Billboard-Top-100-Lyrics

Juhee Lee (jl2755), Yinan Wen (yw654), Kevin Schaich (krs252)

April 28, 2016

Introduction

Our project researched and visualized how lyrics and associated data of popular songs have evolved since 1950. We grabbed the top 100 songs on Billboard for each year, and used natural language processing to analyze a variety of metrics. Users can interactively choose a year/genre range they are interested in to get a closer look at subtleties.

Data

Crawling/Analysis

Billboard Top 100 Songs

Our only initial dataset comes from Billboard's Top 100. We grabbed a **CSV file** of the top 100 songs for the years 1950 - 2015 from reddit's r/datasets.

Lyrics

Using **Wikia Lyrics**, as well as its Python counterpart on Github, **Heroku API** we scraped each song's title/artist combination and downloaded the song's full lyrics.

Please note that this data is not intended to be a full set, as we ran into some problems along the way with slight inconsistencies in our dataset's naming schemes vs. the API's request structure. Many lyrics for older songs in the 1950-60's are less readily available online as well, however we have about 80-90% coverage for all the songs on Billboard's list in our year range.

Genres/Tags

Using the **MusicBrainz API** as well as the Python interface **Musicbrainzng**, we scraped each song artist's associated genre tags. These tags are quite numerous and extensive, so we came up with a total of 15

'*aggregate genres*' based on their total occurrence rate in all our songs to represent the aggregate of our data and to keep our visualization clean. A minified sample of these aggregates can be found below:

```
aggregate_genres = [
{"rock" = ["pop rock", "jazz-rock", "heartland rock", ...]},
{"alternative/indie" = [...]},
{"electronic/dance" = [...]},
{"soul" = [...]},
{"classical/soundtrack" = [...]},
{"pop" = [...]},
{"hip-hop/rnb" = [...]},
{"disco" = [...]},
{"swing" = [...]},
{"folk" = [...]},
{"country" = [...]},
{"jazz" = [...]},
{"religious" = [...]},
{"blues" = [...]},
{"reggae" = [...]},
]
```

Sentiment Analysis

Using the **Natural Language Toolkit (NLTK)** for Python, we used the **VADER model** for parsimonious rule-based sentiment analysis of each song's lyrics. Each song was run through a sentiment analyzer and output an object with data about its sentiment:

```
"sentiment": {
    "neg": [float],             # Negativity assoc. w/ lyrics. (between 0-1 inclusive, 1 being 100% neg
    "neu": [float],             # Neutrality assoc. w/ lyrics. (between 0-1 inclusive, 1 being 100% neu
    "pos": [float],             # Positivity assoc. w/ lyrics. (between 0-1 inclusive, 1 being 100% pos
    "compound": [float]
}
```

The `pos`, `neg`, and `neu` are the three interesting values. Each value represents the percentage probability that the song is associated with a positive, negative, or neutral connotation and sentiment, respectively. Using the positive and negative values, we are able to tell whether a song's lyrics lean towards *"happy"* or *"sad"* in demeanor.

Readability Metrics

Using the **textstat** package for Python, we calculated a number of aggregate readability metrics associated with each song's lyrics:

```
"num_words": [int],             # Number of words in lyrics.
"num_lines": [int],             # Number of lines in lyrics.
"num_syllables": [int],         # Number of syllables in lyrics.
"difficult_words": [int],       # Number of words not on the Dale-Chall "easy" word list.
"fog_index": [float],           # Gunning-Fog readability index.
"flesch_index": [float],        # Flesch reading ease score.
"f_k_grade": [float],           # Flesch-Kincaid grade level of lyrics.
```

While the top few are most explanatory, the **Gunning-Fog Index** and **Flesh-Kincaid Grade Level** are the most powerful. Both of these metrics use a variety of linguistics data like average sentence length, word, length, and complexity/number of syllables to determine the readability of a text. These metrics allows us to graph the trend over time for specific genres, i.e. you would need to be in 2nd grade to understand the average pop song from 1972.

Repetition

For each song, we count the number of duplicate lines that appear in the lyrics. This can be used as a rough measure of repetition in the song content, i.e. the more duplicate lines in the lyrics, the more repetitive a song is.

Data Summary

Our aggregate data JSON file includes all of the following metrics:

```
{
    "title": [string],              # Title of the song.
    "artist": [string],             # Artist of the song.
    "year": [int],                  # Release year of the song.
    "pos": [int],                   # Position of Billboard's Top 100 for year [year].
    "lyrics": [string],             # Lyrics of the song.
    "tags": [string array],         # Genre tags associated with artist of the song.
    "sentiment": {
        "neg": [float],             # Negativity assoc. w/ lyrics. (between 0-1 inclusive, 1 being 100%
        "neu": [float],             # Neutrality assoc. w/ lyrics. (between 0-1 inclusive, 1 being 100%
        "pos": [float],             # Positivity assoc. w/ lyrics. (between 0-1 inclusive, 1 being 100%
        "compound": [float]
    },
    "f_k_grade": [float],           # Flesch-Kincaid grade level of lyrics.
    "flesch_index": [float],        # Flesch reading ease score.
    "fog_index": [float],           # Gunning-Fog readability index.
    "difficult_words": [int],       # Number of words not on the Dale-Chall "easy" word list.
    "num_syllables": [int],         # Number of syllables in lyrics.
    "num_words": [int],             # Number of words in lyrics.
```

```
    "num_lines": [int],             # Number of lines in lyrics.
    "num_dupes": [int]              # Number of duplicate (repetitive) lines in lyrics.
}
```

Data Aggregation/filtering

Another python script aggregates our data by year for easy filtering in real-time in JavaScript. Our output JSON file looks similar to the above song format with the following structure:

```
{
    {
        "year": 1950,
        songs = {
            {song object 1},
            song object 2}},
            ...
    }
    {
        "year": 1951,
        songs = {
            {song object 1},
            song object 2}},
            ...
    }
    ...
}
```

This structure is cleaned and minimized and the original lyrics are removed to keep our file size under 2MB for nearly 5000 songs. Using underscore.js, we are able to utilize functional programming in JavaScript to very quickly filter and sort through our data. Using the above JSON notation with year-oriented objects allows us to filter through nearly 5000 songs in real-time on the user side in fractions of a second.

We also have a `filter(data, year, year, genres)` function in our JavaScript code which takes in our user's parameters and spits out an aggregate list of the data we need for our visualizations with a single function call.

Visualization

We have 2 graphs for different parameters.

Graph 1: Line Plot

The first of which is a line plot, which shows a variety of parameters on the y-axis with time mapped on the x-axis. Any of the following above parameters can be mapped to show their trend over time as well as their relation to genre.

To create this graph, for each genre we filter out the data for a specific year and calculate the average. For each data point on our line, we filter out values that are either infinity or negative infinity so that they do not skew our data towards the origin. We do this for all years and form an aggregate dataset to create our lines. We repeat this process for each genre to form a trendline for all genres, giving us our aggregate dataset. Each genre gets its own color and is aggregated into a legend on the right side of the chart.

Graph 2: Scatter Plot

Our second graph is a line plot which relates time on the x-axis to Billboard rank on the y-axis. This shows the relationship and relative popularity of genres to time, and we can see varying concentrations of genres in different eras of time.

To create this graph, we filter out the data for each genre and graph a point for each song. We filter out values that are extraneous so that they do not occur off of our axis, and we do this for all genres to form our aggregate dataset. Each genre gets its own color and is aggregated into a legend on the right side of the chart.

Interactivity

Data Visualizations

On the upper part of our first graph, we have a dropdown toggle that allows users to select between any of our numerical parameters to graph on the y-axis. Upon selecting a new metric, we update the dropdown text to display the current metric, as well as a helpful overview of what the metric is displaying and how to read it, i.e. what low values vs. high values mean as well as what sort of data range to expect.

We also update the graph itself, and recalculate the y-axis domain/range based on the min/max average values for all our datapoints. These changes are made through d3 transitions so we get a nice smooth fade in between each one. We update the axis label for the y-axis accordingly, also with a transition.

Static Filter Bar

On the bottom of our page, we have a static footer that allows users to filter based on a year/genre range they would like to narrow in on. This filter bar can be expanded/retracted and when a user clicks on that button the page's scroll value shifts up/down to keep the graph in the same page as before the expansion/collapse.

Year Filtering

There is a dual-handle year slider that allows users to focus on whatever year range they would like. Upon updating of this slider either via the keyboard or mouse drags, we update the graph accordingly, stretching our axes and limiting our dataset only to the years selected by our user.

Genre Filtering

There are a variety of genre toggle buttons that allow users to toggle which genres they would like to show on the visualization, and whenever these buttons are clicked we remove/add the corresponding line with a transition. We also update the y-axis scales accordingly with new min/max values in our entire dataset selected.

Mobile/Responsiveness

All of our visualizations are responsive, when the window is resized everything is resized dynamically so that our user always views the best experience for their screen. We also use media queries and aspect ratios to calculate the width/height of our elements and charts.

User Feedback

When any of the parameters are adjusted, we use **hammer-time** to remove the 300ms delay from responsiveness on mobile webkit devices. We also use **Nprogress.js** as well as a page cover fade to convey calculations to our user, so that they know when our page is loading/calculating new data. This provides an optimal experience where our page doesn't simply 'freeze' when a user requests a new set of data/filters.

Observation

The patterns in our first line graph were not as obvious as we expected. However, we do notice some interesting observations:

- **Songs are getting longer over time.** The number of lines, syllables, and words have all been steadily increasing for most genres in 2015 relative to 1950.
- **Most song lyrics are very easy.** We see a huge trend around grade 1 for the Flesch-Kincaid grade index, indicating that most readers don't need above a first-grade education to understand the lyrics being portrayed.
- Song lyrics have not seen a huge increase or decrease in intelligence
- **Songs are getting sadder over time.** We see a very slight degradation in sentiment from 1950 to 2016, however some genres are different than others. This is not as strong of a correlation as the first few.

- **Songs are getting more repetitive over time.** We see a large trend for most genres tended towards more duplicate lines as we move from 1950 to 2015. This is in line with our expectations of popular songs on the radio, as many of the "catchy", trending songs heard on the radio or Spotify have an easy-to remember chorus that occurs a lot of times.