CSE 564 Assignment 2a Lab Report

visualizing my music taste

# Dataset

I have used the same dataset as in Assignment 1. It contains audio analysis and other metadata (artist popularity, number of tracks in the album, etc.).

# Implementation

## Server

The server is implemented in the Python programming language. I’ve used the flask library to implement a REST API server that serves data needed by our graphs. This data is always in the most convenient format with respect to the graph requesting it.

I’ve used the scikit-learn module’s excellent toolkit to perform PCA and k-means clustering. I’ve also used numpy for linear algebra tasks, and pandas for dataframe goodness. These libraries work well in tandem with each other.

I’m not that well-versed in Python, so there are a few anti-patterns, e.g., no main method and use of global variables. Regardless, it gets the job done.

Here is a trace of the logs generated which shows the format of the requests made from the client to this server.

127.0.0.1 - - [03/Mar/2023 19:37:54] "GET /screedata HTTP/1.1" 200 -

127.0.0.1 - - [03/Mar/2023 19:37:54] "GET /labels HTTP/1.1" 200 -

127.0.0.1 - - [03/Mar/2023 19:37:54] "GET /biplotdata HTTP/1.1" 200 -

127.0.0.1 - - [03/Mar/2023 19:37:54] "GET /columndata?di=5 HTTP/1.1" 200 -

127.0.0.1 - - [03/Mar/2023 19:37:54] "GET /actualdata?cols=energy,release\_date,loudness,acousticness HTTP/1.1" 200 -

127.0.0.1 - - [03/Mar/2023 19:37:55] "GET /elbowplot HTTP/1.1" 200 -

Every time we select a new dimensionality index, new API calls are made as such.

127.0.0.1 - - [03/Mar/2023 19:58:55] "GET /columndata?di=8 HTTP/1.1" 200 -

127.0.0.1 - - [03/Mar/2023 19:58:55] "GET /actualdata?cols=energy,key,total\_tracks,release\_date HTTP/1.1" 200 -

## Client

The client uses React.js. Each graph is its own React component. For generating graphs, d3.js is used. I’ve also used Material UI which provides good-looking React components.

The biggest challenge with using JS is wrangling with asynchronous events. All API calls are asynchronous. They provide a promise-based API which lets us define actions that will be performed once the respective async task completes. Updating React state is **also** asynchronous. Forcing async events to behave without erroring out was an interesting task to tackle.

There are a few implementation features that I would like to highlight.

* I’ve adhered to Spotify’s design language as much as I could. Hence their trademarked green/white/black palette.
* For the two clusters, I picked colors that would contrast well against green.
  + I’ve always referred to Pokémon for color palette inspirations because they’re very visually pleasant (and tested on perhaps the most visually-sensitive demographic known to mankind – young children).
  + For this color palette, I referred to <https://pokepalettes.com/#deoxys>

# Insights

* Audio analysis data provided by Spotify comes in the form of dimensions which are already quite orthogonal to each other. There is very little correlation, and often there isn’t any at all.
* This means the generated PCs themselves can only capture very little variance, each. This is why our scree line plot does not have a discernible elbow.
* It is interesting to see how selecting higher dimensionality indexes leads to different dimensions from the original data space taking the spotlight.
  + At di=10, the cumulative explained variance is about 81%. The top four columns are all categorical variables – mode, key, time signature, tempo. They describe the more technical, clear-cut musical aspects of a track, in contrast to the more nebulously defined variables such as energy and acousticness.
* There is a clear elbow at k=2. After that, each iteration of k leads to reduced WCSS (as expected), but it drops comparatively quite slowly.

# References and Documentation

Tools used for implementation are:

1. Spotify APIs - <https://developer.spotify.com/documentation/web-api/>
2. Postman (API calls) - <https://www.postman.com/>
3. d3.js - <https://github.com/d3/d3/wiki>
4. ReactJS - <https://reactjs.org/docs/getting-started.html>
5. Material UI - <https://mui.com/material-ui/getting-started/overview/>
6. Flask - <https://flask.palletsprojects.com/en/2.2.x/>
7. Scikit-learn - <https://scikit-learn.org/stable/modules/classes.html>
8. Axios (for HTTP requests on node.js) - <https://axios-http.com/docs/intro>
9. REST - <https://www.restapitutorial.com/>