REFLECTION

Overview:
For project YUNGTWEETZY, Dennis Chen and I wanted to convert Twitter streams into a rap composition for any generic popular song instrumentals.

Implementation:
Our program works in a step-wise factory-like way. It starts with pulling partially-processed twitter feeds that are tailored to specific search terms. We filter these tweets to remove hashtags at the end of the tweets and remove punctuation. We filter or cut tweets to the appropriate amount of syllables using a pronunciation dictionary to find the number of syllables. Finally, we look for rhyming tweets using rhyme modules that somebody else wrote, and also by using some simple rhyme checking functions we wrote. Then, a program randomly generates a rhyme scheme, and we plug in the rhyming tweets we have into that rhyme scheme. The final results are shown below.

After we created a processed list of tweet rhyme groups, we were able to pass this through a series of rhyme constructors. The 'verse_generator' function creates a sixteen bar verse for a complete song. (A generic song is in 4/4 time and has the order of verse 1, hook, verse 2, hook, verse 3, bridge, hook.) As 'verse_generator' pulls rhymes out of the original input list, this rhyme bank shrinks so that there are no repeated verses. Later in the 'rap' function, we collect together the generated verses, hooks, and bridge to compose a complete song.

Results:
Here's an excerpt of one of our rap compositions.

"get to a Fridays to get one of these 10 burger deals
Teddy why dont you cook your dick Chris itd be a small meal
haleystokes its shit and shut the hell up all in one
saloon with us germans dont need an invitation
TimCowlishaw or a burger between two grilled cheeses
have a burger amp milkshake date at diners like this
Aaurorara Craving a bacon double cheese burger
difference between a cheese burger and a hamburger
a burgerIm pretty sure this is coded language
a seven toenail burger belt Sacramento tu
Hoochie burger how may I help youis yall niggas hiring
the bun we be up in this drive through order for two
ever was a HUGE mistake at 1230 on a Sunday night
I would go grab me a burger but its 20 degrees outside
burger home of the good burger can I take your order
burger house about three miles from my home Not talked 2 hunter"

And another, shorter one:

"Shes going to be sad when its gone
many memories with you rip sadie
cowardly dogyeah Im in the zone
there and go get that dog you always

a great lap dog Plus I have cool ears
guided her around for the past 5 years
w my dog bc boys are stupid
ACADEMY FUCK YOU eat dog shit"

Two issues are apparent here: the first is that the rap is generally nonsense, which is kind of to be expected, since we made no effort to ensure that the sentences would string together in a logical manner. The second is that occasionally, words just don't really rhyme. This is because we wanted our rhyme testing function to be smart enough to detect slant rhymes, so we had a low threshold for what is considered a rhyme. It's nice because it detects rhymes between words like "one" and "invitation", but returns false positives sometimes. The rhyme detector is actually black boxed and is a set of fairly sophisticated python modules we found online. I also implemented a very simple rhyme detector that returns absolute rhymes, because the modules we found were still problematic and returned false positives and didn't return some very obvious rhymes.

Reflection:

What went well?

Finding the resources to build our program turned to be much easier than we expected. Our biggest concern for this whole project is to determine whether we can actually group by near and slant rhymes. At first, we implemented a way using nltk to compare words to each other using the phonetic dictionary. That produced rhymes that were too close to each other. With additional research on the internet, we were able to gain a broader spectrum by adapting some program that was already written. This set of programs, written by Yat Choi, expanded the comparison range and provided some ways to weight close rhymes. Using this, we produced more rhymes for our composition. However, there are holes in these scripts, and there are occasional words that 'rhyme' but don't actually rhyme at all.

What could we improve?

To better organize our rhymes, we could progress by seeing which tweets were most relevant to each other. With time, things we could have tried are sentiment analysis and cosine similarity. Twitter also randomly refused our requests for tweets sometimes, and I'm not sure how we could improve on that, or if using another twitter api would work better, we'll have to experiment in the future. We also need to pickle data so that we have a database of tweets to draw from- when the program fails, it fails because it doesn't have enough rhyming tweets to work with. Our reach goal of having Microsoft Sam rap it rhythmically didn't occur either, but it shouldn't be too hard to implement, once we get everything else working really well.

What are some opportunities for better strategies?

We could have tried using some other rhyming servers to produce better rhymes. People have been implementing ways to score rhyme quality for a while. There are many different sources, and some are better than others. If we had more time, we could use more implementations and even come up with our own algorithms to produce sharper rhymes that make more sense.

We also could go back and refactor some of our programs to reduce computation time, Twitter API stress load, and also increase readability. We approached our program in a step-like manner, so we didn’t account for all the shortcuts that were available to us. Iterating on our script would generate more concise methods for creating our results.

Did we have a good plan for unit testing?
During the first part of our program where we categorize our twitter rhymes, we implemented quite a bit of unit testing functions to collect as many sensical rhymes as possible. The debugging experience during the actual rhyme scheme generation is completed based on print debugging. We wrote lots of unit tests and did a pretty good job documenting it!