Tali Tukachinsky

Intro to NLP

Verhagen

December 19, 2019

Final Report

For my final project, I attempted to build a classifier to discern from which decade a song was made based only on its lyrics. My goal was to determine whether lyrics alone are enough to classify music and to determine whether different decades had any meaningful differences in this regard.

The first thing I had to do to build my classifier was find some data to use for training. Originally, I was planning not only to have my classifier determine which decade a song is from, but also determine traits like genre. For this, I had to find a data set that included a song's lyrics, year produced, and genre. So, I first searched on kaggle.com, where there were multiple lyric data sets. However, none of these data sets had all the information I wanted as none contained genre information. In addition, many of these data sets had lyrics that were inconsistent in some way, often due to formatting. Most importantly, all the data sets I found were skewed in that the vast majority of the lyrics were from the last twenty or thirty years. Since I knew that decade classification was still going to be my main focus, I had to have a data set that was relatively even in this regard.

Therefore, I created my data set myself. In my code, this is found in the class DataCreation. I found online a package called LyricsGenius, which is a Python client for the Genius.com API. Genius is a website that contains lyrics to millions of different songs. The API

allowed me to retrieve the lyrics for any song as long as I provided it with the name of the artist and the title of the track. With the Genius API, one of my problems with the other data sets was solved as these lyrics were all retrieved from the same website, and thus all had the same formatting. In addition, I was able to stipulate that the client remove the section headers (such as "[Chorus]" or the name of a featured artist) within the lyrics and made the lyrics all lowercase to further normalize the format.

Now that I was able to retrieve the lyrics, I had to have a list of songs that solved my other problem, i.e. the lack of equality in the amount of data for each decade. For this, I found a folder (in the project https://github.com/kevinschaich/billboard) which contained CSV files of the top 100 Billboard songs for each year from 1950 to 2015. Using this, I could get a much more even spread of lyrics for each decade. So, I had my program read the CSV files. For each line of each file, I retrieved the lyrics to each song and, along with the song's title, year, artist, and position on the chart, I put the data into a JSON file, which I then dumped into data/song_data.txt. I added the other data into the JSON file because I wanted to have access to everything that the original CSV provided me with, but in the end I only used the lyrics and the year of the song, for the most part.

While this data was much better suited for my needs than any that I could find online, there are still some problems with it. First, as I mentioned, the CSV file only goes up to 2015. Because I was planning to split my data into decades, this was an issue since the 2010s would only contain half the data that the others had. Similarly, the 1950-1955 Billboard charts only included the top 30 songs of the year, as opposed to all the other years containing 100. Thus, my data was still relatively unbalanced. Still, I decided to use all of the data that I had access to

instead of making it even because that would drastically reduce the amount of data that I had access to. All together, in my JSON file, I had around 4500 song lyrics. This is less than the expected number, which is over 6000, because I removed any songs for which the Genius API couldn't find the lyrics.

I also attempted to find some API which could get the genre for these songs, but this ended up being very difficult. All the APIs I tried (including Spotify, MusicGraph, etc.) only included genre information for artists, not songs. They also often included multiple genres. Because of this, I was unable to find an intuitive and accurate way to label each song's genre, and thus I ended up not attempting to classify based on this trait.

Next, I had to build my actual classifier. This required building a feature vector for each song's lyrics. To begin, I had to read my own JSON file. I put the lyrics of each song into a list that I called 'x'. Each time I put new lyrics into this list, I also appended the song's corresponding trait. Since the trait I was going to be focused on was the decade, I took the year of the song from the JSON file and, knowing that all the years were in the same four digit format since I took them from the CSV files, I took the third digit from the years and added '0s' to make the trait. For example, '1975' turned into the trait '70s'. This string was appended to a corresponding list to 'x' called 'y'.

After the two lists x and y were made, I had to make sure, for the classifier to work, that x was a list of features. I also, therefore, had to decide what kind of features I should use. The first features I decided to use were fairly surface-level, but it was for these features that I needed the formatting of the song lyrics to be consistent. These features were percentage of distinct words per song, number of words, average number of words per line, percentage of repeated stanzas,

number of stanzas, percentage of repeated lines, number of lines, and percentage of rhyming lines. Since many of these traits were dependent on the structure of the lyrics, I decided it would be easier if I turned each song's lyrics into lists of lists. In essence, in my lyrics_format method, each song's lyrics were transformed to instead be a list of stanzas, which were lists of lines, which were lists of words. This was easy to split because, like I said, the formatting between the songs' lyrics was consistent. Each stanza was split by '/n/n' and each line was split by '/n'. To split the lines into words, I used regex to split at anything that wasn't an alphanumeric character except for the apostrophe, because I wanted contractions to stay as one word. Thus, lyrics that originally looked like this:

twinkle, twinkle, little star,
how i wonder what you are.

up above the world so high,
like a diamond in the sky.

Got turned into a list of lists like this:

[[["twinkle", "twinkle", "little", "star"], ["how", "i", "wonder", "what", "you", "are"]],
[["up", "above", "the", "world", "so", "high"], ["like", "a", "diamond", "in", "the",
"sky"]]]

This way, it was easy to iterate through the each stanza, line, or word, depending on which part of the song I wanted to analyze. I added each of those features I listed above to a new list called x2, which was a list of lists of features for each song's lyrics. I made this separate from x because I wanted to add features to x2 without getting rid of the original information found in x, which at

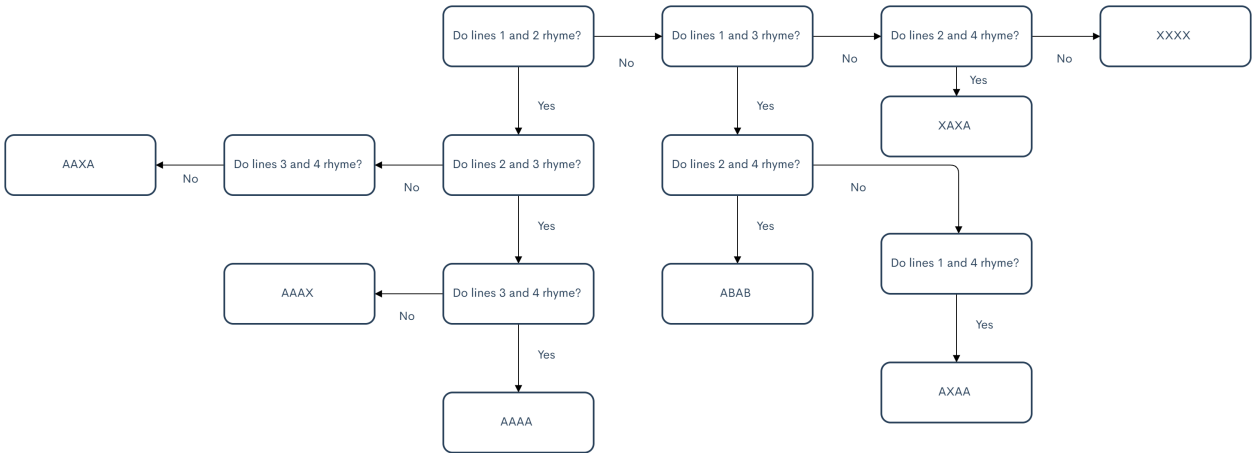
this point are the lyrics as formatted above. Because I iterated through x in order and enumerated it, x2 still corresponded with y, just like the original x did.

All those features above are for the most part self-explanatory, as I usually just counted the number of stanzas/lines/words. To find the percentage of unique or repeating ones, I put the stanzas, lines, or words into a set and into a list and divided the length of the set by the length of the list to find the percentage of unique elements. If the feature I wanted was repeated elements, I would subtract this percentage from 1. The only one that worked a little differently was the percentage of rhyming lines. For this, I made a list of the last words of each line. Then, for each of these words, I made a list of all their possible rhymes using the Pronouncing library. If these lists overlapped, that meant that this line had a rhyme somewhere within the song. I also made it so that the same word used at the end of two different lines would also count as a rhyme.

This feature, I think, has several problems. First, songs will often use slant rhymes or otherwise imperfect rhymes which aren't accounted for by the Pronouncing library, at least as far as I could tell. Also, this didn't account for internal rhymes, i.e. rhymes within single lines of a verse, as opposed to end rhymes. Finally, since I used all the last words of each line in order to determine the percentage of rhymes, this didn't necessarily mean that these words were meant to rhyme in the song. For example, if the first line of a song and the last line rhymed, even if in the song they were sung minutes apart, my algorithm would still count this as a rhyme.

Therefore, when my classifier using these traits only yielded an accuracy of about 20% on average (note: this is still better than random classification because there are 7 possible decades to choose from), I decided to add some more features, which would primarily work to solve the last issue I mentioned in the paragraph above.

I decided to have a trait that would account for the rhyme scheme in a song. For this, I referred this article (musicnotes.com/now/tips/enhance-your-songwriting-with-these-rhyming-schemes) which I found online. To summarize, the article stated that there are generally eight main rhyming schemes that occur in the first four lines of each stanza. It referred to them as AAAA, AAAX, AABB, AAXA, ABAB, AXAA, XAXA, and XXXX, where two letters A or B that are the same mean that those two lines rhyme, whereas an X means that that line doesn't rhyme with any of the four. I decided that for each song, I would test the first four lines of each stanza to see if it fit any of these rhyme schemes. If it did, then that would make the binary trait that refers to that specific rhyme scheme positive. I decided to use binary traits for this because I felt that the presence of the rhyme scheme was more important than the number of times it appeared, as that could have too much correlation with the number of stanzas in general. I tested the stanzas for rhyme schemes using the method rhyme_scheme_test, which was a series of if/else statements that operated based on whether the two lines I specified rhymed. This was essentially operated like a flow chart. Below is a simplistic representation of the way my algorithm functions. Notably, the one in my program includes options for a 'none' rhyme scheme if the specifications are not adequately met for any of those listed.

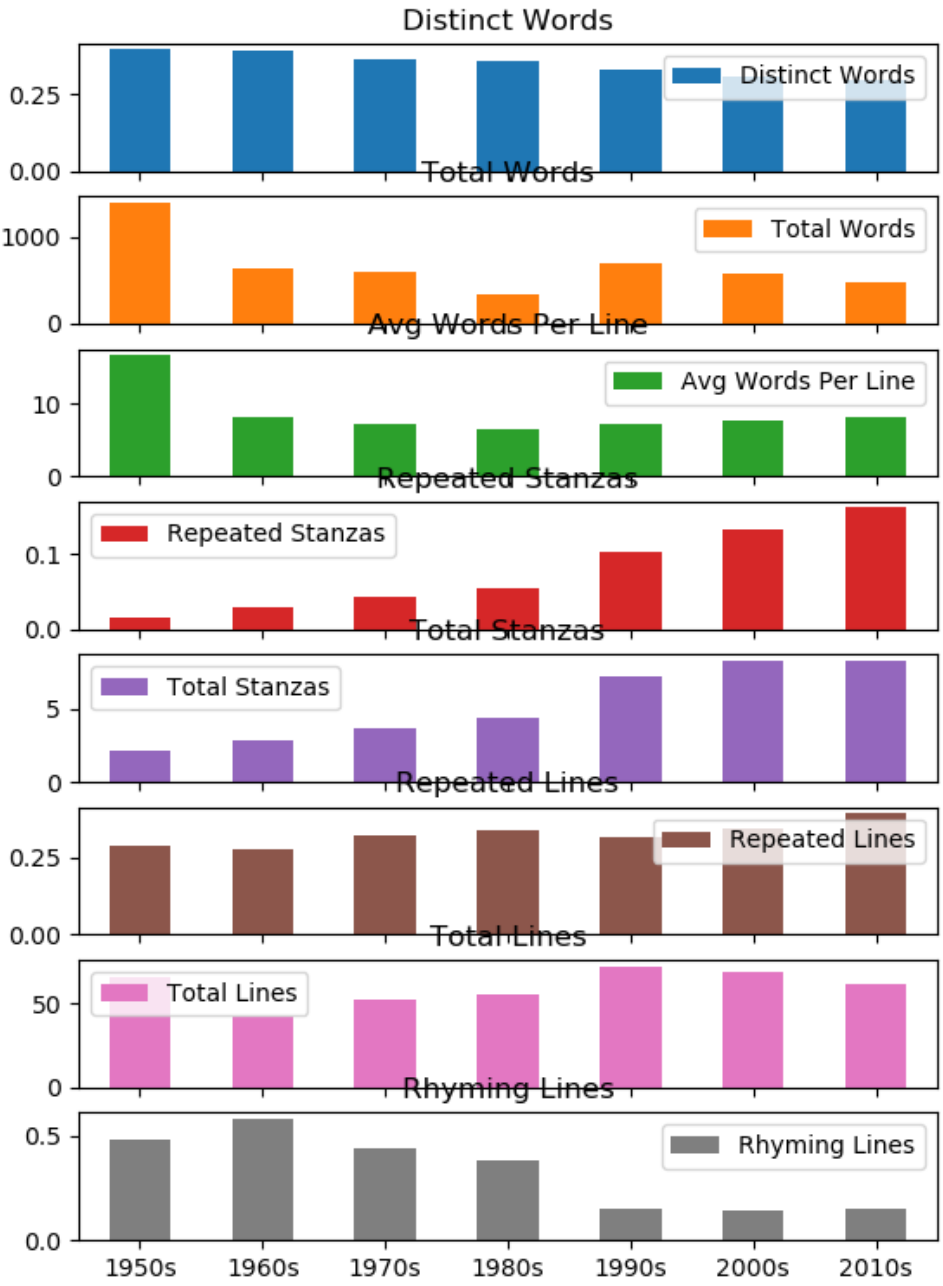


Once I added this feature (or rather, this list of eight binary features), my classifier became much more accurate, at 36%. Still, I wondered if there were any other ways that I could differentiate between different decades. I decided to add some more binary features, and those were the mentions of each decade. I figured that it's reasonable to assume that a song that mentions the 1950s would be more likely to be from that decade, for example. Unfortunately, this only raised my classifier's accuracy by 1%, but this was expected because it is a very specific feature that relatively few of the song lyrics had.

I decided, after finishing the classifier, to also analyze the data that I had collected. First, I printed the average of each category for each decade. I did this in the method year_analysis by creating 7 different lists of 22 lists, one list for each decade. The lists within these lists referred to the 22 features that I had created. After populating these lists, I took the average of each of the lists within the 7 lists to create a list of averages for each decade, with the index of the list corresponding to the feature number. This resulted in 154 numbers being printed. While they were labeled, the fact that they were in a list made them hard to compare.

Thus, I also made bar graphs of each of the features. For the first eight, I made separate graphs because I expected the Y-axis of each feature to be different, due to some of the features (percentages) being between 0 and 1 while others, like the word count, could be in the hundreds. Below is the first graph that is printed in year_analysis.

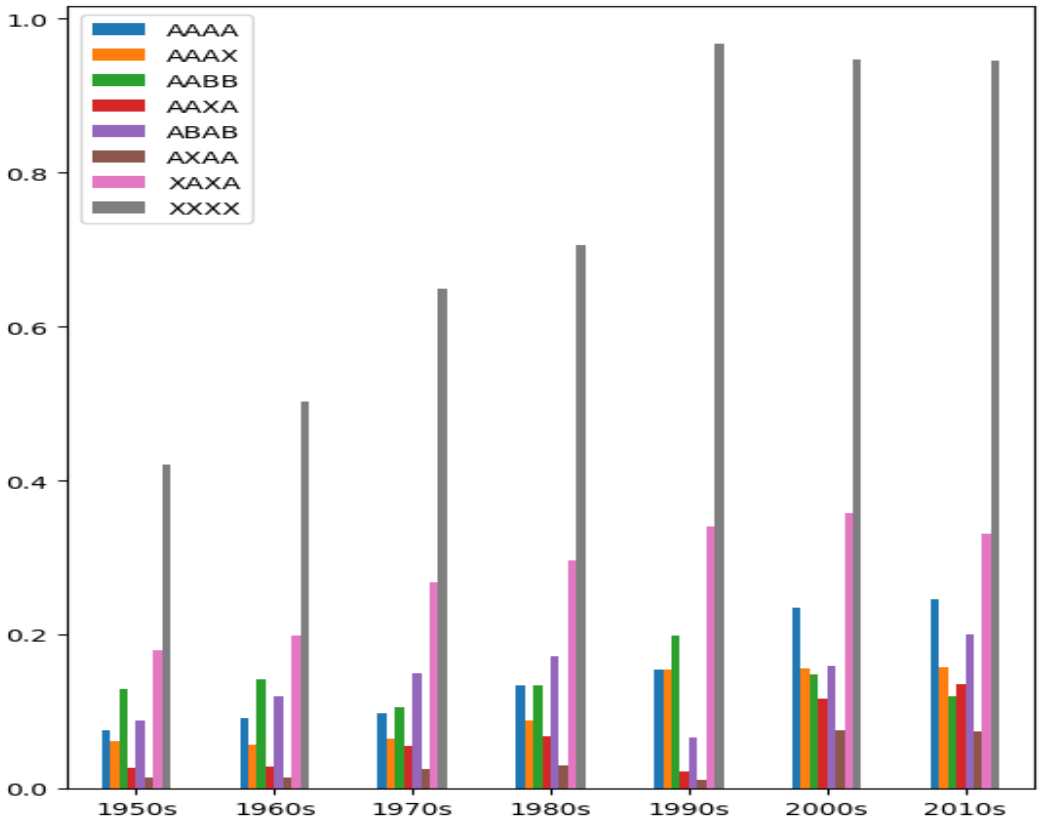
The fact that the bars were at different heights in essentially every category proved to me that each decade is indeed distinct enough to have its own features. I also found it notable that the total and repeated stanzas actually increased continuously over time, showing that there has



been a continuous progression in the direction of more stanzas per song. There were other categories, like total words and average words per line, where one category, the 1950s, particularly stood out. This should have made it easier for the classifier to classify the 1950s in

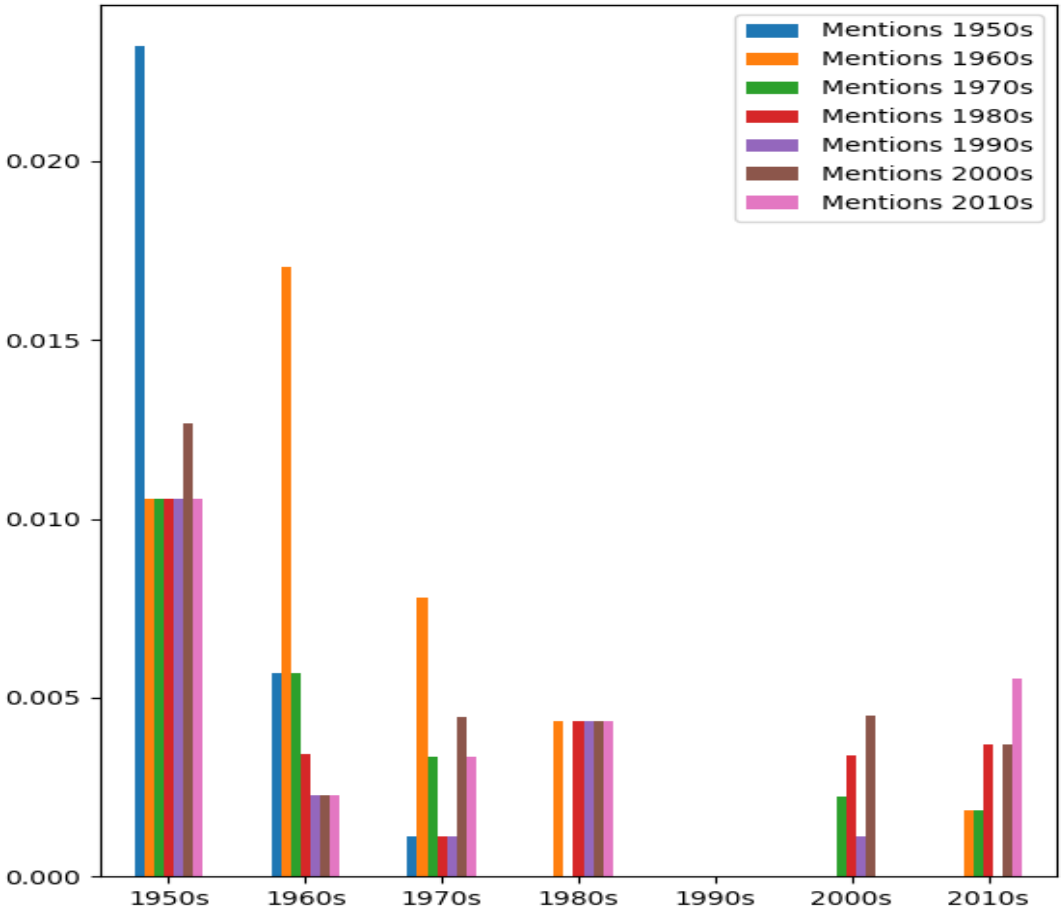
particular. Overall, I was pleased with the fact that the difference between most of the categories for most of the features seemed to be distinct.

I made a separate graph for the rhyme scheme.



At first glance, these seem pretty similar. However, looking at certain rhyme schemes, you can see that there's a continuous increase of, for example, the AAAA rhyme scheme as time goes on. There are also some anomalies, like the 1990s, where the ABAB rhyme scheme seems almost nonexistent, whereas it remains within the top 3 or 4 rhyme schemes of the other decades. Notably, the XXXX rhyme scheme seems to be the most prevalent. While it's possible that this is indeed common, I think that this is mostly the result of my system for finding rhymes being largely imperfect. As XXXX refers to none of the lines rhyming, I think it's very possible that I

was just unable to find the rhymes. Lastly, I also compared the year mentions. However, as I've mentioned, this feature category was largely unhelpful and as such the graph data doesn't show anything very meaningful. Notably, the 90s don't contain any year mentions, but this is not strong enough evidence to classify a song.



The last thing I did was test how my classifier classified specific songs that it hadn't seen before. I used three songs. Song1.txt came from 2014, song2.txt was from 1978, and song3.txt was from 1996. While my classifier labeled song2 correctly, it notably labeled both song1 and song3 as the 2000s. While this is wrong, my categories, since they are linear time, are actually quite dependent on one another. Thus, I found it notable that the classifier was only one decade

off. Thus, I thought it might be beneficial to include this data with the accuracy of the classifier. So, I used the method year_to_int to assign a numerical value 0-7 to each of the decades. Then, I calculated the average difference between the numerical value of the predicted category and actual one. The average distance was usually about 1.7, which I thought was useful to at least mention, as it provides more context to the 37% accuracy.

My overall conclusion is that while there are some differences between the song lyrics of each decade, it's still very difficult to classify songs into these categories using just lyrics. If I had more time, I would look more into developing a more accurate way to determine whether two words rhyme and extending that to include internal rhymes, as I think that was one of the more useful features for this classifier.