead of exact word to be able to match similar words such as run, running and runners.

Fine tune the weighting and normalizing of *Song Relevance* to be able to influence but not overpower n-grams algorithm.

Find synonyms for keywords entered by user and add those to *Keyword Song* to be able to better find thematically relevant songs even if they don't use the exact words as the keywords.

<u>Framing as a search problem:</u>

We will still generate the map of n-grams using the same principle as earlier, but now the generation of sentences is not random with weighted words, but actually a search problem that attempts to find the best possible solution.

We will define the problem as a search problem with the following properties

Start: Random selection of one of the keys in the map.

State: Partially filled word vector.

isEnd: Fully filled word vector (full is defined by an input from the user as to how many "bars" or lines the rap needs to be).

Action: Appending another word to the rap by considering

Successors: all possible next words that can come after the last n words in the states, if they satisfy the constraints we will implement (e.g. rhyming, proper number of syllables per line).

Cost: (Max weight - Current Weight), of the word being added, so that a more heavily weighted word (through song relevance combined with number of occurrences) is more likely to be chosen.

Once the search problem is defined, we will solve it initially using Uniform Cost Search, since we are guaranteed to have non negative costs (since no current weight will be > max weight, min cost is 0), and then attempt to speed up the process by determining a heuristic to be able to use A*.

Github link to repository: https://github.com/sseira/2Pac-Resurrected.git