Spotify Million Playlist Challenge

Brendan Ryan, Christopher Lescinskas,
Matthew Martin

Agenda

- Questions sought to be answered
- Data preparation work
- Tools used
- Classification/clustering applied
- Knowledge gained
- How knowledge can be applied

Questions Sought to be Answered

For this project we wanted to take some of the information learned through the semester and apply it to the public Spotify Million Playlist Challenge. This is a public challenge to take a prompt and extend a playlist. The following are questions that we sought to answer through our work.

- What do the basic heuristics of the dataset we are working with look like?
  - Histogram, Box Plot, Network Graph, etc.
- Can we use the Apriori algorithm to extend a playlist looking only at song titles?
- Can K-Nearest Neighbors be applied to extend a playlist?

Data Preparation Work

Fortunately for us, there was very little needed for data preparation. The data set came as part of a public challenge, much of the preprocessing work had already been done for us. The data had already been anonymized, had offensive content removed, and came in a standardized format.

Data set basic heuristics

Metric	Value
Number of Playlists	1,000,000
Number of Tracks	66,346,428
Number of Unique Tracks	2,262,292
Number of Unique Albums	734,684
Number Of Unique Artists	295,860
Number Of Unique Titles	92,944
Number Of Playlists with Descriptions	18,760
Number Of Unique Normalized Titles	17,381
Avg Playlist Length	66.346

Data set available attributes

Attribute Name	Attribute Type	Value
pid	Ordinal	Integer
name	Nominal	String
description	Nominal	String
num_artists	Numeric – Ratio-scaled	Integer
num_albums	Numeric – Ratio-scaled	Integer
num_tracks	Numeric – Ratio-scaled	Integer
duration_ms	Numeric – Ratio-scaled	Integer
collaborative	Binary	Boolean
track_name	Nominal	String
album_name	Nominal	String
artist_name	Nominal	String
pos	Ordinal	Integer (zero-based)

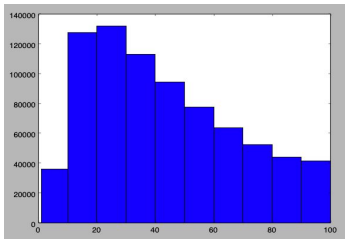
Tools Used

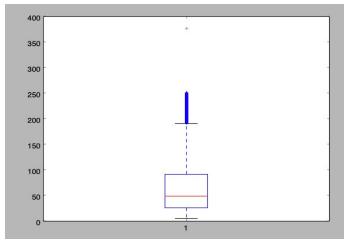
We used a number of different tools for this project.

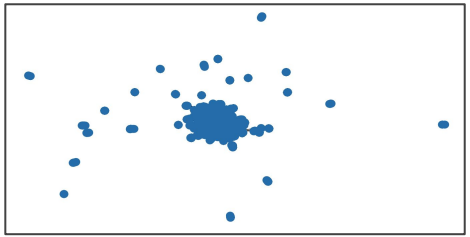
Tool	Purpose
Python	Used for all of our coding. The public challenge came with some pre-existing python routines that we were able to adapt and improve on to run the balance of our analysis Libraries: ● nltk - includes methods for stemming (ex. converts “running” to “run”), tokenizing, Levenshtein distance and Frequency ● efficient_apriori - includes method to determine frequent item sets ● Pylab - for charting
Microsoft Word	Used for all of our reports
Google Slideshow	Used for all of our presentation materials and group collaboration
Github	Used for storage of all of the group artifacts and version control

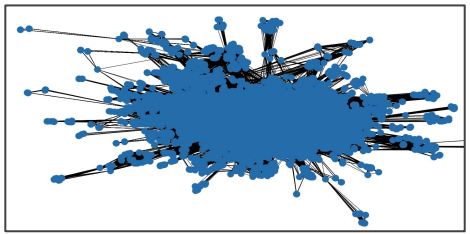
Classification/Clustering applied

We started evaluating the data with basic heuristics. We looked at playlist lengths in both histogram and box plot form. This helped validate that the playlists were sufficiently long to have meaningful connections of songs. We also looked at a network diagram of (a small set of) the data.









Apriori Pseudocode Overview

- Process_playlist_single_seed
  - ‘Main Function’ → incorporate helper functions
    - Open seed CSV and extract seed playlists and playlist IDs
    - For each song in seed playlist:
      - Search 50,000 playlists in dataset for song matches
      - Run apriori market basket analysis on playlist matches
      - Generate output playlists from top candidates
        - No seed or duplicate tracks, 500 tracks total
    - If < 500 songs after going through each seed song:
      - Cut support in half and try again
    - If not enough data in first 50,000 playlists:
      - Move on to next 50,000

Knowledge Gained

Apriori worked the best for us to determine a playlist extension. We were able to use the algorithm to determine an playlist extension with reasonable accuracy. (Refer text file under results GitHub for full output)

Example prompt: `'spotify:track:1zWMf9bVyhY5W3ZORbjNWt',`
`'spotify:track:5tf1VVWniHgryyumXyJM7w', 'spotify:track:27SdWb2rFzO6GWiYDBTD9j',`
`'spotify:track:2aFiaMXmWsM3Vj72F9ksBl', 'spotify:track:4JLojdKkKNM3rgvP2zvUGR'`

Prompt return: `1003478, 'spotify:track:0PDUDa38GO8lMxLCRc4lL1',`
`'spotify:track:40YcuQysJ0KlGQTeGUosTC',`
`'spotify:track:6eT7xZZlB2mwyzJ2sUKG6w''spotify:track:14WWzenpaEgQZlqPq2nk4v',`
`'spotify:track:0wdKiSBUT7aZkXUIdJWcwC', 'spotify:track:0PJIbOdMs3bd5AT8liULMQ'... [and so on]`

Code started with support .1 and then moved to .05 to get to 500 tracks.

Code run on 1000 seed playlists to generate 1000 playlists with 500 new songs (CSV on GitHub). Run time is 10-20 hours

k-Nearest-Neighbor Classifier

At the outset of this project, we identified the k-nearest-neighbor method as a candidate to classify our data objects and accomplish our goal of extending a playlist based on a song title as a prompt. However, after reviewing our data set and the attributes of the KNN method, it became clear to us that some preprocessing would be required. Crucially, the KNN method uses Euclidean distance as a metric to define closeness. As our data consists primarily of categorical attributes as opposed to numerical attributes, we would have to find another approach to solve our problem, since attempts to process the data into a format that would be acceptable for KNN were unsuccessful.

Natural Language Processing

In response to the need for a replacement for KNN, we turned back to our data for inspiration for our next move. Two of the major components of a playlist are the playlist title and the song titles associated with the playlist. NLP came to mind, so we set out to use the playlist title to mine for relationships. Given some prompt, such as “run” (as in a playlist that might be consumed while running for sport) or “rap” (a music genre), can we return a list of tracks frequently associated with this prompt? Noted in our Project Proposal, Ludwig et al. took a similar approach, which we have patterned our approach after.

NLP Results

Using the Natural Language Toolkit (‘nltk’ Python module), we iterate through every playlist title in the data set, looking for playlists that are a “fuzzy match” against a given search term with a Levenshtein distance of 1. Playlist titles were normalized and tokenized in the process. Once candidate playlists and their corresponding songs are discovered, 10 individual songs are recommended based on their frequency of occurrence (top 10 most frequent songs) among the candidate playlists.

As an example, using “rap” as the search term, the algorithm identifies 11,150 candidate playlists consisting of 888,071 songs. Of the 10 most-frequent songs, “HUMBLE.” by Kendrick Lamar is at the top of the list with a frequency of 2,708. All of the top-10 songs returned, upon close inspection, are songs in the “rap” genre, though the algorithm doesn’t claim to return songs that are genre-specific.

How to Apply Knowledge

This knowledge can be applied to several other areas with similar/same problems to be answered.

- **Extend a viewing list for video applications**
  - This is almost the same use case for a different application type. Assumes that knowledge of only the video name and relation to other videos (through playlist or similar) could provide valuable extension information.
- **Recommended shopping items**
  - This type of application could be extended to either (1) offer up other potential items someone may want to purchase or (2) fill out a shopping list for the user saving them time and increasing value proposition of application
- **Recommendation for connections on social media applications**
  - This is likely to be only a part of a social media application implementation. It can help in making connections to other folks who seem to have a similar network. This is probably not the primary value proposition for social media applications however so other implementations and analysis would be needed.