Song Recommendation System

Nithin Raj
IMT2017511
nithin.raj@iiitb.org

R Arvind
IMT2017519
r.arvind@iiitb.org

Ronak Doshi
IMT2017523
ronakvipul.doshi@iiitb.org

***Abstract*—The key to a recommendation system is the prediction of users' preferences. Personalized recommendation depends on capturing the current trend of songs of the listener as well as the change in the "type" of songs. In this paper, we propose a novel personalized next-song recommendation system. To capture the song features, we used Spotify's Web API. Using these features, we apply clustering techniques to determine related songs. The cluster which the input song belongs to is determined. This is known as content-based clustering. We then find the songs which are "close" to the input song, by computing the similarity measure, and based on popularity measures of this list of songs, we determine the set of songs which can be played in sequence.**

***Index Terms*—Song Recommendation, Clustering, Similarity**

I. Introduction

The number of songs available exceeds the listening capacity of an individual in their lifetime. It is tedious for an individual to sometimes choose from millions of songs and there is also a good chance of missing out on songs which could have been the apt for the occasion.
These days all effort and research is going in predicting the kind of songs that a particular user will like, using their entire song history. Here, people are trying to recommend songs without taking in mood as a factor. A user may like party songs as well as soothing songs, but recommending a song in any one of these categories requires not just the history of the songs the user like but also capturing the current mood of the user. If a person is in the mood to listen to a party song, he wouldn't want the app to recommend a slow song, which statistically, according to his history is apt but just not suitable for the current mood.

So, we can solve this problem by asking one particular question; given a song the user is currently listening to, which song will the user like to listen to next?
This is the question we are trying to answer in this project. We are trying to find similarity in the song being played by the user with the list of songs available in our dataset by using clustering techniques. After generating a playlist of songs similar to the current song, we use popularity based recommendation system to recommend the next song to the user.

II. Dataset

Spotify is one of the most widely used audio streaming platform. The Spotify database has a wide range of songs and is made available to developers through their Web API. The API provides access to user related data, like playlists,tracks that the user saves and also provides metadata about music artists, albums, tracks directly from the Spotify Data Catalogue.

We scraped a total of around 10,000 unique tracks from the spotify API and all the necessary metadata related to the track. The meta data contains variety of information regarding the track and the artist. For the tracks present in the catalogue, Spotify also provides acoustic features associated with the tracks like their danceability, valence, loudness etc.

We constructed the dataset using the data we fetched from the spotify API and used it to train our models. Let us look at the audio features provided by spotify in more detail:

- **Track Name** : Name of the Track
- **Artist Name** : Name of the Artist/Singer
- **Duration** : The duration of track in milliseconds
- **Preview** : Link to 30 sec preview of the track
- **Key** : Key represents the tonal center of a song. For example, C = 0, C# = 1 and so on.
- **Popularity** : The popularity of the track. The value will be between 0 and 100, with 100 being the most popular.
- **Time Signature** : An estimated overall time signature of a track. The time signature is a notational convention to specify how many beats are in each measure.
- **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.
- **Danceability** : 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** : Energy is 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. For example, metal and rock will have high energy.
- **Instrumentalness** : Predicts whether a track contains no vocals. 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 recording. Higher liveness values represent an increased probability that the track was performed live.
- **Loudness** : The overall loudness of a track in decibels (dB). Loudness is the quality of a sound that is the primary psychological correlate of physical strength.
- **Speechiness** : Speechiness detects the presence of spoken words in a track. The more exclusively speech-like the recording the closer it is to 1.0

- **Valence** : A measure from 0.0 to 1.0 describing the musical positiveness conveyed by a track. Tracks with high valence sound more positive.
- **Tempo** : The overall estimated tempo of a track in beats per minute (BPM).

Track name and artist name serve as label information for the unclustered datapoints. Other than these two, the columns can be classified into **Categorical** and **Non-Categorical** data.

III. Data Visualization

We first tried to find the relationship between the non-categorical columns of the dataset. We can do this by finding correlation between non-categorical data.

	danceability	energy	loudness	speechiness	acousticness	instrumentalness	liveness	valence	tempo	duration_ms
danceability	1	0.32022	0.467422	0.208767	-0.358697	-0.329161	-0.1258	0.499286	0.0180233	-0.0605141
energy	0.32022	1	0.785207	0.176132	-0.806444	-0.398532	0.225028	0.408443	0.273369	0.0506999
loudness	0.467422	0.785207	1	0.153372	-0.678513	-0.602274	0.0900085	0.40336	0.252095	-0.00482126
speechiness	0.208767	0.176132	0.153372	1	-0.194587	-0.132311	0.108006	0.0996715	0.090854	-0.0647668
acousticness	-0.358697	-0.806444	-0.678513	-0.194587	1	0.39148	-0.132127	-0.323875	-0.224542	-0.0811833
instrumentalness	-0.329161	-0.398532	-0.602274	-0.132311	0.39148	1	-0.0227027	-0.393224	-0.140401	0.107339
liveness	-0.1258	0.225028	0.0900085	0.108006	-0.132127	-0.0227027	1	-0.0141007	0.0358309	0.034142
valence	0.499286	0.408443	0.40336	0.0996715	-0.323875	-0.393224	-0.0141007	1	0.143214	-0.128314
tempo	0.0180233	0.273369	0.252095	0.090854	-0.224542	-0.140401	0.0358309	0.143214	1	-0.0134977
duration_ms	-0.0605141	0.0506999	-0.00482126	-0.0647668	-0.0811833	0.107339	0.034142	-0.128314	-0.0134977	1

Fig. 1. Correlation matrix of non-categorical features

We can see that there is high correlation between a few features, such as energy and loudness, valence and danceability, and also danceability and loudness. This shows that we need to reduce the dimensionality of our data.

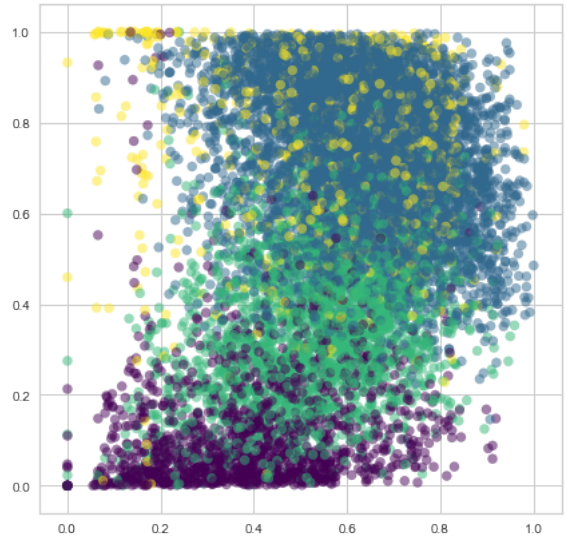

Fig. 2. 2-dimensional visualisation of non-categorical data

We used **Min-Max Scaling** to normalize the data. to see if cluster formation is possible, we tried making 4 clusters out of the scaled data using **K-Means Clustering**. Although the clustering doesn't look neat, we can see some clear distinction between the clusters.

IV. Feature Engineering

Clustering algorithms don't work well with categorical data. Hence, we decided to remove these features from the dataset. As discussed earlier, we found out high correlation between a few features. To combat this, we applied **Principal Component Analysis(PCA)** with a variance preservation of **95%**. PCA helped reduce the number of features from 10 to 6.

PCA serves a dual purpose in clustering problems. As we saw earlier, the clusters formed from the normalised data weren't very distinct. Since clustering algorithms such as K-means operate only on distances, the right distance metric to use is the distance metric which is preserved by the dimensionality reduction. This way, the dimensionality reduction step can be seen as a computational shortcut to cluster the data in a lower dimensional space. Thus, PCA serves as a noise reduction technique, enabling us to make distinct clusters.

V. Model Building

Silhouette analysis can be used to study the separation distance between the resulting clusters. The silhouette plot displays a measure of how close each point in one cluster is to points in the neighboring clusters and thus provides a way to assess parameters like number of clusters visually. Silhouette coefficients near +1 indicate that the sample is far away from the neighboring clusters. A value of 0 indicates that the sample is on or very close to the decision boundary between two neighboring clusters and negative values indicate that those samples might have been assigned to the wrong cluster.

To determine the optimal value of k for the K-Means algorithm, we use the elbow method on silhouette score analysis. If the line chart resembles an arm, then the "elbow" (the point of inflection on the curve) is a good indication that the underlying model fits best at that point. In the visualizer "elbow" will be annotated with a dashed line. The silhouette score calculates the mean Silhouette Coefficient of all samples.

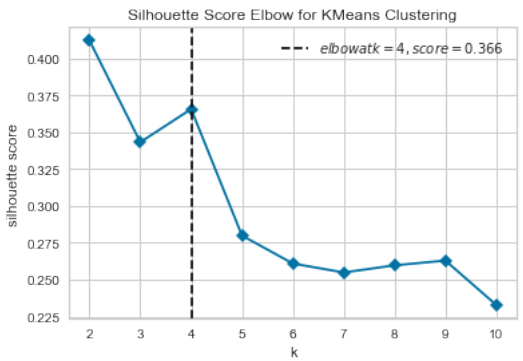

Fig. 3. Mean Silhouette Coefficient vs k line chart

From the graph, we can see that a value of **k = 4** best suits the K-Means Clustering algorithm.

So now we apply K-Means Clustering algorithm with K = 4 and generate our model.

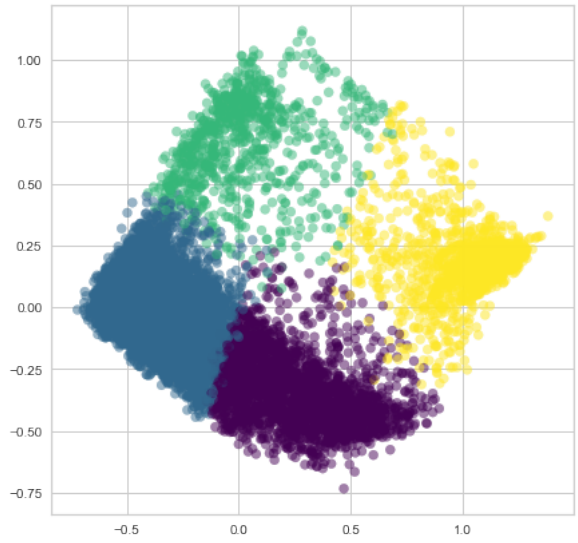

Fig. 4. 2-dimensional visualisation of the 4 generated clusters

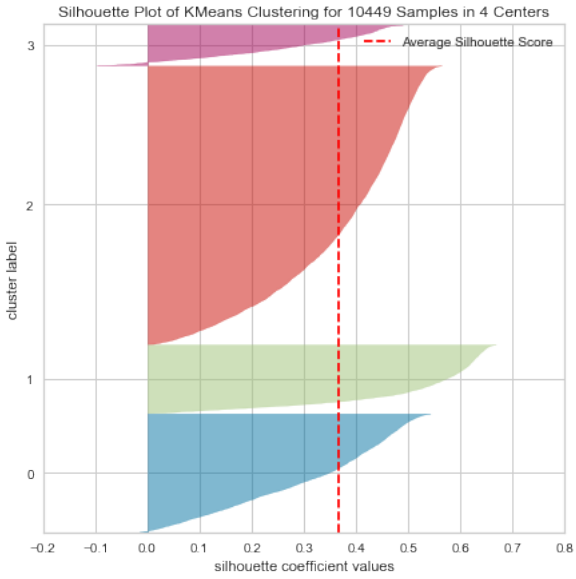

Fig. 5. Silhouette Coefficient per cluster

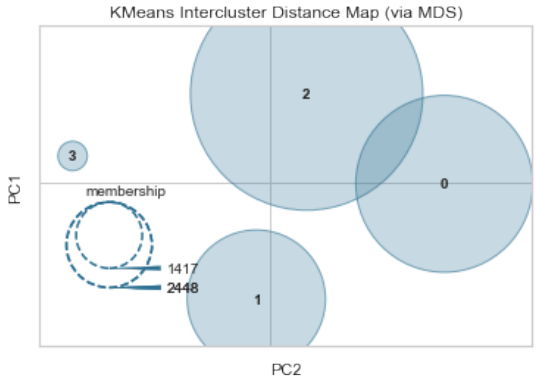

Fig. 6. Intercluster Distance Map for 4 clusters

To create our similarity model, we extract the label of the current song being played and generate a playlist of songs, by collecting all the songs present in the same cluster as that of the current song (by taking songs having same label as of the current song). After extracting the playlist we sort the playlist according to a particular value named as popularityDistance, for each song in the playlist. This particular value is computed by:

$$pD = \frac{Popularity}{DistanceOfSongFromCurrentSongs}$$

Where:
pD: popularityDistance
Popularity: Value provided in the dataset for each song
Distance Of Song From Current Song: Euclideon distance of the song from the currenct song

After sorting the playlist in descending order of popularity-Distance value, we recommend song linearly from the playlist.

VI. Model Evaluation

To evaluate our model, we use two metrics, namely:

- Silhouette Visualizer
- Intercluster Distance Maps

From the plot, we can see that all the clusters have above average silhouette scores. Even though the width of cluster 2 is big, thus making the clusters unevenly sized, k=4 is a good fit, because k=5 showed a poor silhouette score as seen from the elbow plot in Fig. 3.

Intercluster distance maps display an embedding of the cluster centers in 2 dimensions with the distance to other centers preserved. E.g. the closer to centers are in the visualization, the closer they are in the original feature space. The clusters are sized according to a scoring metric. By default, they are sized by membership, e.g. the number of instances that belong to each center. This gives a sense of the relative importance of clusters. In our plot, we can see that the cluster sizes are quite varied, which we attribute to the limited amount of data. The spacing between the cluster centres shows that the clusters aren't very mixed up.

VII. Recommendation Algorithm

After the similarity model is generated, the question arises that how to recommend the next song considering the change in mood as a factor. Before we discussed in the model building section on how to generate playlist similar to the current song being played. As the mood starts to change, our algorithms should detect it and recommend some different kind of music. This can be achieved by using simple reinforcement learning technique.
What we do is that we recommend a song similar to the one being played, and if the user rejects it, (rejection of a song in this content can be referred as; amount of listening time of that

song being less that 50%) then suggest different song similar to the first one. If the user still rejects it, we can conclude that user wants different type of music. So then we suggest song from some different cluster and repeat the above process until we converge to the type of song the user wants to hear and suggests the next songs accordingly.

VIII. Conclusion

The method used in this task is a primitive approach to music suggestion. Although the predictions made by the model seems fair, due to limitations in the amount of data that we could procure, they couldn't perform up to the mark. In order to achieve a more comprehensive music suggestion, we need to perform both Content based filtering and Collaborative filtering.

Content based filtering focuses on suggestion based on acoustic properties of the music whereas Collaborative filtering uses data from patterns of other users who browsed the same music to suggest a track. On combining both of these factors, we can improve the quality of suggestions drastically.

References

[1] "Spotify web api," https://developer.spotify.com/documentation/web-api/.

[2] "Yellowbrick documentation," https://www.scikit-yb.org/en/latest/api/cluster/.

[3] "Kmeans clustering with scikit learn," https://towardsdatascience.com/k-means-clustering-with-scikit-learn-6b47a369a83c.

[4] "Clustering and dimensionality reduction," https://www.imperva.com/blog/clustering-and-dimensionality-reduction-understanding-the-magic-behind-machine-learning/.

[5] C. Ding and X. He, "K-means clustering via principal component analysis," August.

[6] K. Zhang, Z. Zhang, K. Bian, J. Xu, and J. Gao, "A personalized next-song recommendation system using community detection and markov model," August.