Bianca Trombetta   btrombetta@college
Calvin Tonini   cni@college
Chris Ramirez   cramirez01@college

"Moosify"

Brief Overview

We will be using user generated playlist data to find groupings of songs and use them to create recommendations. As Morrisett said in lecture, there are some things that humans are better at doing than computers - making playlists is one of them. We will look at user generated playlists to create a traversable graph that will give us information as to which songs go together in playlists, based on graph data and centrality algorithms.

Feature List

- Generate a graph[1] data structure[2] procedurally from standard playlist files.
- Generate playlist files from Spotify
- Use graph data structure to find closest path between two nodes
- Use graph data structure to find centrality of a node
- Use graph data structure to find closeness of a node to a set of nodes
- Given a song, gives a suggested playlist by finding songs that are close to the given song
- Given a playlist, suggests a song that is close to those playlists
- Given two sets of songs, suggests a new song similar to both sets
- Given two playlists, gives a playlist that is similar to both original playlists
- Use cluster algorithms[3] to categorize songs into "genres" based on playlist content

Technical Specification

Main Features

Graph Data Structure
- The graph data structure will have nodes as Songs and edges weighted by how often two songs appear together in playlists. This means that the graph data structure is undirected.

---

[1] https://en.wikipedia.org/wiki/Graph_(mathematics)
[2] https://en.wikipedia.org/wiki/Graph_(abstract_data_type)
[3] http://docs.scipy.org/doc/scipy/reference/cluster.html

Spotify Playlist Crawler

- This crawler program traverses different Spotify users using the Spotify API[4] and downloads a user's public playlists into lists of songs before moving on to other users, similar in concept to the crawler used in the Moogle problem set. We may implement the ability to export these lists of songs in a standard file format[5].
- As an interface, the crawler will intake a user identifier, an integer representing the total number of playlists to crawl, and information specific to connecting the Spotify API such as rate limits or OAuth tokens, and output a list of playlists.
- Behind the interface, there will probably be a recursive function that remembers the set of users and Playlists already traversed and saved and remembering future possible playlists and users to connect to, while collecting a list of playlists, similar to the Moogle problem set.

Graph Maker

- Using a list of playlists, this function will procedurally generate a graph data structure, the explanation of which is above. For each playlist, the function will ensure that a node exists for each song on that playlist in the graph data structure, creating a new node, if necessary. Then, the function will create or modify the weighted edges between each combination of song nodes in the playlist accordingly.
- Will traverse the playlist data we have to make a graph where every song becomes a node, where edges are other songs included on the same playlists as each node.

Shortest Path Finder 🗨

- Will use Floyd-Warshall's algorithm[6] to find the least distance between every pair of nodes. This will be useful for finding songs that match two people's tastes. This is also essential to calculating centrality and/or closeness.
- This function will take in a graph data structure and output another graph data structure with nodes having an extra field on average and median distances calculated from that node. We project needing more research on this issue.

---

[4] https://developer.spotify.com/web-api/
[5] https://en.wikipedia.org/wiki/Playlist#Types_of_playlist_files
[6] http://www.algorithmist.com/index.php/Floyd-Warshall%27s_Algorithm

Centrality Closeness Calculator/Assigner

- Will examine the graph data structure to assign values to each node. This examination will take the form of running our data structure through Johnson's algorithm[7] or the Floyd-Warshall algorithm. If there is enough time, we may implement both to see which works better.
- This will take a graph data structure and return another graph data structure where every node has an added field that contains a ranking of its centrality.

(Like a) Good Neighbor (State Farm is There)[8] Finder

- Will take a node and compare every edge value it has against each other. It will return the top x (nodes) songs with the highest weighted edge value. This will be useful for making a playlist out of a single song.
- The interface is a song title, which is matched to a node, and output a playlist. Under the hood, the function takes the given node and analyzes it's edges before returning songs. If we have time, a more complicated version may traverse to other songs and look for triangles.

Extra Features

Cluster Decider 💬

- Using a provided Python library, we will examine our graph data to make "genres". These genres will hopefully map on to types of playlists that people make, either reflecting music taste, moods created by playlists, or both.
  - http://docs.scipy.org/doc/scipy/reference/cluster.html

Next Steps

We plan on utilizing the Spotify API to access public playlist data. By focusing first on implementing our Playlist Crawler, we will be able to build up our playlist database, which is crucial for the functionality of our Graph Maker and additional features. Our goal is to generate a database of at least 10,000 playlists, by executing our crawler over several days.

Since the Spotify API returns objects in JSON format we intend on working with Python, which contains dictionaries that are easily suited to our needs. As Python is an unfamiliar language to everyone in the group, we're considering using Codecademy to get a crash-course on the syntax. Our environment set-up will involve downloading a

---

[7] http://www.geeksforgeeks.org/johnsons-algorithm/
[8] https://www.youtube.com/watch?v=OB6r2Wi0E98

Python debugger, compiler, and a text-editor with Python-friendly extensions and integrating with a GitHub repository that will be created soon.

Preliminary Timeline

4/12: Familiarize ourselves with Python
       Set-up the environment and Github
4/14: Implement Playlist Crawler
       Begin building up database
4/15: Begin working on implementing Graph Maker (core feature)
4/16: Reach goal of 10,000 playlist database
4/20: Graph Maker implemented
       Begin working on implementing extensions (Centrality and Neighbor Finder)