CS6220 PROJECT REPORT

SPOTIFY SONG RECOMMENDATION SYSTEM

Madhuri Palanivelu - NUID: 001059643

INTRODUCTION

Data Science has become one of the most popular and necessary branches of computer science in recent years due to the increasing demand and amount of data being generated across the world. It is a domain that specializes specifically in capturing, maintaining, processing, analyzing and finally, communicating all kinds of data. This project will be making sure these steps get completed in order to obtain the data required. There are tons of different areas of data science and machine learning that can be explored and the area that I've chosen to learn more about are ***Recommender Systems.***

They are an extremely important application of data science and machine learning, especially in recent years. These systems are being made and perfected for applications like Netflix, LinkedIn, YouTube, Amazon, etc. for recommending movies or shows, people to connect with, items to buy and so on, based on the user's individual taste.

With the advent of the Internet, people have access to so much more data and more options that it becomes harder for them to choose what they'd like and make a decision. This is a different case if they have a smaller set of options. That's how Recommender Systems become useful, by suggesting the top few, best items they might like . This project will concentrate on a problem identified, which is solvable with a ***Cluster-based Recommender System.***

PROBLEM IDENTIFIED

Spotify is a popular music streaming app and media service provider used by millions of subscribers, all across the globe. ***Discover Weekly*** is a unique playlist created for each Spotify user every Monday with 30 songs, picked out or recommended to the user based on their previous listening habits.

Initially, the problem that I identified was regarding the fact that Discover Weekly was slightly inaccurate for my personal tastes and I wanted to improve the results and create a model that worked better. For this I had to create a dataset of songs I liked and disliked, and make this a classification problem. But

I found this to be a complicated endeavor because in order to make this a user-facing application, I'd need to know the good and bad songs for any user who uses the application.

So my updated problem statement is:

To create an application that allows the user to input a song and retrieve a recommended list of songs, similar to it.

This way, the user can get a wider variety of songs similar to a song they like, which is comparable to the purpose Discover Weekly. The solution to this problem, as discussed below, will also eliminate the problem of having to know all users' current taste in music.

SOLUTION IMPLEMENTED

The solution I proposed for the problem statement identified above, encompasses the following steps:

1. Build the song dataset with Spotify API
2. Apply K-Means clustering algorithm with Principal Component Analysis on the dataset
3. Use the K-Means model created to predict which cluster, the user's searched song belongs to
4. Create Flask application to perform the above functionalities and display a randomized set of 30 songs from the same cluster to the user
5. Dockerize the Flask application

Each section is discussed in detail in the following sections.

DATASET CREATION

The dataset for this application would require a pretty large set of songs in order to provide a lot of variability to the user. The Spotify API for Developers had a client API called Spotipy, used by Python developers. With Spotipy, I was able to perform the following to create the full dataset:

1. Authenticate my personal Spotify account in order to gain access to the Spotify library
2. Collect all the categories'/genres' playlists available on Spotify
3. Retrieve all the tracks of each playlist

4. Extract each tracks' audio features
5. Create a Pandas Dataframe to hold each tracks' Spotify ID, Title, Artist and all of their audio features
6. Bundle the newly created DataFrame into a pickle object, for ease of use while creating the model in the later steps. This will prevent having to create the dataset from scratch multiple times, as this is a time consuming process

The whole dataset created contains 112,566 songs each with 11 audio features having different values:

- Energy
- Danceability
- Acousticness
- Instrumentalness
- Liviliness
- Mode
- Key
- Tempo
- Valence
- Loudness
- Speechiness

id	name	artist	energy	danceability	acousticness	instrumentalness	mode	liveness	key	tempo	valence	loudness	spe
5mnvqisoDJilY0uCEdT8rG	Danny's Song	Loggins & Messina	0.198	0.507	0.80600	0.000000	1	0.1020	2	141.261	0.601	-16.089	
5sY2beqWPiCRmTyjEkRPPZ	To Find A Friend	Tom Petty	0.673	0.542	0.02140	0.000444	1	0.2810	5	88.039	0.505	-8.080	
2Y90nL1ohB4sgYELDs7uNx	Glory Days	Bruce Springsteen	0.960	0.574	0.04570	0.000000	1	0.1210	9	117.486	0.978	-4.906	
0sDqo9UPzPUtu9wEkl3zRB	The Weight - Remastered	The Band	0.519	0.630	0.22500	0.000004	1	0.0974	9	143.942	0.502	-10.997	
24NwBd5vZ2CK8VOQVnqdxr	Sweet Emotion	Aerosmith	0.760	0.380	0.00298	0.029400	1	0.1040	9	99.437	0.491	-10.961	

Dataset with ID, Title, Artist and some Audio Features

MODEL CREATION

Classification vs. Clustering Decision:

In the first, discarded problem statement, the application would be able to recommend songs by classifying them as good or bad. This is a classification

problem that represents ***Supervised Learning***, where the input data would be labelled with predefined classes and the probable output would be known.

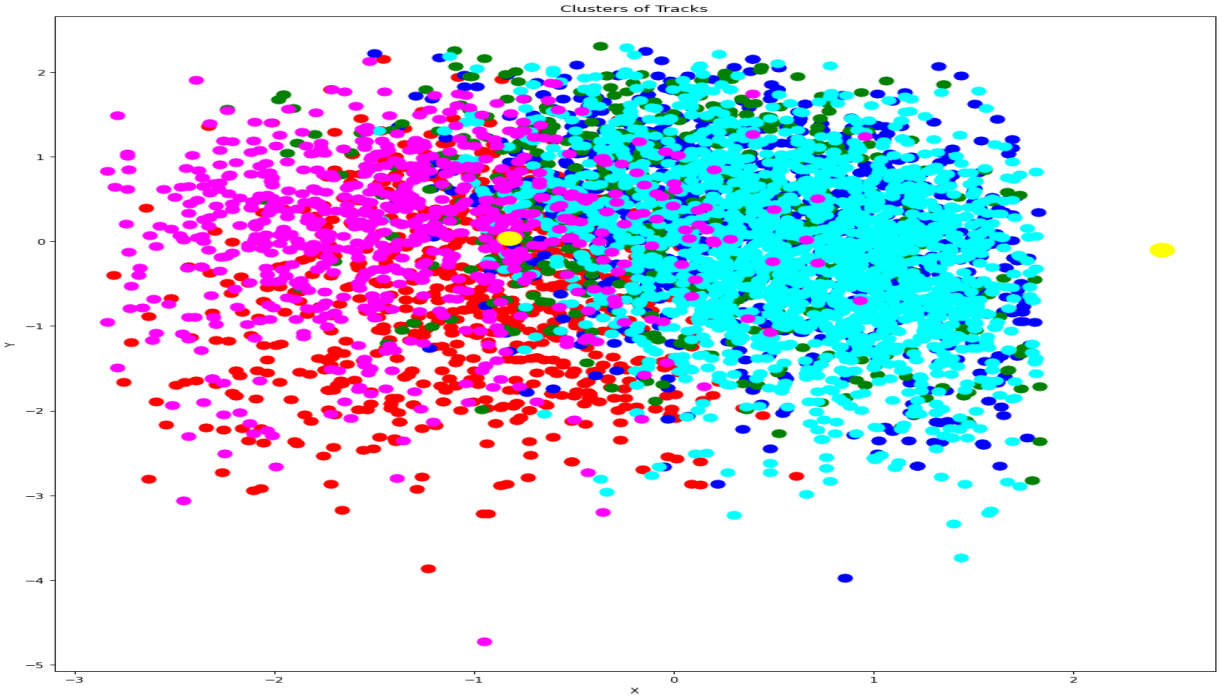
In the case of the final problem statement, songs would be grouped into clusters based on their similarities based on audio features like the acoustics, danceability, etc. Here the labels of the input data aren't known beforehand and the nature of the output isn't known either. This is a clustering problem that represents ***Unsupervised Learning***.

In light of the comparisons made between these two types of models, I chose to use K-Means clustering.

Clustering Approach:

K-Means Clustering allows the data to be split into 'k' clusters or groups in this case. I chose K-Means over Hierarchical Clustering because with Hierarchical Clustering, we're building a dendogram where smaller clusters keep combining with each other until they form into one large cluster. Based on the nature of the results I'm aiming to achieve, K-Means clustering was the right choice as I want to split my data into K groups that have data similar to each other, and display one group fitting to the user's song query.

I used the Python sklearn package in order to implement the K-Means model. Initially I worked with just 5500 tracks and not the 112,566 tracks I was able to retrieve later. Initially I used around 5 clusters to get an idea of how the clusters were getting formed but this was how the plot with all the clusters looked like:



Failed K-Means Clustering Plot between Energy and Danceability

This plot was formed between Energy and Danceability, for example. As one can easily see, the clusters overlapped heavily, the cluster centers (yellow) weren't visibly seen correctly and I was unable to gauge any information from this K-Means model. This happened between all pairs of features.

It seemed that possibly ***reducing the dimensionality*** of the dataset would help organize the clusters a little more.

On looking at the correlation between the 11 features of my dataset, this is how it looked:

	energy	danceability	acousticness	instrumentalness	mode	liveness	key	tempo	valence	loudness	speechiness
energy	1.000000	0.311106	-0.791947	-0.447718	-0.067689	0.228919	0.039333	0.250418	0.424055	0.803880	0.138610
danceability	0.311106	1.000000	-0.330823	-0.340426	-0.080429	-0.073679	0.026816	0.007112	0.547238	0.443800	0.197797
acousticness	-0.791947	-0.330823	1.000000	0.419751	0.065747	-0.167539	-0.033514	-0.211367	-0.307314	-0.678517	-0.136080
instrumentalness	-0.447718	-0.340426	0.419751	1.000000	-0.021086	-0.084887	-0.023469	-0.140500	-0.383642	-0.627077	-0.125773
mode	-0.067689	-0.080429	0.065747	-0.021086	1.000000	-0.013341	-0.143667	0.004327	-0.017941	-0.038071	-0.088691
liveness	0.228919	-0.073679	-0.167539	-0.084887	-0.013341	1.000000	0.002180	0.037933	0.021407	0.135344	0.088395
key	0.039333	0.026816	-0.033514	-0.023469	-0.143667	0.002180	1.000000	0.001123	0.030481	0.033246	0.018973
tempo	0.250418	0.007112	-0.211367	-0.140500	0.004327	0.037933	0.001123	1.000000	0.126131	0.235523	0.074261
valence	0.424055	0.547238	-0.307314	-0.383642	-0.017941	0.021407	0.030481	0.126131	1.000000	0.417432	0.093272
loudness	0.803880	0.443800	-0.678517	-0.627077	-0.038071	0.135344	0.033246	0.235523	0.417432	1.000000	0.117156
speechiness	0.138610	0.197797	-0.136080	-0.125773	-0.088691	0.088395	0.018973	0.074261	0.093272	0.117156	1.000000

Correlation Map between Dataset Features

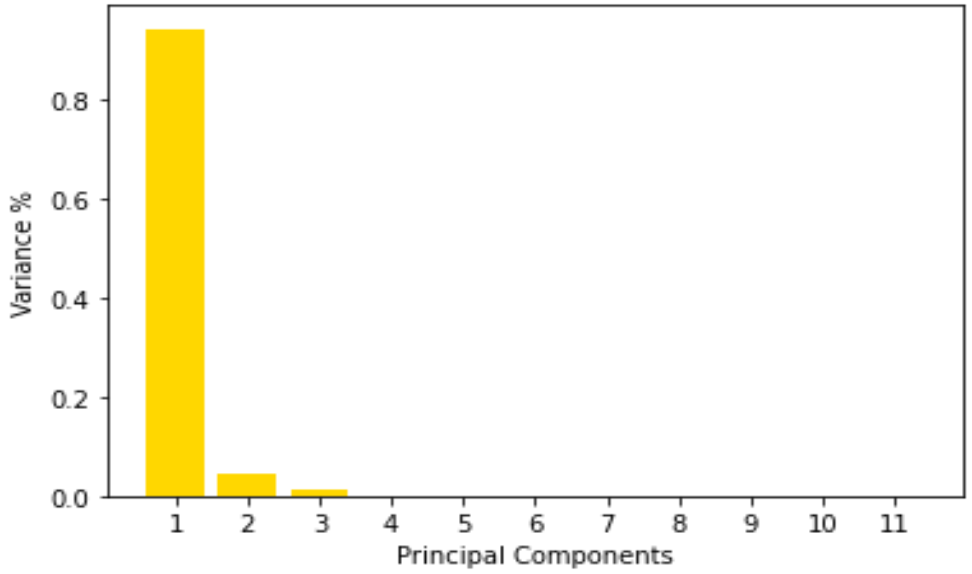
The redder the boxes, the more correlated its corresponding features are ie., they have a lot of repeated and similar data amongst them. It can be seen from this ***correlation map*** that quite a few features have significant correlation like between loudness and energy (80%) and between valence and danceability (54%). Even if those features were removed, there would still be close to 10 more features left to visualize with K-Means each time, till we found an optimal result.

Dimensionality Reduction:

With this information, I decided to use ***Principal Component Analysis (PCA)*** for dimensionality reduction of my dataset. The goal was to project this large number of features into a lower dimensional space and reduce them into the most important components. This means that instead of removing features as a whole, these components will compact important information ie., the variance,

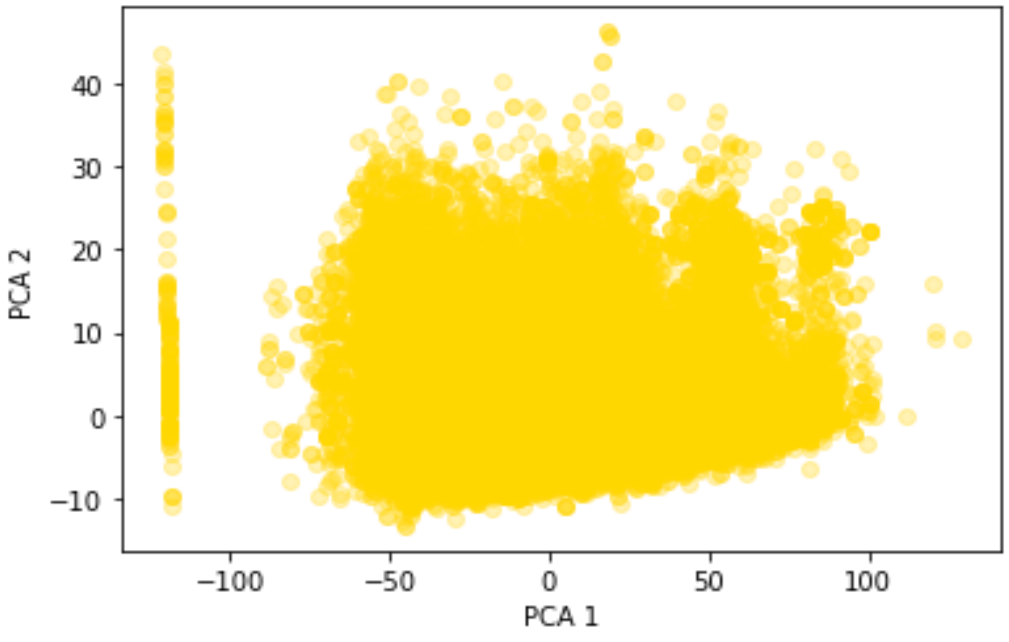
from all the features into a few components. The first few components will have the most variance.

In the following plot, PCA was applied to my dataset and one can see how the variance is highest in the first component and how it decreases in the next components. For this particular dataset, the data was encompassed in the first 3 components itself, in comparison to the next 8 components. For this reason, I chose to use the ***first two components*** for my model:



PCA Components vs Variance Levels Plot

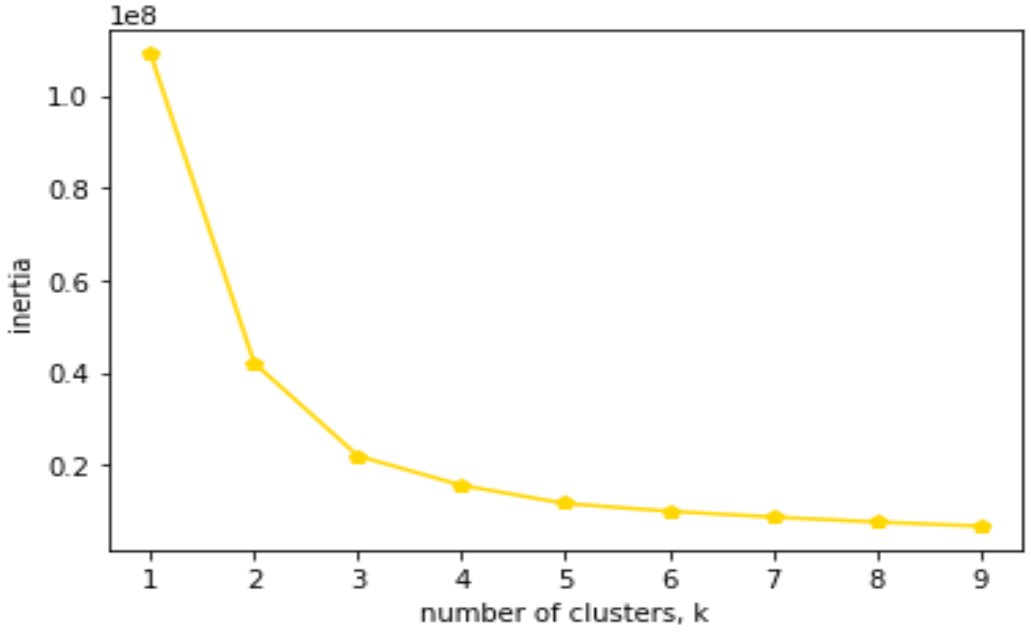
Now while plotting the first two components with the data they hold, the plot looked like this:



Variance Spread between first two PCA Components

This spread seems meaningless because of the density, but then I was able to perform K-Means clustering on this data and get more valuable clustered data.

Initially, I kept very few clusters for my data and this proved to be useless for the nature of the application. Many different songs that had less similarity, were put together in a single cluster. Even while evaluating the optimal amount of clusters through the approximations of the ***Elbow Method***, I found that three clusters seemed optimal:

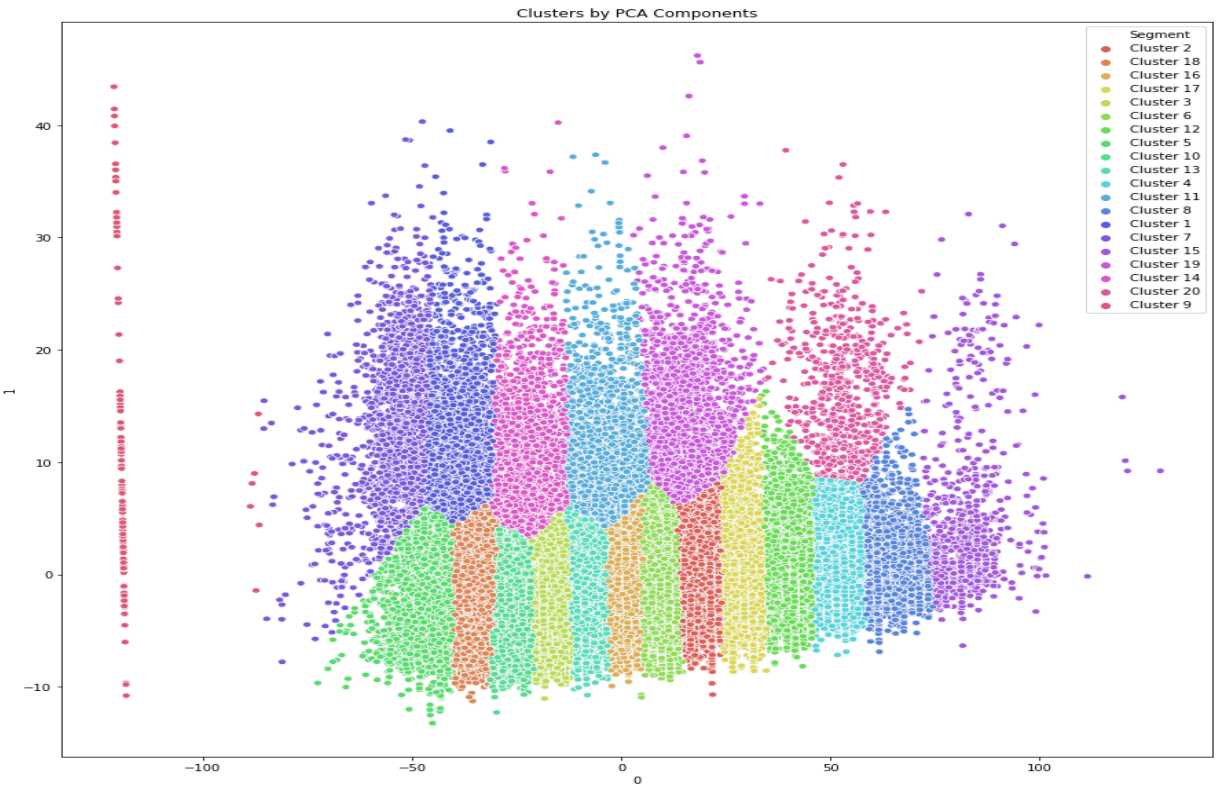


Elbow Graph based on Inertia

The Elbow Method is particularly useful if and only if, we don't know the number of clusters to use for our model. I had a fair idea that my model needed many clusters to present the data more accurately for any song query, so I decided to amplify the number five-fold and use ***twenty clusters*** instead. This helped get more similar songs for the song query than while using very few clusters.

Though there are fewer features than required for dimensionality reduction, (there are just 11 features) I was able to get a ***more cleaner, organized and well-clustered*** plot after mapping the two most important variance components from my dataset. Performing analysis on individual songs and their clusters, was much easier after using PCA.

As it is visible in the following plot between the two PCA components, the clusters are much more well-defined and easier to make out than the plot seen earlier. The clustering performance seemed to be more successful in this trial:



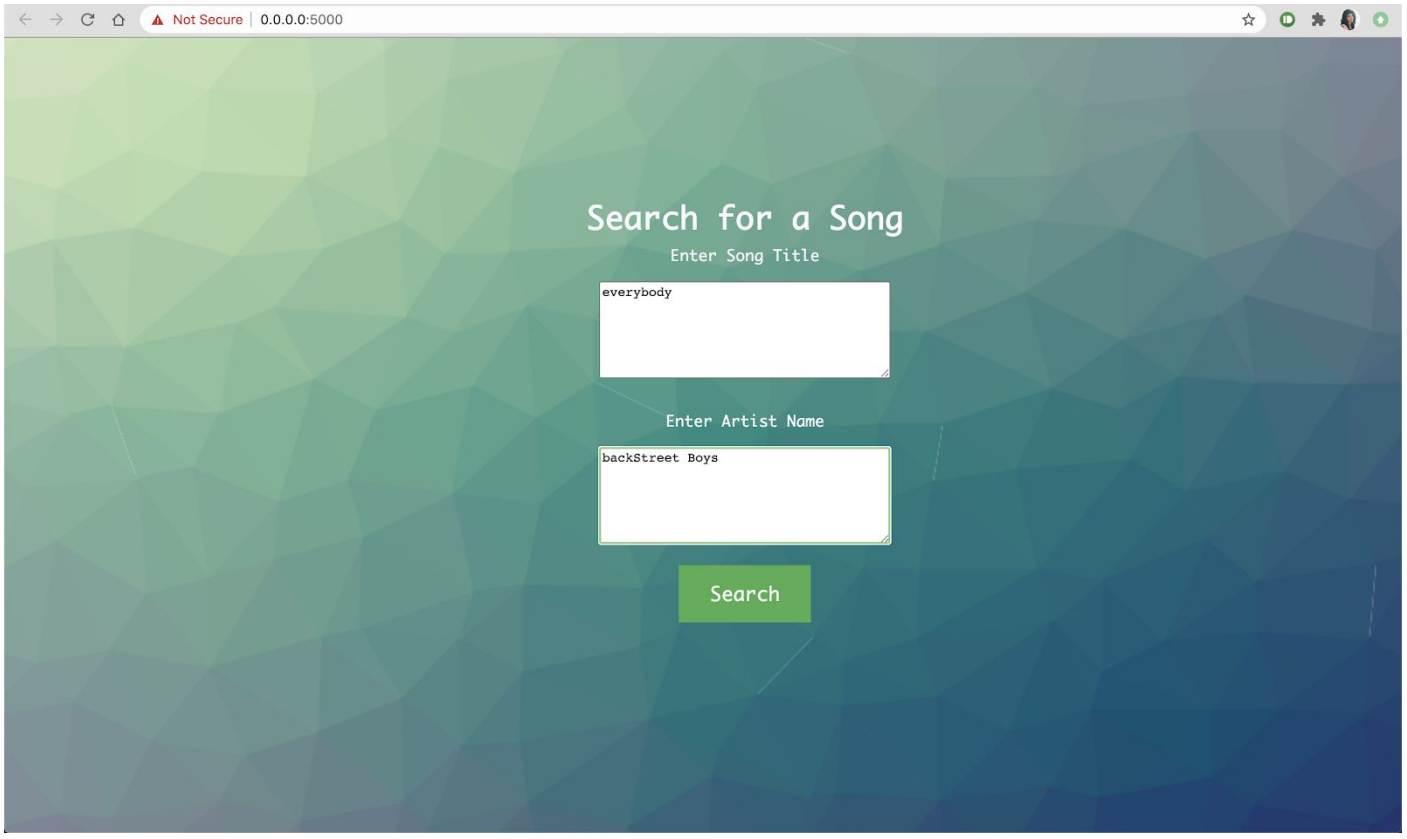
Successful K-Means with PCA Clustering Plot on Full Dataset, between two PCA components

USER INTERFACE, DOCKERIZATION AND RESULTS:

After the model was optimized, it was ready to be made into a service application that users could interact with. I decided to create a ***User Interface with Flask*** where the user could enter the song title and artist name as their query, in one webpage. They would be able to view a randomized set of 30 songs from the cluster their song belonged to in the second webpage. HTML, CSS and Javascript were used to create a more visually pleasing and functionality laden experience for the user.

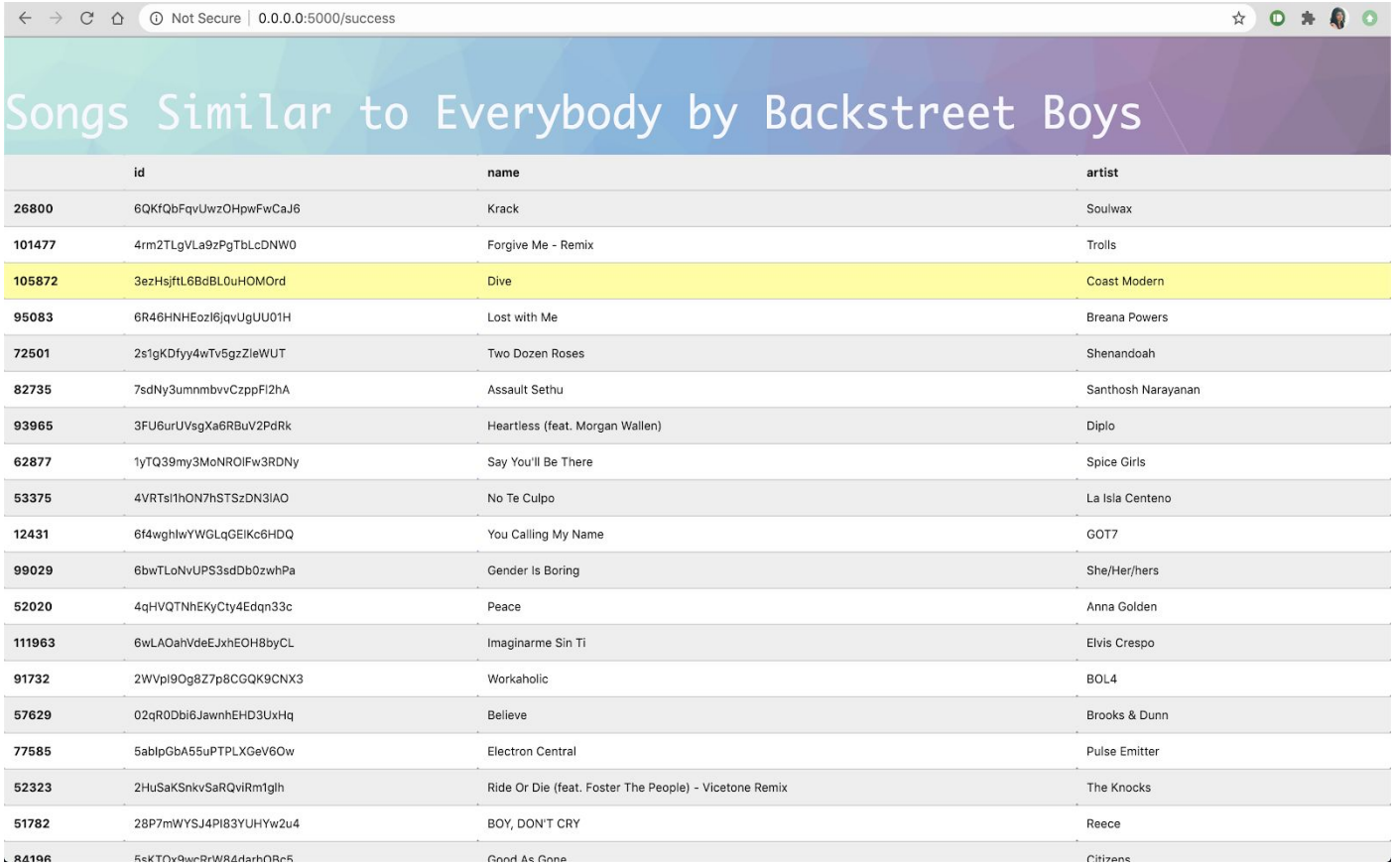
After creating the Flask application, I used ***Docker*** to containerize it. This meant that anyone can use the containerized applications easily even if they don't have the necessary software required to create the application from scratch or run it locally in their machines. Managing and deploying the application in the future would be easier too, if it is Dockerized. For instance, the user wouldn't need any of the Python packages or environments used in the development phase, to use the application.

They also won't need their own Spotify account as it has already authenticated with my personal Spotify account and it doesn't use any of my data in particular. The application only uses the common playlist and track content present in all Spotify accounts.



Input Webpage for Song Title and Artist Name

It can be seen from this example of using the song “Everybody” by the “Backstreet Boys” that this application is user friendly by being ***case insensitive***, allowing the user more flexibility in their search.



	id	name	artist
26800	6QKfQbFqvUwzOHpwFwCaJ6	Krack	Soulwax
101477	4rm2TLgVLa9zPgTbLcDNW0	Forgive Me - Remix	Trolls
105872	3ezHsjftL6BdBL0uHOMOrd	Dive	Coast Modern
95083	6R46HNHEozI6jqvUgUU01H	Lost with Me	Breana Powers
72501	2s1gKDfyy4wTv5gzZleWUT	Two Dozen Roses	Shenandoah
82735	7sdNy3umnmbvvCzppFl2hA	Assault Sethu	Santhosh Narayanan
93965	3FU6urUVsgXa6RBuV2PdRk	Heartless (feat. Morgan Wallen)	Diplo
62877	1yTQ39my3MoNROlFw3RDNy	Say You'll Be There	Spice Girls
53375	4VRTsl1hON7hSTSzDN3lAO	No Te Culpo	La Isla Centeno
12431	6f4wghlwYWGLqGEIKc6HDQ	You Calling My Name	GOT7
99029	6bwTLoNvUPS3sdDb0zwhPa	Gender Is Boring	She/Her/hers
52020	4qHVQTNhEKyCty4Edqn33c	Peace	Anna Golden
111963	6wLAOahVdeEJxhEOH8byCL	Imaginarme Sin Ti	Elvis Crespo
91732	2WVpI9Og8Z7p8CGQK9CNX3	Workaholic	BOL4
57629	02qR0Dbi6JawnhEHD3UxHq	Believe	Brooks & Dunn
77585	5ablpGbA55uPTPLXGeV6Ow	Electron Central	Pulse Emitter
52323	2HuSaKSnkvSaRQviRm1glh	Ride Or Die (feat. Foster The People) - Vicetone Remix	The Knocks
51782	28P7mWYSJ4PI83YUHYw2u4	BOY, DON'T CRY	Reece
84196	5sKTOx9wcRrW84darbQRc5	Good As Gone	Citizens

Search Results Webpage with similar songs to the Song Query

CONCLUSION

By looking at the type of tracks returned as a result of the song query, the interesting finding is that the songs don't necessarily need to be from the ***same genre or even language***. It should be noted that the problem statement from the Problem Identified section, mentions *"songs similar to it"*. Sometimes we "cluster" our tastes in music to a certain genre or couple of genres and we won't try out other kinds of music. The fascinating part of the audio features for each track and the resulting clusters, is that we can observe how close in similarity some songs of different genres really are and how we might appreciate them too. This marks the end goal of this application where it introduces ***songs of different genres and languages to the user, while having a close similarity to each other in terms of audio features.***

This entire application finally proved to be a Cluster-based Recommendation System, where the recommendations were being made to the user, in clusters.

This project was an excellent way to apply the various concepts learned in the classroom and it provided a lot of clarity. It helped improve our backend, frontend and deployment capabilities which collectively rewarded me with some interesting experience for the industry.

REFERENCES

These are several of the websites referred to while developing the application and doing research:

1. https://spotipy.readthedocs.io/en/2.12.0/#api-reference
2. https://stackoverflow.com/questions/36195457/python-sklearn-kmeans-how-to-get-the-samples-points-in-each-clusters
3. https://andrewmourcos.github.io/blog/2019/06/06/PCA.html
4. https://medium.com/@dmitriy.kavyazin/principal-component-analysis-and-k-means-clustering-to-visualize-a-high-dimensional-dataset-577b2a7a5fe2
5. https://365datascience.com/pca-k-means/
6. https://towardsdatascience.com/build-your-own-clustering-based-recommendation-engine-in-15-minutes-bdddd591d394
7. https://www.geeksforgeeks.org/elbow-method-for-optimal-value-of-k-in-kmeans/