Final Project Report

Author(s): Juliana Alscher, Joel Brandinger, Clea Demuynck
Affiliation: Tufts University
Address: Medford, MA, 02155

1 Introduction

A major area of exploration in the music industry is how to classify songs, one method being by genre. As music evolves, historical genres fuse, and the boundaries of what constitutes a genre become blurred. This lack of standardization in defining musical genres became a major motivation for our project. Moreover, the task of analyzing and clustering songs based on their characteristics has been of great interest to researchers and music enthusiasts alike. Not only can it provide insights into the structure and patterns of music, but it can also serve as a powerful tool for recommendation systems and genre classification itself.

In this research project, we explore the use of K-means clustering as a means of grouping songs based on their acoustic features, such as beats per minute, energy, and danceability. By clustering songs this way, we aim to identify patterns and similarities that can inform our understanding of how the different features play into a song's genre label.

While K-Means can certainly be an effective model, we realize it is not probabilistic. Therefore, we suggest upgrading K-Means to a Gaussian Mixture Model. This upgrade will allow us to quantify uncertainty in our results better and make more informed decisions regarding our model. We will also explore upgrading our model to improve efficiency and runtime. This upgrade will be performed through a method called Mini-Batch K-Means.

2 Data and Analysis Plan

2.1 Data

To perform our analysis, we built a unique Spotify playlist containing 1417 songs with a variety of genres. We then used a website, Machinery, to extract 14 features for each song including title, artist, genre, and several acoustic features. The following table explains these features in detail:

Feature	Data Type	Description
title	String	Name of song
artist	String	Name of artist
top genre	String; categorical	Type of genre
year	Int; ; [1959,2023]	Year released
bpm	Int; [44, 208]	Tempo of the song
Energy	Int; [0, 99]	The energy of the song. Higher values correspond to a more energetic song.
Dance	Int; [0, 98]	The higher the value the easier it is to dance to the song.
dB	Int; [-37,-1]	The higher the value the louder the song.
live	Int; [4,96]	The higher the value the more likely the song is a live recording.
val	Int; [0, 98]	How positive the song is. Higher values mean that the song is a more positive.
len	Int; [45, 993]	Duration of the song in seconds.
acous	Int; [0, 100]	The higher the value the more acoustic the song
spch	Int; [0, 76]	The higher the value the more spoken words the song contains.
pop	Int; [0.0, 96.0]	The higher the value the more popular the song is.

The necessary preprocessing consisted of excluding certain features we did not feel would have an impact on clustering each song. We chose to remove year, liveliness, popularity, and duration because we felt that they should not be accounted for in classification of genre. Additionally, we chose to exclude genre as it is essentially the metric we were trying to evaluate. Next, we removed all songs containing any Nan values as this could potentially impact model performance. Figure 1 shows a bar graph of the number of songs in each genre after this preprocessing.

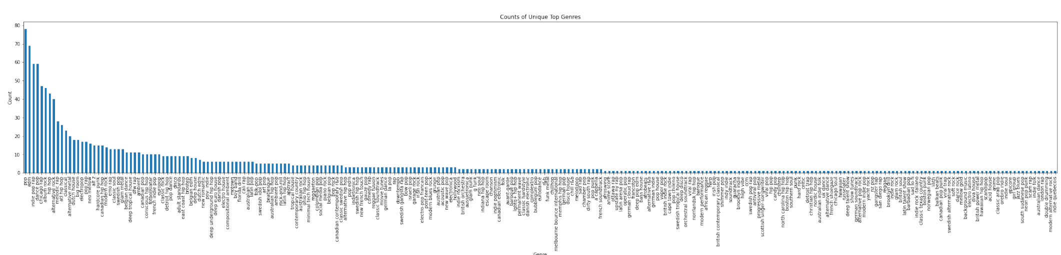

Figure 1: Frequency of All Genres

As can be seen, there are high counts of “pop” and "edm" songs, whereas many genres include just one song. Upon further analysis, we noticed that many of these less frequent genres are hyper specific and similar to one another—for example “canadian hip hop” and “north carolina hip hop” could both be more simply classified as “hip hop.” We began to consider what this class imbalance would mean for our k-means clustering process, particularly because we expect similar sub-genres to have similar features. In terms of visualization, we would expect the clusters of said sub-genres to overlap. With this to consider, we decided that we will explore methods for combining hyper-specific sub-genres into one genre (ie, “canadian hip hop” and “north carolina hip hop” as “hip hop”). Additionally, instead of running k-means with k equal to the total number of genres in the data set, we will explore methods for optimizing k, the number of clusters. This learned value of k is especially relevant considering our observations about overlapping genres. Figure 2 depicts a bar graph of the overarching genres and their counts. We further chose to remove any overarching genres with less than 10 songs. At the conclusion of our preprocessing, we were left with 1166 songs.

We were able to implement this preprocessing by importing our dataset and storing it as a Pandas DataFrame. We then refered to the pandas.DataFrame documentation for further access and manipulation. Notably, preprocessing also called for the use of sklearn.preprocessing.StandardScaler which allowed us to standardize our features.

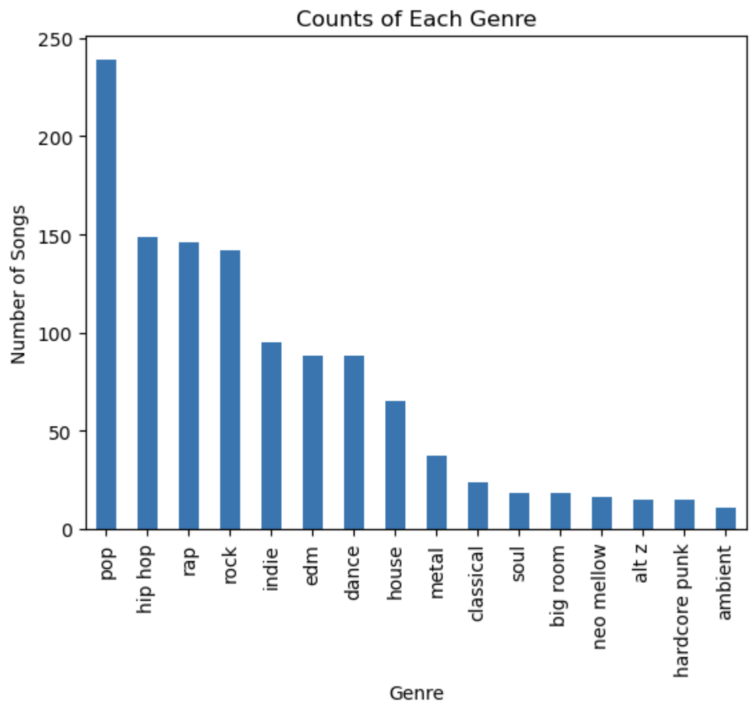

Figure 2: Frequency of Overarching Genres

2.2 Analysis Plan

The features we will take into account for clustering the songs will be beats per minute, energy, danceability, dB, valence, acousticness, and speechiness. To evaluate performance of the model, genre will be excluded so we may use it for validation.

The data will be split into 70% training and 30% testing. This split results in 816 data points for training and 350 for testing. We felt that this split gave us enough data to train our models, and left enough of the dataset unseen for us to gauge the performance of our clustering on new songs.

To evaluate performance of the baseline K-Means model, we will calculate objective cost, or the sum of square distances from each data point to the center of its assigned cluster. The equation for cost can be seen as follows:

$$J(x_{1:N}, r_{1:N}, \mu_{1:K}) = \sum_{n=1}^{N}\sum_{k=1}^{K} r_{nk}(x_n - \mu_k)^T(x_n - \mu_k)$$

As we are proposing a probabilistic upgrade, we will also compute log-likelihood of the baseline model to allow for easier comparison with the GMM upgrade. While log likelihood is traditionally a metric reserved for Gaussian Mixture Models, we can manipulate the function to be applicable to our K-Means model. We will define log likelihood in this setting to be as follows:

$$\mathcal{L}(x_{1:N}|z_{1:N}, \mu, \sigma^2) = \log p(x_{1:N}|z_{1:N}, \mu, \sigma^2) = \sum_{i=1}^{n} \log \mathcal{N}(x_i|\mu_{z_i}, \sigma^2)$$

Where $z_{1:N}$ are the assignments of the data points to clusters.

Since K-Means has hard assignments, each data point is assigned to a single cluster with a probability of 1. Therefore, we will calculate μ_k of each cluster as the centroid of the data points assigned to that cluster, and the variance σ^2 is the mean squared distance of the data points to their cluster centroid.

If C_k denotes the set of data points assigned to the k-th cluster, then the mean μ_k is given by:

$$\mu_k = \frac{1}{|C_k|} \sum_{x \in C_k} x$$

The variance σ^2 is given by:

$$\sigma^2 = \frac{1}{n} \sum_{i=1}^{n} \|x_i - \mu_{z_i}\|^2$$

3 Baseline Method

Our baseline K-Means model was implemented using the scikit-learn "KMeans" function sklearn.cluster.KMeans. This algorithm minimizes the objective cost to generate cluster centers. To pick our desired value of K we tested a range of values from 1 to 100. Each number for k was fit on the train set then scored on the test set using predicted outputs. Using this approach we landed on 95 clusters being the optimal amount. This model generated a cost of 427.81. Here is a mathematical explanation of this algorithm, based on class notes (Hughes [c])...

Inputs:

- Dataset: $x_1, ..., x_{1417}$ where $x_n \in R^D$
- Initial cluster locations: $\mu^0_{1:K}$

Cost Function:

$$J(x_{1:N}, r_{1:N}, \mu_{1:K}) = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk}(x_n - \mu_k)^T(x_n - \mu_k)$$

Procedure:
for iter t = 1... until converged:

- $r^t_{1:N} \leftarrow \arg\min_{r_{1:N}} J(x_{1:N}, r_{1:N}, \mu^{t-1}_{1:K})$
- $\mu^t_{1:K} \leftarrow \arg\min_{\mu_{1:K}} J(x_{1:N}, r^t_{1:N}, \mu_{1:K})$

return optimal cluster locations $\mu^*_{1:K}$ and optimal assignment indicators $r^*_{1:N}$

This coordinate descent implementation of the k-means algorithm iterates until we reach a convergence where each iteration includes an assignment update step and a cluster location update step. Each step is guaranteed to improve the cost function or stay the same when it reaches convergence. Convergence can be assessed by determining a stop in the change of the assignment indicators or for the cost improvement to become minimal.

In order to analyze the performance of the baseline model we looked at the resulting score of the cost function which was 427.81 and the modified log-likelihood as explained above was -6.656.

4 Proposed Upgrade Method

After exploring our dataset and baseline method, we propose two upgrades: Mini-Batch K-Means and a Gaussian Mixture Model (GMM). We hypothesize that Mini-Batch K-Means will improve runtime for the model to converge with minimal sacrifice to accuracy. We also hypothesize that a GMM will give us more insight into uncertainty of model predictions as it will give us probabilistic results allowing for more informed decisions in regards to model clustering and performance.

4.1 Upgrade 1: Mini-Batch K-Means

We hypothesize that the Mini-Batch K-Means will outperform K-Means in runtime. We imported the python Time module to access our runtime measurements.

Mini-Batch K-Means updates the centroids (i.e., the cluster centers) using a subset of the data at each iteration, rather than the entire dataset. This allows Mini-Batch K-Means to converge faster than standard K-Means, especially when dealing with large datasets. We used sklearn.cluster.MiniBatchKMeans for this implementation.

The algorithm for Mini Batch K-Means is similar to that of K-Means but with an additional hyperparameter, M, which is the batch size. We plan to test 5 different batch sizes: 49, 100, 256, 512, and 1024, and timed model convergence for a range of k-values from 1 to 100. We will compare these run times to K-Means along with model cost to gain an understanding of the time-accuracy trade-off.

4.2 Upgrade 2: Gaussian Mixture Model

We hypothesize that a Gaussian Mixture Model will provide more detailed information of our data through probabilistic assignments. A GMM assumes data to be generated by a mixture of several Gaussian distributions, each representing a different cluster. The model estimates the parameters of these Gaussians and assigns each data point to the most likely cluster. We believe this will result in better performance for cases where the cluster boundaries are not well defined.

To train our GMM we will use a similar approach to training our K-Means model, but instead of minimizing cost as the objective function, we will use per-sample average log likelihood. That is, we will fit the model on our training set and then predict the log likelihood on the test set. As GMM's assignments are probabilistic cost isn't necessarily a correct implementation and log likelihood is therefore preferred. We will implement this upgrade using scikit-learn's Gaussian Mixture. This will allow us to use the built-in score function to calculate per sample average log-likelihood.

Based on class notes (Hughes [b]), the algorithm for training GMM can be described as follows...
Inputs

- Dataset: $x_{1:N}$ such that $x_n \in R^D$
- Number of assumed clusters: K

GMM Parameters
The goal of this model is to learn optimal GMM parameters for K clusters.

- Assignment probabilities: $\pi_{1:k} = \pi_1, \pi_2, \ldots, \pi_k$ where $\pi_{1:k} \in \Delta^k$
- Cluster locations: $\mu_{1:k}$ where $\mu_{1:k} \in R^D$
- Cluster covariances: $\Sigma_{1:k}$ where Σ_k is $D \times D$ symmetric positive definite

Pseudo Code for EM Hughes [a]
The EM step updates our GMM paramters as follows until the model converges.

- Parameters π, μ, σ initialized at random
- Iterate t $\in 1, 2, \ldots, T$:
  - Iterate for examples n $\in 1, 2, \ldots, N$:
    - E Step: Updating assignment probability ...vector, r_n

$$r_{nk} = \frac{\pi_k \prod_d \text{NormPDF}(x_{nd}|\mu_{kd}, \sigma_{kd})}{\sum_{\ell=1}^{K} (\pi_\ell \prod_d \text{NormPDF}(x_{nd}|\mu_{\ell,d}, \sigma_{\ell,d}))}$$

  - Iterate for clusters $k \in 1, 2, \ldots, K$:
    Updating GMM parameters based on current state...
    - M step for weights: $\pi_k = \frac{\sum_n r_{nk}}{N}$
    - M step for means: $\mu_{kd} = \frac{\sum_n r_{nk} x_{nd}}{\sum_n r_{nk}}$
    - M step for variances: $\sigma^2_{kd} = \frac{\frac{1}{s \cdot m} + \sum_n r_{nk}(x_{nd} - \mu_{kd})^2}{\frac{1}{s \cdot m} + \sum_n r_{nk}}$

To tune our hyperparameter for the number of clusters we will test a range of K values from 1 to 100, just as we did for our baseline K-Means model.

To implement GMM, we utilized the scikit-learn sklearn.mixture.GaussianMixture and its functions. As a performance measure, we called its score function which calculates the per-sample average log-likelihood.

5 Results

5.1 Mini Match K-Means Results

After performing our upgrade tests for Mini-Batch K-Means we determined that our hypothesis was correct. Mini Batch outperformed K-Means in runtime for all batch sizes we tested. To contrast the runtime of Mini-Batch against K-Means, we measured how long the model took to converge for each batch size along with the range of K-Values (number of clusters). Figure 3 and Figure 4 depict the results of these tests.

```
Batch size = 49:       best cost of 432.95    w/ 99 clusters  took 0.089 seconds
Batch size = 100:      best cost of 435.13    w/ 99 clusters  took 0.093 seconds
Batch size = 256:      best cost of 428.59    w/ 99 clusters  took 0.132 seconds
Batch size = 512:      best cost of 432.21    w/ 94 clusters  took 0.129 seconds
Batch size = 1024:     best cost of 419.30    w/ 93 clusters  took 0.126 seconds
K–Means baseline:      best cost of 427.81    w/ 95 clusters  took 0.136 seconds
```

Figure 3: Mini-Batch K-Means Results

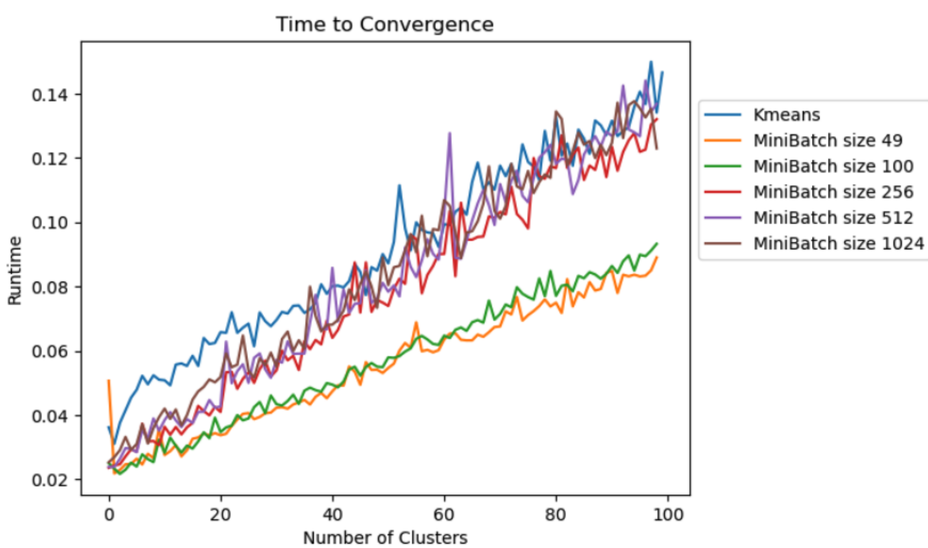

Figure 4: Runtime for Various Model Convergence

The best scoring model in terms of cost was generated using a batch size of 1024 and 93 clusters. Surprisingly enough, this model actually outperformed the baseline on the objective cost and scored scored 419.30 as opposed to 427.81 for the baseline. This model also had a better runtime of 0.126 seconds compared to 0.136 for the baseline, meaning this model outperformed the K-Means baseline in all aspects, which was quite incredible.

When it comes to pure runtime, the model with a batch size of 49 resulted in the best time of 0.089 seconds and achieved a cost of 432.95 with 99 clusters. This runtime was a significant increase in comparison to the baseline K-Means model which ran in 0.136. The increase came with a minimal sacrifice in accuracy, as the baseline had a cost of 427.81. Overall, we were very pleased with the results of the Mini-Batch K-Means upgrade.

5.2 GMM Results

After training our GMM, the log-likelihood of the model was maximized using 8 clusters. This result was quite surprising as it is much lower than the optimal number of clusters for our baseline K-Means model. In order, to evaluate the performance of our GMM in relation to our K-Means model we used a modified version of log-likelihood so that we could calculate a value for our K-Means model. In this regard, the best K-Means model with 95 clusters scored slightly better with a log likelihood of -6.656 while the best GMM with 8 clusters had a log likelihood of -6.865.

In attempt to visualize our results, we performed principal component analysis (PCA) to reduce our 7 feature vectors down to 2. We were then able to plot the clusters and their mean location. The results can be observed in Figure 5

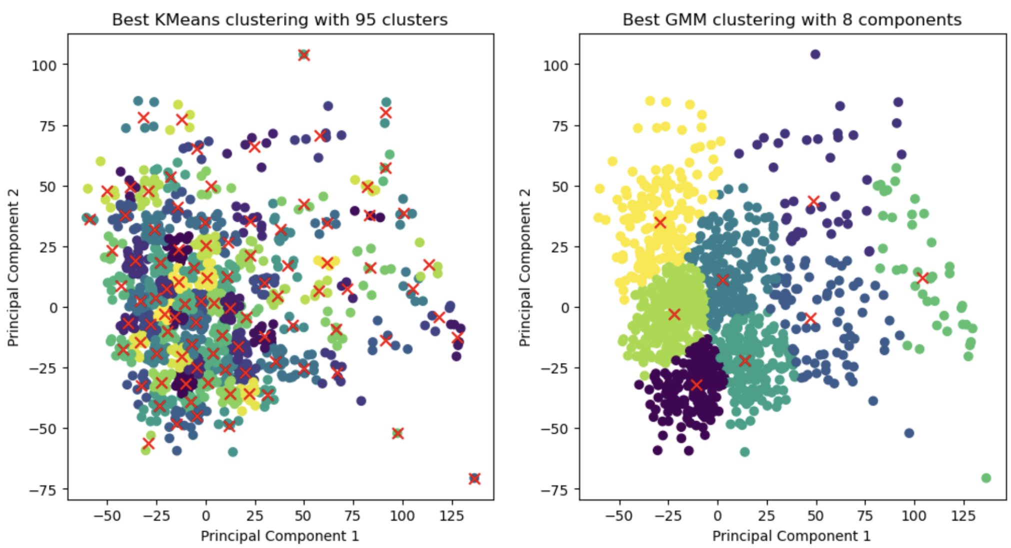

Figure 5: K-Means vs GMM

Figure 6 and Figure 7 show the songs that were placed into cluster 0 for both K-Means and GMM. Note that the cluster for GMM does not contain all the songs as the GMM had only 8 clusters and each contained roughly over 100 songs.

```
Lucid Dreams - Juice WRLD
A Thousand Years - Christina Perri
Knockin' On Heaven's Door - Guns N' Roses
Stole the Show - Kygo
Fine China - Future
Freaky Friday (feat. Chris Brown) - Lil Dicky
Cinderella Man - Eminem
Pretty Girl - Cheat Codes X CADE Remix - Maggie Lindemann
To Be With You - 2010 Remastered Version - Mr. Big
I Remember - Cheat Codes
Rock N Roll - Scorey
BLUE - Tiësto
Tough (feat. Noah Kahan) - Quinn XCII
Seesaw - Ummet Ozcan
They Call Me Tiago (Her Name Is Margo) - Tiagz
```

Figure 6: K-Means Cluster 0

```
God's Plan - Drake
Lucid Dreams - Juice WRLD
See You Again (feat. Charlie Puth) - Wiz Khalifa
Dile - Don Omar
SAD! - XXXTENTACION
Pa' Que Retozen - Tego Calderón
Going Bad (feat. Drake) - Meek Mill
Stronger - Kanye West
Gold Digger - Kanye West
Run This Town - JAY-Z
Homecoming - Kanye West
Where Is The Love? - Black Eyed Peas
GONE, GONE / THANK YOU - Tyler, The Creator
Danza Kuduro - Don Omar
Runaway - Kanye West
ROCKSTAR (feat. Roddy Ricch) - DaBaby
Go Flex - Post Malone
Taki Taki (with Selena Gomez, Ozuna & Cardi B) - DJ Snake
Killing Me Softly With His Song - Fugees
Lie - NF
Nevermind - Dennis Lloyd
Slow Hands - Niall Horan
Good Life - Kanye West
Stole the Show - Kygo
Let Her Go - Passenger
Beautiful People (feat. Khalid) - Ed Sheeran
No Church In The Wild - JAY-Z
Swervin (feat. 6ix9ine) - A Boogie Wit da Hoodie
```

Figure 7: GMM Cluster 0

5.3 Conclusion

The motivation for this project was to investigate how different features of a song play into its genre label. In this exploration, we used three different models, our base model of K-Means, followed by Mini Batch K-Means and Gaussian Mixture Models as upgrades. The aim was to cluster our dataset of songs based on their musical features and analyze the results to determine if these features influence the definition of a song's genre, or if additional components are more relevant to the categorization of a song.

The results of our two upgrades were surprising compared to our baseline model. First, the best cost of Mini Batch K-Means was similar to that of K-Means. However, runtime seemed to significantly improve as batch size decreased. Even though our data set was not so large that runtime posed problem, using Mini Batch as an optimization to K-Means seems like an interesting model to explore. For future projects, we could compare K-Means and Mini Batch K-Means models on a much larger data set to see if this runtime optimization would serve as a significant upgrade.

Another surprising factor was the fact that our K-Means Model and Gaussian Mixture Model (GMM) gave a dissimilar number of optimal clusters: K-Means being 95 and GMM being 8. That being said, this large difference in the number of clusters still gave similar log-likelihoods for each model, -6.656 and -6.865 respectively. Since the log-likelihoods for the two models are close to each other, we concluded that overall, there was no improvement when it came to using K-Means or GMMs in this investigation. However, while the log-likelihoods are very similar, the difference between the two models and their results boils down to the size of their clusters. Our K-Means model produced clusters that included an average of 15 songs while GMM has approximately 177 songs per cluster. Looking at the results, we were impressed by how the songs were clustered together. Comparing the individual genre label of each song, we noted that dissimilar genres were clustered together. However, upon further inspection of the songs in each cluster, we could identify similarities. For example, each song in K-Mean's cluster 0, shown in Figure 6, has comparable valence. Consider "A Thousand Years" by Christina Perri and "Knocking on Heaven's Door" by Guns N' Roses. Although they would be defined as classic rock and soft pop respectively, both songs are observably similar in the way that they are not very upbeat, and have similar composition regarding instrumentals. For each song in cluster 0, we could recognize this pattern which led us to agree that all songs worked well as a cluster of similar data points.

These findings raise the question of how genre can be defined at all. It is interesting to see how our models clustered songs and how these clusters differed from our expectations. While we expected to see the grouping of songs by their pre-defined genres, the actual results did not reflect this. Despite the varying genre labels, we know that the clusters were based on given features and we were able to hear commonalities between the clustered songs. Our group discussed the ways in which long standing genres have evolved and combined overtime. Genre does not bound an artist's creative process and especially as technology broadens access to varied music, the music industry has seen the creation of songs that take inspiration from multiple genres. Additionally, in our project introduction we introduced music recommendation algorithms as an application of song classification methods. As this practice advances, we note the value in identifying similar songs by features like valence, dancability, etc. As these fields develop, music classification may be more suited to labels other than genre.

6 References

References

Michael Hughes. Cp4: Gaussian mixture models. https://www.cs.tufts.edu/comp/136/2023s/cp4.html, a.

Michael Hughes. day16.pdf. https://www.cs.tufts.edu/cs/136/2023s/notes/day16.pdf, b.

Michael Hughes. day15.pdf. https://www.cs.tufts.edu/cs/136/2023s/notes/day15.pdf, c.

Playlist Machinery. Organize your music. http://organizeyourmusic.playlistmachinery.com/.

pandas.DataFrame. pandas.dataframe. https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html.

sklearn.cluster.KMeans. sklearn.cluster.kmeans. https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html.

sklearn.cluster.MiniBatchKMeans. sklearn.cluster.minibatchkmeans. https://scikit-learn.org/stable/modules/generated/sklearn.cluster.MiniBatchKMeans.html.

sklearn.mixture.GaussianMixture. sklearn.mixture.gaussianmixture. https://scikit-learn.org/stable/modules/generated/sklearn.mixture.GaussianMixture.html.

sklearn.preprocessing.StandardScaler. sklearn.preprocessing.standardscaler. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html.

Time. Time module. https://docs.python.org/3/library/time.html.