Chris Sciavolino (cds253)
Neel Taneja (nt297)
Ziyad Duron (zad22)

CS 3300 Project 2 Written Description

Data

The datasets we chose to integrate were parsed and put together with other various datasets we found online and with some manual scraping. In particular, we chose to utilize lyrical data in our project. When finding datasets for this project, we only chose to include lyrical data from studio albums, meaning EPs or anything similar were not included, because we wanted to limit the scope of our project and felt that an artist's main series of studio albums was most indicative of their music. Our eight datasets are described below:

1. *coldplaylyrics.csv*
    a. All of the data for this .csv were taken from the massive dataset of song lyrics called 55000+ Song Lyrics. We simply took the data containing Coldplay's lyrics and put it into its own .csv.
    b. Link: https://www.kaggle.com/mousehead/songlyrics/data
2. *kanyewestlyrics.csv*
    a. This lyrical data was taken from an online project that text-mined Kanye West. The dataset can be seen here. We took the lyrical data and put it into its own .csv.
    b. Link: https://github.com/zachsilvey/Text-Mining-Kanye-Lyrics/tree/master/Data
3. *keithurbanlyrics.csv*
    a. Some of the data for this .csv was taken from the previously mentioned *55000+ Song Lyrics* dataset, but it only contained about a third of this artist's entire discographical data. Therefore, we manually scraped (copy-pasted) the rest of the data from azlyrics and appended the data together.
    b. Link: https://www.azlyrics.com/
4. *linkinparklyrics.csv*
    a. Similar to the previous .csv, the *5500+ Song Lyrics* dataset only had partial lyrical data on this artist, so the rest was scraped from *azlyrics*.
5. *rihannalyrics.csv*
    a. Similar to the previous .csv's, the *5500+ Song Lyrics* dataset only had partial lyrical data on this artist, so the rest was scraped from *azlyrics*.
6. *taylorswiftlyrics.csv*
    a. Similar to the previous .csv's, the *5500+ Song Lyrics* dataset only had partial lyrical data on this artist, so the rest was scraped from *azlyrics*.
7. *pride_parsed.csv*

    a. We obtained this data from Project Gutenberg. Since the data is the text itself, we had to manually count all the words in the novel and save those word counts. We did this using a Python script and saved the results in a csv format, similar to the other datasets we found. The only two variables are *word* and *count*, and all of the words are sorted in count descending order.
    b. https://www.gutenberg.org/files/1342/1342-h/1342-h.htm
8. *stopwords.csv*
    a. The data for this .csv was taken from Natural Language Toolkit's (NLTK) own stopwords data, which can be found here. The data was was translated into a .csv format so that it could be read into our project. We also additionally added some of our own stopwords that we felt NLTK failed to include, for example the word "oh." We also added some custom words that appeared a lot in the manual scraping process. In particular, *azlyrics* liked to delineate who the speaker of the current verse was, and this was especially true of the band Linkin Park who has two main singers: Chester Bennington and Mike Shinoda. We included their names to the list of the stopwords so that they would not bias the data or appear as one of the most frequent words in our project.
    b. Link: https://gist.github.com/sebleier/554280

All of the .csv's had columns labeled *Album, Song, Lyrics*. Our initial project idea included accessing specific album data, but our project morphed over the timeline so we were really just concerned with the *Lyrics* column of data.

Data Analysis

We chose to asynchronously read in all eight .csv's and analyze the data within the callback function.

Data Variables

We first built an array with the stopwords and called it *stopwordArr*, which will be used later on in the lyrical parsing process. We also built two other arrays called *allData* and *allArtists*, the first of which is just an array of the read-in .csv data and the second of which is just a list of all the artists in string format.

Parsing the Lyrics

We ran a series of *forEach-loops* over each of the data within the *allData* array. The first *forEach-loop* made every set of lyrics per song lowercase, for consistency's sake. Then, each set of lyrics was split using a custom regular expression that only looked for letters and excluded anything else, including numbers, punctuation, and other various keyboard symbols. Splitting the lyrics on the regular expression created an array of the lyrics, which was then iterated over using

a second *forEach-loop* that checked if each individual lyric was in *stopwordArr*. If the lyric was found in *stopwordArr*, then the lyric was disregarded and only approved words were appended to an intermediate array of the form: [*Artist*, [*list of lyrics*]]. We chose to not use stopwords because it would be underwhelming to see words like "the" and "of" appear in the top ten words for each artist.

Finding Word Frequency

Taking the intermediate array from the previous section, we then used another *forEach-loop* to go over each lyric in the array per artist and count up its usage. This was done by creating a dictionary with each key being the lyric and each value being the number of times that word appeared in total. We then pushed all of this information into another intermediate array of the form: [*Artist*, [*dictionary of lyrics with counts*]].

Sorting by Frequency

At this point, we now had the lyrical frequency of every non-stopword per each artist, but the lyrics were unsorted and not particularly helpful. Once again, we ran a *forEach-loop* over the lyrics to sort them. Using a simple sort function, we were able to create a list of tuples where the first element of the tuple is the word and the second element is the word count. Thus, by the end of this final process, we had an array of the form: [*Artist*, [[*lyric1, count*], [*lyric2, count*], …]].

From Data to Visuals

The first graphic we have is just a visualization of Zipf's Law as static data. This is not meant to be interactive, but is just there to show the user what a Zipf's law distribution could look like. In this graphic, we used a linear scale for both the x- and y-axes and plotted a baseline curve that follows an exponential distribution. The thick line is meant to model the ideal while the small grey circles are meant to plot the data points seen in the *Pride and Prejudice* manuscript.

The majority of our data started out as word counts of words used in a given artist's aggregate music. In order to make our argument about Zipf's Law, we needed to show that plotting the usage among words followed a Zipfian distribution, indicating that the use of each subsequent word decays in usage each time. We allow the user to swap between two scales: linear and log. The linear scale displays the normal curve associated with Zipf's law, which the log-scaled usage follows the curve found on the Wikipedia page (linked above) associated with different *s* values.

(referenced link: https://en.wikipedia.org/wiki/Zipf%27s_law)

In order to allow the user to compare multiple artists, we gave each artist a distinct color on the bottom graph. Instead of having a key that maps artist to color, we overlay the color associated

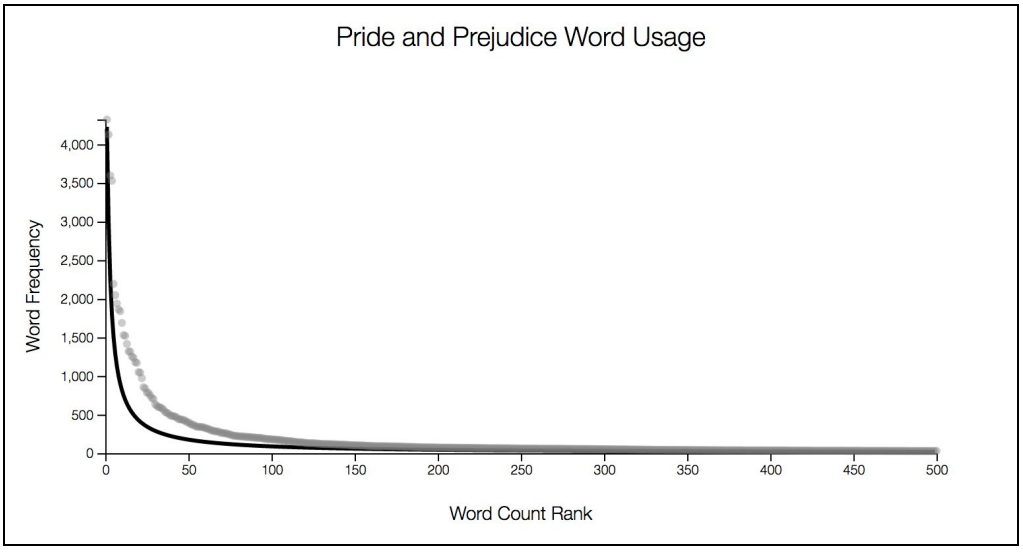
with the artist when the user adds that artist's lyrics to the graph. In order to be 100% clear, we additionally added an implied key in the descriptor paragraph right above the interactive SVG.

We also noticed a few words that varied across all of the genres and allow the user to interact with the data by highlighting words of interest in the graphs. This is done by adding a thick, bold, noticeable circle to the SVG on the point where the word appears. This circle has a larger radius and a higher opacity relative to the rest of the points, so they should be easy to spot. The highlighting point functionality allows the user to view trends of interesting words across genres and artists and draw conclusions from them.

Finally we incorporated a slider at the bottom of the graph that limits the number of unique word values that we go through. The slider has values from [50, 1100] inclusive, so the user can plot the first 50 words, the first 1100 words, or anywhere in between. We found that after around 400 words, the data points become a little too cluttered and are difficult to distinguish from other another. Because of this, we added the slider in order to zoom into the beginning words and analyze trends from that view instead. When zooming out, the data better follows the trends of Zipf's law, so having a slide allows for the best of both worlds (interactivity with the points and generalization of the argument at larger scales).

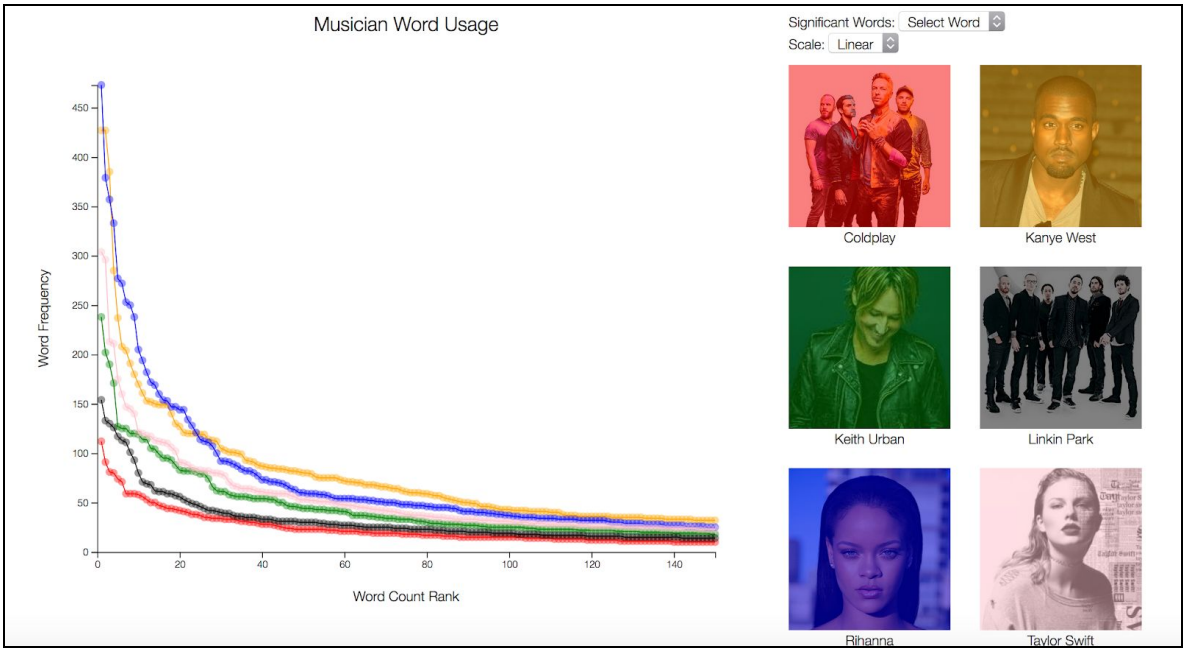
Narrative

We wanted to tackle Zipf's Law, an empirical mathematical law that states given a large enough sample corpus of words, the frequency of any word in the sample is roughly inversely proportional to its rank in the sample's frequency table. This law is most often applied to literature, as novels serve as perfect datasets for this law. Our first visualization demonstrates Zipf's Law on Jane Austen's *Pride and Prejudice*.



While not a perfect fit, the *Pride and Prejudice* visualization gives the user a clear understanding of the distribution of data under Zipf's Law. The highest ranked words in the corpus clearly have the highest frequency as well and the entire distribution seems to follow a negative exponential distribution (while this may visually be the case, it actually follows its own distribution called a Zipfian).

Now positive that Zipf's Law is an actual mathematical phenomenon and not something made up, we wanted to extend this concept outside of the literary realm and into one that was more exciting for us: music. Specifically, we wanted to test Zipf's Law against the lyrical discography of our favorite artists and see if the same trend is present. **We hypothesized that the law would indeed fit the data** but we also made a secondary hypothesis. We specifically chose six artists of different genres to see if the trend would be present across all genres or just one, and **we tentatively believed the trend would be similar across the genres chosen**. While it's difficult to make all-encompassing conclusions about a genre when it is being represented by one artist, we felt that the artists we chose represented their musical category quite well (especially given the difficulty of acquiring lyrical data).



We decided to create an interactive data visualizer and let the user come to a conclusion by themselves without us holding their hand to the conclusion. In the image above, it's evident that the Zipfian curve is a good fit for the data, but far from a perfect one. The reason is the nature of the law itself. Zipf's Law is a 1/r function, which naturally creates decimals with odd, integer data. For example, the most occuring word might occur an odd number of times, thus half that

number would create a decimal, but word occurrences can only be integers. Compound this problem hundreds of times for more and more words and the data becomes less and less perfect.

It's clear that the law is a pretty good approximation for the distribution of the words within all of these artist's discographies. Our initial hypothesis seems to be proven correct by the data. All of the artist data seems pretty consistent. Looking at the visualization through the lens of our second hypothesis, we also seem to be proven correct. There does not seem to be any bias toward a particular genre except for the words used across them, but these have no evident effect on the actual relationship between word rank and frequency.

Conclusion

At the beginning of this visualization we set out to prove Zipf's Law is not only present in realms like literature, but also in the musical realm as well. After presenting a proof of concept with *Pride and Prejudice*, we dove into the lyrical with tentative hypotheses about the results. Playing around with the data in the interactive visualizer gives credibility to our hypothesis. Based on the lyrical data of these artists, *Zipf's Law appears to be a good approximation between the frequency and rank of lyrical data*.

Surprises

While parsing through the data ourselves, we found several interesting words that helped us compare and contrast artists/genres beyond the Zipf's Law narrative. In particular, the word *love* occurs incredibly frequently across all artists except Linkin Park, and in some cases, like Keith Urban and Rihanna, it is their top-occuring word. Perhaps this has implications about the type of music these artists are crafting. Regardless of the reason, the common occurrence of *love* across most of the artists was very surprising.