Milestone 3: Revised Project Statement and EDA

AC209a Project Group 21 - Project D (Spotify Playlist Generation)
Team Members: Matthew Finney, Kaivalya Rawal, Royce Yap, David Zheng

Description of the Data

Million Playlist Dataset (MPD) Playlists	We started with Spotify's Million Playlist Dataset (MPD) to learn about the nature of playlists in order to identify other tracks that a user might enjoy in the context of an existing "stub" playlist. The MPD, released by Spotify in 2017, contains a random sample of 1 million playlists from the 2 billion public playlists on Spotify at that time. Each row in the data represents a song on a playlist, containing a unique playlist id `pid` as well as the `track_uri`, `artist_uri`, and `album_uri`.
Spotify API (API) Song Features Artist Information	We used Spotify's Web API (API) to scrape track-level data for each song contained in the MPD, as information about tracks in each playlist could be useful to train a model that recommends or discriminates between songs. We used the `track_uri` field in the MPD to call each song's `uri` in the `audio-features` API. The `audio-features` object contains estimated features of the song (Acousticness, Danceability, Energy, Tempo, etc.). We also used the `artist` API to gather Artist information for each track, namely the `genres`.

Data Cleaning and Reconciliation

Our source data was well structured, and contained primary/foreign key pairs that facilitated cross-referencing information from the MPD and API. The main objectives of our data cleaning and reconciliation process were to:

1. Extract data from 1,000 individual Million Playlist Dataset CSVs into a single dataframe.
2. Cross-reference the MPD with the API to incorporate artist and track features for analysis and prediction.
3. Filter playlists that would not be good candidates for training or prediction due to their small size or the relative obscurity of their tracks.

We took the following data cleaning and reconciliation steps:

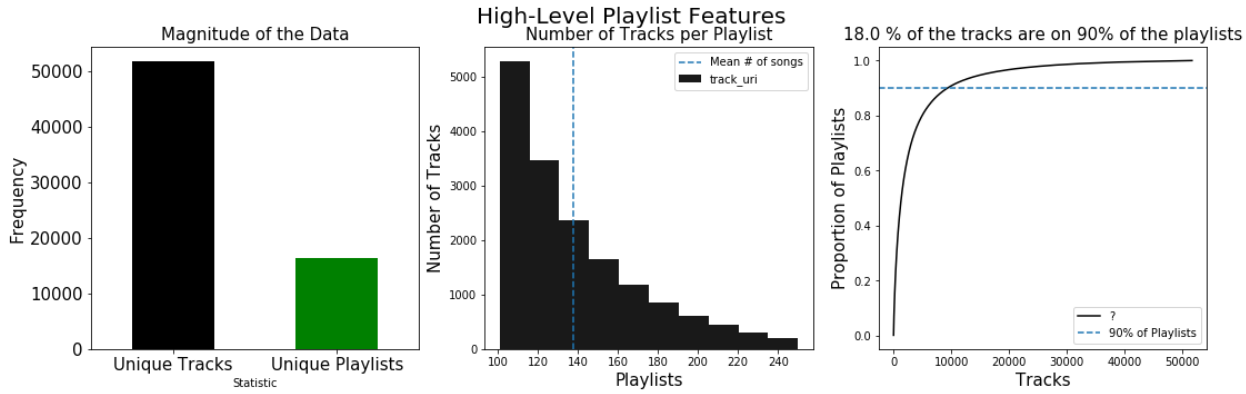
1. Combine the Million Playlist Dataset CSVs into a single dataframe
2. Assign a unique `pid` each playlist, such that the first 3 characters are the serial number of the source data CSV and the last three characters are the original `pid` in the source file.
3. Count the number of appearances of each `track_uri` in the Million Playlist Dataset
4. Create track dataframe by calling the `audio-features` API on the top 100,000 `track_uris`
5. Create artist dataframe by calling the `artist` API for unique `artist_uris` in track dataframe
6. Downselect playlist dataframe to those with >100 tracks and containing only songs from top 100,000 tracks
7. Cross-reference the playlist, artist, and track data using `track_uri` and `artist_uri` as foreign keys

Exploratory Data Analysis

Initial Explorations: What is the composition of playlists in the cleaned dataset?

First, we investigate the high-level composition of our cleaned dataset. How many playlists and tracks do we have? How many tracks are on a playlist? Do some tracks appear more than others? These inputs help to scope further EDA and model selection

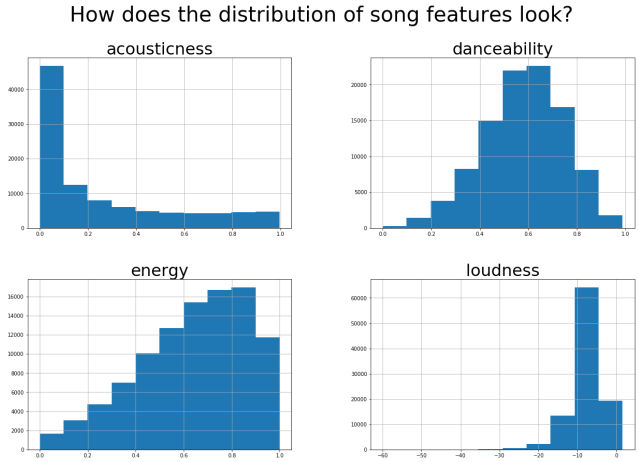
.

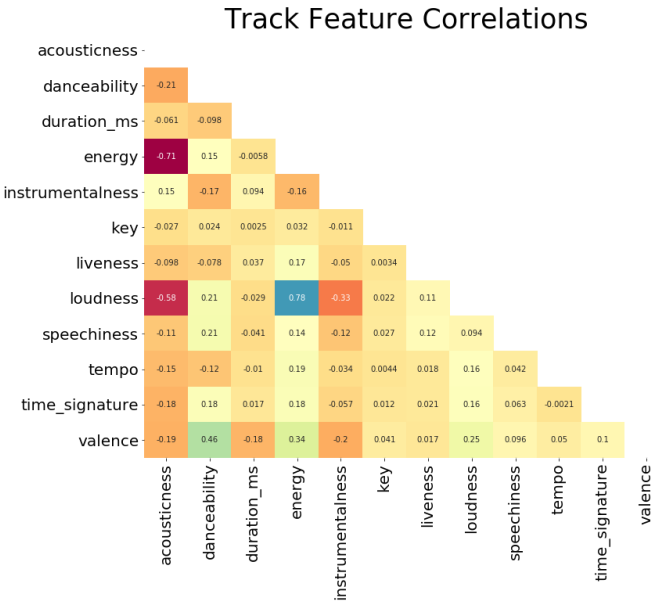


The plots above show immediately the impact of our playlist selection criteria on the dataset that will be used for training, testing, and validating our models. The number of songs per playlist is not normally distributed, and we only have playlists that include more than 100 songs. Additionally, 18% of songs are on 90% of playlists, so we can expect overlap of songs between playlists.

What relevant data do we have about the tracks?

In this section we tried to analyse the song level data, to guess what feature transforms may be required when fitting models later. We wanted to see the distribution of the features, to analyse things like outliers and missing values, and see if we could use our results to guide any potential feature selection or feature augmentation choices. For brevity, we present a subset of the feature histograms here, clearly showing that the distributions vary largely. For instance, acousticness seems to follow something akin to a power law distribution, while danceability is more normal. Some features like loudness have a lot of values in one region, and a few potential outliers.

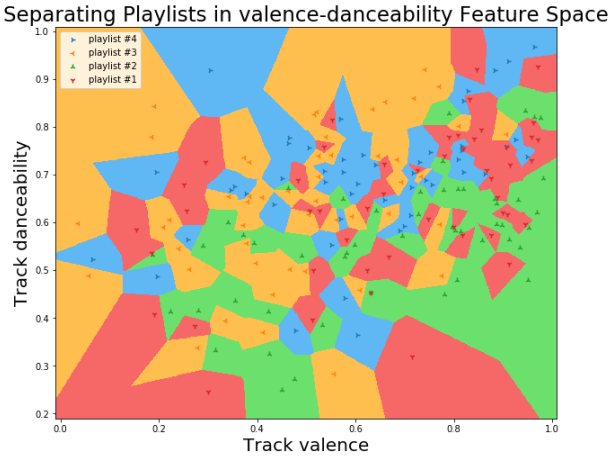




Next, we examine feature correlations to find any redundant features. We find that acousticness is negatively correlated with loudness and energy. Loudness is in turn correlated positively with energy, and negatively with instumentalness. Apart from these, there seems to be no particular multicollinearity, and no features can be discarded entirely as redundant. We now hypothesize that these features are more similar within a playlist, and different across playlists, and examine these claims further in the following sections.

Do track features differentiate playlists?

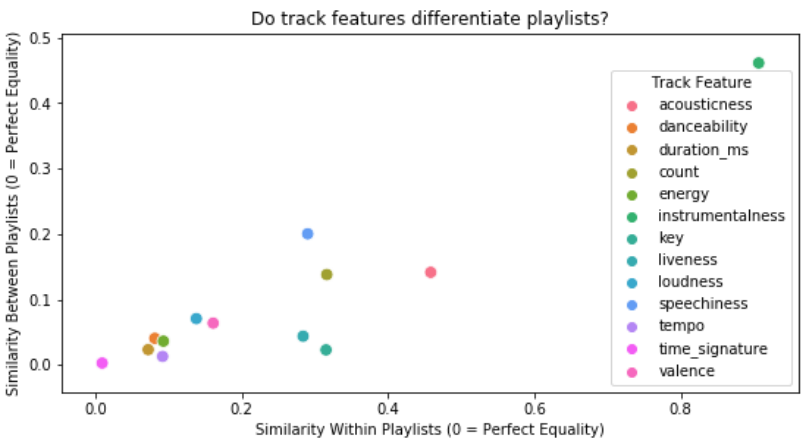
Finally, we investigate the relationship of songs in a playlist, and between playlists. We wanted to know whether playlists in our training set tend to be built from songs that are similar to each other or songs that are different from each other. If playlists tend to be composed of songs that all have similar features, we could use information about the distribution of feature scores in a stub playlist to recommend additional songs for the playlist.



First, we consider if playlists separate in song-feature space, based on the features of individual tracks. We find that, for various arbitrarily selected pairs of song features, there was little semantic separation of playlists. A visualisation for one such pair: valence and danceability. Note that most of the points are in the center, where there is extremely high variance in colors. It seems like regions on the plot edges are acceptable, but this is actually due to dearth of data (large regions, defined by single points), and is not representative of the songs genuinely falling into clusters.

Next, we consider whether the average features of an individual playlist differentiate it from other playlists. We calculated the gini index for each song feature by playlist. To measure the average within-playlist similarity of songs, we took the mean of the gini index for each playlist. To measure the average between-playlist similarity, we took the gini index of the mean feature scores for each playlist.

We find that, as illustrated in the above scatterplot, playlists are, on average, composed of songs that have similar features, particularly danceability, duration, energy, tempo, and valence. However, our exploratory data analysis doesn't find that playlists vary greatly in individual features. This could be because the majority of playlists in the sample data used to generate this plot are actually similar to each other; for example, this could be the case if most playlists are composed of Top 100 pop songs. However, if this is instead because even playlists with different moods/contexts have songs with similar features, then this could mean that a model that discriminates between new songs based on the features of songs in a stub playlist struggles to match the intended context/mood of the stub playlist.



Given that individual song features do not differentiate individual playlists, we next plan to consider the group characteristics of groups of playlists. We propose k-means clustering on the full training dataset in an attempt to algorithmically identify natural clusters of playlists in high-dimensional space, which could in turn be used to build a distribution of songs likely to mesh well with a playlist in a given cluster.

Revised Question

Starting with a stub playlist of songs chosen by a user, can we recommend additional songs for the user by suggesting songs that appear in similar playlists?

Baseline Model Description

For our baseline model, we fitted an unsupervised k-Nearest Neighbors classifier on a randomly selected training set of 100 playlists, which contained track features for 3098 unique tracks. The predictor we built takes a list of tracks as an input, finds nearest neighbors for each track, and returns the list of "neighbors" (recommended songs not in the original input) that have the highest classification probabilities.
To test our baseline model, we selected another subset of 100 playlists at random. In this test set, we split each playlist into two subsets: calibration (20%) and withheld songs (80%), where the calibration songs are used as the input for our predictor. These recommended songs output by the predictor are then compared to the withheld songs, using R precision score as a comparison metric.

	No. of neighbors	No. of predictions	R-precision score
0	10	10	0.002523
1	10	50	0.005493
2	10	100	0.008523
3	50	10	0.006577
4	50	50	0.006710
5	50	100	0.007639
6	100	10	0.006718
7	100	50	0.006487
8	100	100	0.006978

We tuned the model across across different values for the parameters of number of neighbors and number of predicted songs to return. We computed the average R-precision score for each set of selected parameters (number of neighbors and number of song predictions) by averaging the R-precision score across playlists for each set of parameters. The highest value returned across the combinations of parameters was 0.008, which implies that on average, out of all the withheld songs in the test playlists, we only manage to predict 0.8% of the actual songs using our model. This is extremely low - however, as a baseline model, this model gives us a sense of the challenges involved in predicting songs, such as the fact that most of the withheld songs in the test data set are not in our original training set.

More details can be found in the attached Jupyter notebook outlining the steps we undertook for the baseline model.

Next steps

One of the significant challenges we identified could be the fact that the withheld songs in the test data set are not in our original training set. Hence, logically, the model trained on songs in the training set would not be able to return predicted songs that would match 1-to-1 with the withheld songs. Test and training sets that considered a total of 200 playlists proved efficient for computation, however, given a cleaned dataset of ~16,000 playlists and 100,000 unique songs, these samples may have been too small.

Going ahead, for the next milestone, this will mean expanding the training set to increase the overlap in songs between the two sets while exploring other potential models. We will also explore the use of K-means clustering in future iterations to increase the likelihood of choosing songs from playlists that are similar to the given list of songs.