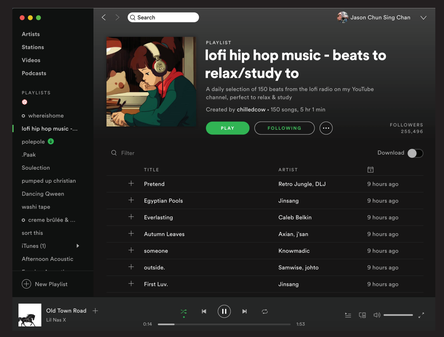
Moodify for Spotify®

By Nazem Droubi, Martin Chu, Kevin Xiang, Jason Chan

Introduction & Goals

Have you ever wondered how Spotify makes their recommended playlists? Don't you love the process of creating your own?

Our project aims to use Spotify past user-generated playlist data and Spotify song audio features to group songs into similar topics and be able to generate our own data-driven playlists.



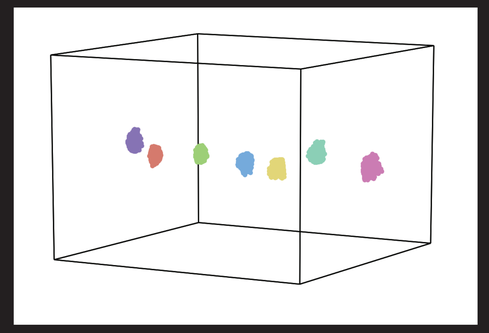
The Spotify Data Set

Our project is based on a **Spotify data set of 4000 *user-generated* playlists** (containing in total **over 2 million songs**). This data is stored as a JSON object: each playlist is a dictionary of its features, with one feature being a list of tracks within said playlist. Our project uses this JSON, alongside the Spotify API, to construct two corpuses of data:

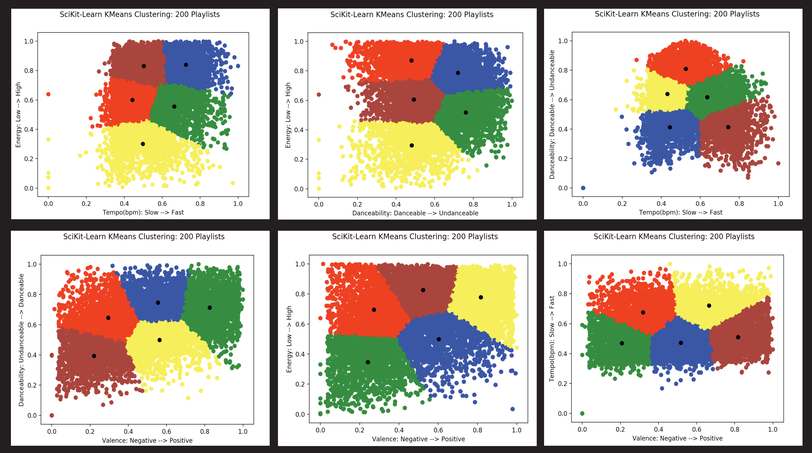
1. A **mapping from playlist titles to track titles** within that playlists. This would be used later to implement a **playlist generation model based on the DL word vector algorithm**.
2. A **mapping from song titles to a predefined set of audio features** that we estimated to be most important (***tempo - valence - danceability - energy***). We find these by making API calls to the spotify server through the unique track IDs of each track. This will be later used to implement a **playlist generating model using KMeans clustering.**

Song Clustering By Audio Features (KMeans)

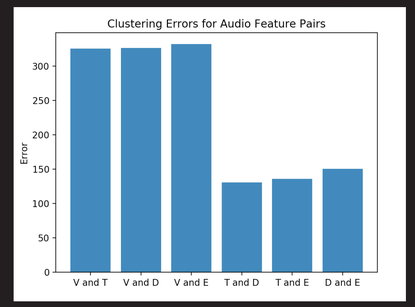
For our second playlist generating program, we decided to **cluster tracks based on their audio features' similarity**. That is, tracks were similar to each other if the difference between their audio features was small. Since audio features have different scales, we normalized each feature by it's maximum existing value. We decided to implement KMeans clustering in 2 different ways:



1. We **clustered songs based on all 5 features and used PCA dimensionality-reduction algorithm** to reduce our plots to 3 dimensions. We ran the program for a range of N-clusters and used the accumulated errors to plot an elbow graph and **concluded that the optimal number of clusters is 7**. Scikit-learn and Hypertools were used to create the graph on the right.



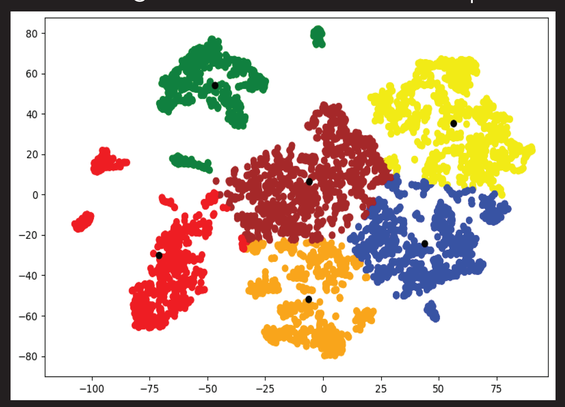
2) We used all **6 possible pairs of features from our 4-feature** set and clustered based on each of them to get the 6 clustering graphs on the left (**e.g. Energy & Danceability, Valence & Tempo, etc**). We **compared the clustering errors** of each pair, as seen in the graph below. We concluded that the pair of audio features resulting in the lowest clustering error is: **Tempo & Danceability.**



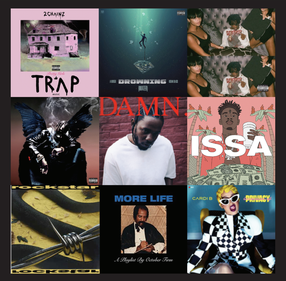
Song Recs by Playlist Similarity

Our first method of determining similarity between songs was through using **song vectorizations and context windows**.We **determine the closeness of songs by how often they appear in the same playlist together**.

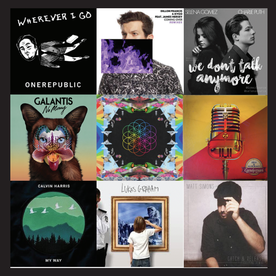
In other words, this method represents **similarity as determined through the user generated playlists** - representative of user opinions. Each song is assigned a song vector which determines how often every other song appears in context of the same playlist as the song. We find the distance between songs using a **cosine distance metric** between the two vectors. This allows us to determine the most similar songs and be able to find the top most similar songs to a particular song.



Next, we applied **TSNE dimensionality reduction** algorithm to map the distance vectors into two dimensions most representative of the variation. Then we ran K-Means clustering on it. The result, on the left, from **6-cluster K-Means** is our song set divided into 6 topics of songs that are most similar to each other.



"Rap"

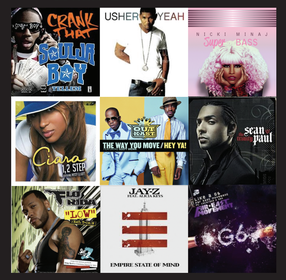


"Pop"

Finally, we ran **K-Means** using **500 clusters to result in 500 playlists of the most similar songs**. Here are examples of playlists generated by our program. Playlists are self-labelled.

"Christmas"

"Throwback"



Design Choices

1) We needed to make **75,833 API** calls to retrieve each track's audio features. This was extremely slow since the Spotify server enforces a limit on calls per user. To solve this, we had our data collector program run overnight and **saved the audio feature data into a database**. We chose to construct a .db database rather than a csv or JSON file because it **allows us to manipulate and query the data more efficiently**.

2) In order to visualize and convey the information of the playlists that our programs created through our two methods, we **created album cover visualizations** similar to the ones Spotify creates.

3) Since we were clustering on many dimensions (e.g. the five audio features, and the many dimensions of the song vectorizations) we decided to use **PCA and TSNE dimensionality reduction** to graph in 2D and 3D.

Conclusion

We determined from clustering errors that the combination of audio features that best represents song similarities is **tempo and danceability**.

Using two methods, 1) similarity by user generated playlists, and 2) similarity by audio features, **we created 500 playlists each of similar songs**.