Jon Zimbel

CS 4100

Spotify Track Audio Feature Clustering

Abstract

This AI project explores the idea of clustering Spotify tracks on a subset of their audio features, which are computed internally by Spotify and accessed using requests to its public `audio-features` API endpoint. Using this clustering, one might then be able to predict whether they will like a given track based on whether or not its features fall within the bounds of an existing cluster. The DBSCAN (Density-Based Spatial Clustering of Applications with Noise) clustering algorithm was used, with relative success. An attempt was made at choosing the ϵ value that results in the best possible clustering. Possibilities for future improvements are discussed.

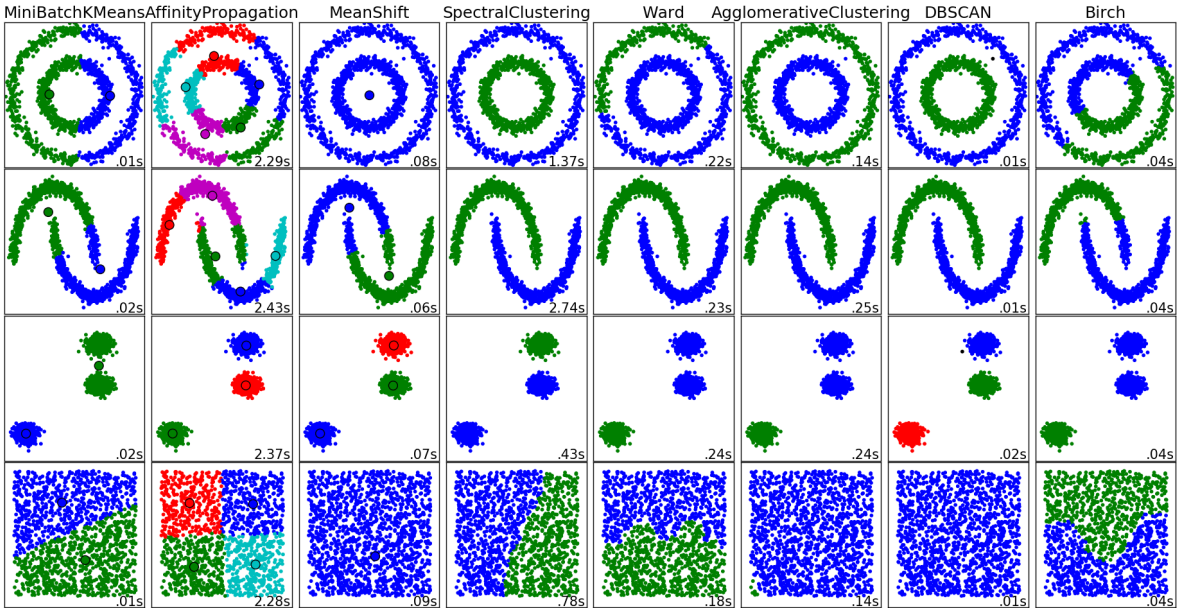
Introduction

Motivation

My main motivation for choosing to explore clustering on Spotify data was that I was already familiar with the Spotify API from a previous hackathon project. I was able to cannibalize some code from that project to make getting started on this project easier, and I was already familiar with the `spotipy` Python library that simplifies authentication and endpoint requests. While working on the hackathon project, I had noticed the `audio-features` endpoint but did not have time to explore it, so I thought this project presented a good opportunity to do that. I thought I might be able to partially reverse engineer Spotify's track recommendation system with the features provided by the endpoint.

Clustering

This project makes use of an artificial intelligence technique called clustering. Clustering algorithms take a list of items and group together the items that share the most similarity.



Examples of different clustering algorithms[1]

In the plots above, each item is a point in 2-dimensional space. They are usually represented using x,y coordinates. In clustering terminology, x and y would be called features.

The subset of audio features I use for clustering Spotify tracks contains 7 features, each of whose values is within the range [0,1]. Here is an example set of features for one track:

```
{
  "danceability": 0.735,
  "energy": 0.578,
  "speechiness": 0.0461,
  "acousticness": 0.514,
  "instrumentalness": 0.0902,
  "liveness": 0.159,
  "valence": 0.624,
}
```

These can be treated as coordinates in 7-dimensional space, and clustered in the same way as the 2-dimensional points (although the result is not nearly as easily visualized).

Approach

I use the DBSCAN algorithm as provided by the `sklearn` library to cluster the track features. DBSCAN finds points that are densely packed and uses them as core that it then builds clusters out from. Points are marked as one of core, reachable, or noise, based on given `minSamples` and

[1] http://scikit-learn.org/stable/modules/clustering.html

ε parameters. Core points form the core of a cluster. A point is marked as core if at least `minSamples` points are within distance ε of it. A point is marked as reachable if it is within distance ε of a core point, but fewer than `minSamples` points are within distance ε of it. A point is marked as noise if it is not core or reachable.

I chose density-based clustering with noise because I am more interested in finding the most densely-grouped clusters of tracks than trying to categorize all tracks into a cluster, which another algorithm like K-Means would do.

The main difficulty I faced in writing the audio feature clusterer was evaluating whether a given clustering was "good". Cluster evaluation is known to be difficult.[2] On the same list of features, DBSCAN produces very different results when the value of ε is changed even slightly. Using one ε value, it may produce one clustering with few, relatively equally sized clusters, but the majority of the items labelled as noise. With another ε value, it will place most of the items into clusters, but contain one cluster that is very large compared to the others. It is hard, and even somewhat subjective, to determine which of these is the best or most accurate clustering.

My solution was to cluster the data repeatedly with ε values in the range [.1,.3] with step size 0.01. I scored each clustering on three metrics:

- percentage of features placed in clusters, not marked as noise (maximize)
- number of clusters (maximize)
- mean difference in size of clusters' cores (minimize)

Then, I picked the clustering whose overall ranking (sum of its three metrics' ranks among all others) was lowest. For the two feature sets I tested my code on, it chose ε values of .11 and .14, so the general best ε value for Spotify audio features is probably somewhere in that ballpark.

Results

On feature set one, my own Spotify library containing 671 tracks with audio features, my code produced 10 clusters with sizes 86, 31, 5, 6, 8, 7, 5, 4, 5, 8; ~75% noise.
On feature set two, my classmate Joe Ruane's library containing 2758 tracks with audio features, my code produced 22 clusters with sizes 5, 5, 673, 30, 6, 4, 8, 163, 5, 6, 5, 8, 5, 7, 11, 9, 6, 7, 5, 5, 5, 5; ~64% noise.
Both of these results contain two relatively large clusters—86 and 31 tracks from my library, and 673 and 163 from Joe's—and many small clusters. I came across a FiveThirtyEight article titled *Spotify Knows Me Better Than Myself*[3] that may explain this phenomenon. The author details his visit to the Spotify office in New York, where he learns about Spotify's efforts to quantify users'

[2] https://en.wikipedia.org/wiki/Cluster_analysis#Evaluation_and_assessment
[3] https://fivethirtyeight.com/features/spotify-knows-me-better-than-i-know-myself/

music tastes using tech from its recently acquired (as of 2014) Echo Nest. The director of the "Nestify" tool tells the author that many listeners have two main clusters in their library. One contains music that they tell other people they listen to—indie, classics, and other genres perceived as "cool". The other, nicknamed the "shame cluster" by the author, contains catchy pop music from the top charts list.

I tested this theory by sampling a few tracks from each of the two largest clusters. My 86-track cluster had a lot of indie music in it. My 31-track cluster was full of quiet songs, mostly piano solos or similar. The hypothesis didn't hold up for my library!

I tried Joe's 673- and 163-track clusters next. The tracks I tried from the large cluster were mostly high-energy rock/electronic music. All of the tracks I tried from the smaller cluster were electronic music, also mostly high-energy.

Neither of these clusterings supported my initial guess that they were composed of "shame" and "indie" clusters, but I was quite surprised by how similar the tracks within each cluster sounded to each other. It seems like Spotify's audio features are a very accurate way of classifying music!

Conclusion

While I expected to see a greater number of roughly equal-size clusters in the output of my program, the clusters it does produce are definitely spot-on for predicting the musical tastes of the user. I tend to listen to a lot of indie and quieter music, and Joe reports that he is a big fan of electronic and "pump-up" music.

Looking ahead, I would like to try tweaking parameters a little bit more to see if the large clusters can be subdivided into smaller, more accurate groups. It would also be interesting to count the number of instances of each genre within a given cluster to see how closely clusters correlate to genres. I would also like to add functionality to create playlists from the clusters, something I did not get to before the project due date.