Team 137

Recsys 137

Sujay Thakur

Simon Warchol

Jovin Leong

Data description

- Base data
    - Our base data is the Million Playlist Dataset that we obtained from the course staff; we will most likely be primarily using this dataset until we've done up our system's functionality to a point where we will require additional data to make improvements to our system
    - Base data features
        - Playlist ID: Which playlist each track corresponds to in the CSV file
        - Position: What is the position (index value) of each track within each playlist
        - Artist name: Name of artist for the respective track
        - Track URI: Uniform Resource Identifier with unique track ID
        - Artist URI: Uniform Resource Identifier with unique artist ID
        - Track name: Name of respective track
        - Album URI: Uniform Resource Identifier with unique album ID
        - Duration (ms): Duration of the track
        - Album name: Name of album in which the respective track is in

- Augmented data
    - Using our base data, we've fed the unique track identification codes through the Spotify API for Python (Spotipy) and extracted additional song features for each song. This is done through our features_extract.py module that we have also included in our Milestone 2 notebook.
    - Subsequently, we created an augmented dataset that basically extends our initial set of songs from the Million Playlist Dataset by including additional features (columns that contain the value corresponding to each metadata metric provided by the Spotify API) for each song (each row). We essentially have the same dataset with additional features for each observation (each song).

- Augmented features (beyond the base features)[1]:
  - Danceability: Describes how suitable a track is for dancing based on a combination of musical elements including tempo, rhythm stability, beat strength, and overall regularity. A value of 0.0 is least danceable and 1.0 is most danceable.
  - Energy: A measure from 0.0 to 1.0 and represents a perceptual measure of intensity and activity. Typically, energetic tracks feel fast, loud, and noisy. Perceptual features contributing to this attribute include dynamic range, perceived loudness, timbre, onset rate, and general entropy.
  - Key: The estimated overall key of the track. Integers map to pitches using standard Pitch Class notation; E.g. 0 = C, 1 = C♯/D♭, 2 = D, and so on. If no key was detected, the value is -1.
  - Loudness: The overall loudness of a track in decibels (dB). Loudness values are averaged across the entire track. Loudness is the quality of a sound that is the primary psychological correlate of physical strength (amplitude). Values typicalLY range between -60 and 0 db.
  - Mode: indicates the modality of a track, the type of scale from which its melodic content is derived. Major is represented by 1 and minor is 0.
  - Speechiness: Speechiness detects the presence of spoken words in a track. The more exclusively speech-like the recording, the closer to 1.0 the attribute value. Values above 0.66 describe tracks that are probably made entirely of spoken words. Values below 0.33 most likely represent music and other non-speech-like tracks.
  - Acousticness: A confidence measure from 0.0 to 1.0 of whether the track is acoustic. 1.0 represents high confidence the track is acoustic.
  - Instrumentalness: Predicts whether a track contains no vocals. “Ooh” and “aah” sounds are treated as instrumental in this context. Rap or spoken word tracks are clearly “vocal”. The closer the instrumentalness value is to 1.0, the greater likelihood the track contains no vocal content.
  - Liveness: Detects the presence of an audience in the track; a higher value implies a higher probability that the track was performed live.
  - Valence: A measure from 0.0 to 1.0 describing the musical positiveness conveyed by a track. Tracks with high valence sound more positive, while tracks with low valence sound more negative.
  - Tempo: The overall estimated tempo of a track in beats per minute, i.e. the pace of a track which derives directly from the average beat duration.
  - Type: The object type: “audio_features”
  - ID: The Spotify ID for the track
  - URI: The Spotify URI for the track
  - Track href: A link to the API endpoint providing full details of the track.

[1] Based on Spotify’s developer notes:
https://developer.spotify.com/documentation/web-api/reference/tracks/get-audio-features/

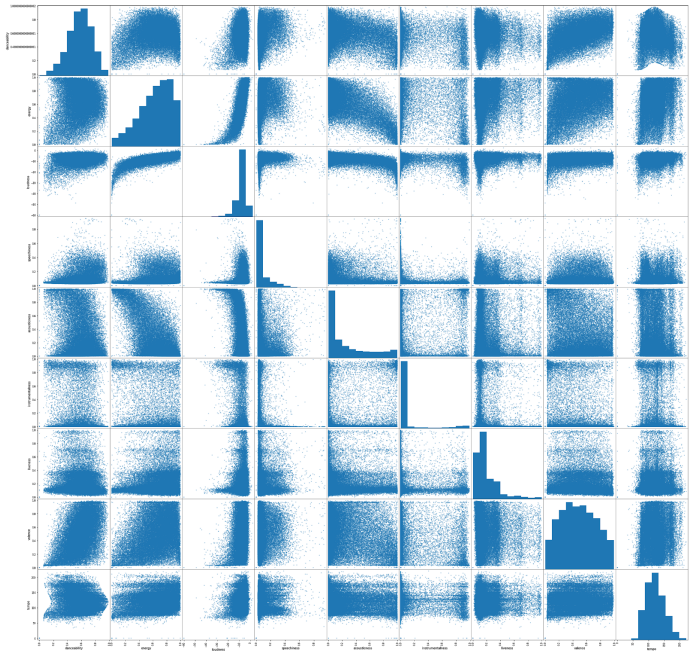
        - Analysis URL: A URL to access the full audio analysis of this track.
        - Time signature: An overall estimate of how many beats are in each bar.

    - Data cleanliness
        - Generally the data is quite clean in that we do not have many null values or incorrect formats—the Spotify API does a good job of having values for all features and ensuring that they are of the right format.
        - The Spotipy API has issues with reading certain songs from our Million Playlists Dataset; this is largely the case when songs in the dataset are no longer on Spotify, when the track ID has changed, or when some feature of the song generates a conflict. We've skipped past these error cases and extracted features for all other songs and we've noted down which songs are left out of our dataset.
        - Certain songs in the Million Playlists Dataset are repeated; we dealt with this algorithmically, by getting our programs to disregard repeated entries

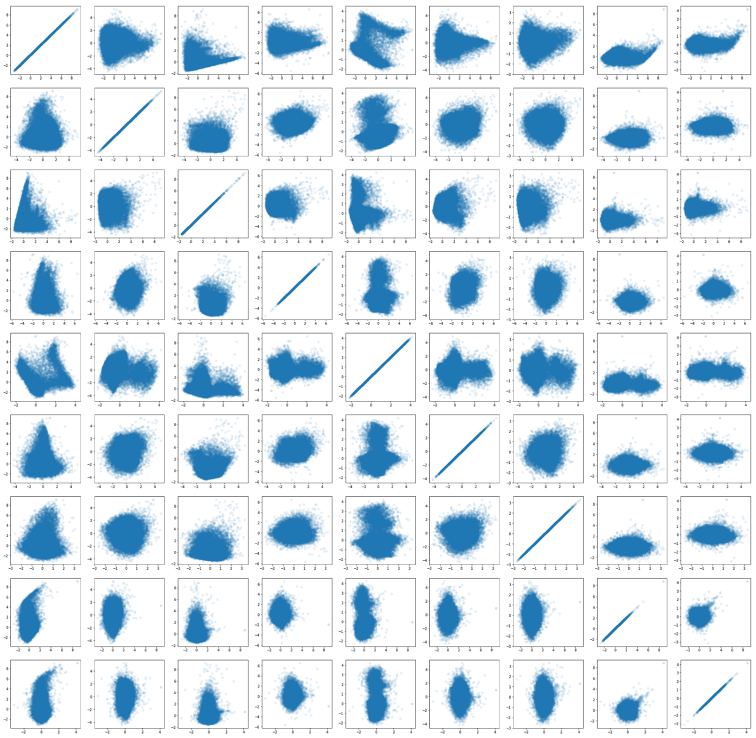
Exploratory Data Analysis

- Motivation:
    - At this juncture, our main aim was to develop a system that takes user inputs and generates a Spotify playlist of with recommended songs. The recommendations will, in turn, be based on the metadata metrics from our augmented dataset.
    - As such, we wanted to examine the metadata in order to see which metrics should feature (and carry what weight) in our recommender system.
- Looking for correlation:
    - We take in subsets of our entire dataset and read the data with all its attributes; we then store these values as Numpy arrays and standardize our data

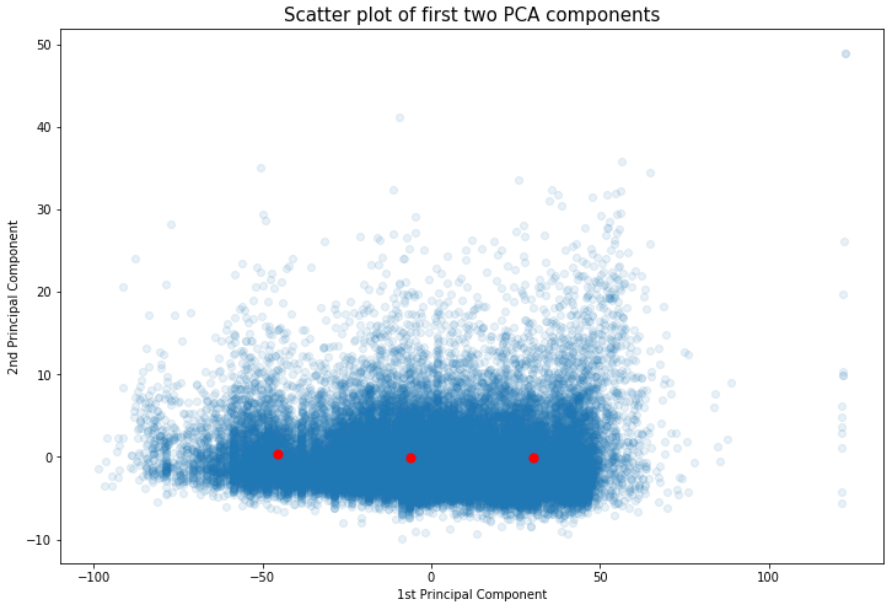
We can plot the correlation scatters for all attributes. Note that features like energy and loudness seem to feature collinearity. We can choose to deal with this by either dropping one of the attributes or by downscaling one.



Since the data has already been standardised, we can also compute the PCA vectors and visualise all pairs in a matrix. This would give some intuition for the number of clusters we might expect in the data.



We see roughly 3 clusters, and can now fit these using K-means to get some intuition. We note that the learned cluster centers correspond correctly to the 3 clusters we visually saw in the PCA projections.



Based on our visualizations and from our reading of the Spotify developer notes, we find that it is likely that some of the attributes in the Spotify data are correlated and that our model will

exhibit multicollinearity if we include all variables. As such, in subsequent stages of development, we will more rigorously test for multicollinearity (such as ANOVA) and we will either weight the variables appropriately or drop them entirely.

Additionally, the clustering is insightful because it allows us to understand what parameters we should include in a K-means model.

Baseline Model

Our baseline model is mostly written in Python scripts, but we've broken it up into various components that are workable in our Jupyter notebook that we've included in our submission.

Our baseline model presently takes in a track choice and searches for all playlists where the input track can be found and assess all other tracks within this set of playlists and considers how the audio features of these tracks compare with that of the input track. We can fit a GMM and NN to this data. This allows us to generate recommendations based on two techniques.

GMM: We can fit the data to the chosen number of clusters we expect. Then, when the user likes a song, we can infer the posterior $p(zn|xn)$ to select the cluster that the song belongs to. We can then sample $p(xn|zn)$ to generate a new song from that cluster. Since this sample from a multivariate Gaussian may not exactly correspond to a song, we can use our pre-trained NN to find the closest match.

NN: We can simply run unsupervised NN on the data and recommend the k closest songs corresponding to the song that the user likes.

Note that we would expect NN to generate recommendations closer to the song input, since it is literally picking songs closest to the input, whereas GMM would recommend songs in the same cluster but potentially far away from the input. Our system could use both techniques together, with NN suggesting relevant similar songs and GMM suggesting relevant 'exploratory' songs for the user to better explore that cluster.

In other words, our model presently generates two lists of song recommendations by using two separate techniques. Our system is presently able to make reasonable and intuitive predictions - though we've yet to perform more robust tests. Our system is also able to take in those lists and generate Spotify playlists that are completely usable in the Spotify app or web app.

Revised Project Statement

- Insights from EDA
    - Our data itself does not contain any user information or ratings that we can use as outcome variables on which to train a model on; the closest proxy we have to this is the playlist IDs that each song belongs to - though playlist membership says very little about each song. We cannot, given our existing model, develop a system that is trained on user information and/or inputs without somehow either collecting user input or expanding our dataset; we can, at this juncture, rely only on unsupervised models.
    - As such, we can, at best, use the data for a nearest-neighbours approach with some classification model to make recommendations. We can try to relate the playlist IDs to the song features to make recommendations. Any more complex model will require data collection or generation.
- Our new approach
    - To clean and process data as before and use our methods of analysis to understand which features and which weights we assign to each feature will generate useful recommendations.
    - Our focus, then, will be on refining our model and creating the necessary interface to effectively translate user input into filled Spotify playlists
    - Once this base functionality has been achieved we will create a system that is able to automatically generate Spotify playlists with minimal user input and we will have made efforts to use this sparse data to overcome the Cold Start problem. Subsequently, we will attempt to improve our system in the following ways
        - Make the interface more user-friendly and accessible; be able to catch more errors and mistakes
        - Implement additional inputs that users can make which are factored into the recommendations. Consider more creative approaches to gather this information (like Buzzfeed-type quizzes).
        - Generate data or gather more data such that we can implement a more sophisticated trained model where we have a toy dataset with user ratings, user information, etc that we can use to train our system on and generate richer recommendations.