Genre Classification of Songs using Supervised and Unsupervised Machine Learning

Palmy Klangsathorn, Matthew Hall, Prompt Eua-anant
CS320: Machine Learning
Email: {klangsathornp, hallm6, eua-anantp}@carleton.edu

***Abstract*—In this project, our goal was to investigate the extent to which a song's genre can be predicted based on Spotify's audio features, such as tempo, acousticness, and speechiness (how many words are in a song). We experimented with both supervised learning models (multinomial logistic regression and neural networks) and unsupervised learning techniques (k-means, DBSCAN, hierarchical clustering). Our goal is not only to evaluate the accuracy of the predicted models but also to investigate whether clusters derived from our unsupervised models correspond to Spotify's labeled genres. We apply various regularization techniques, explore multicollinearity and interaction terms, and use interpretability tools to assess feature importance. We find that while supervised models can reach moderate accuracy in genre prediction, clustering results reveal low accuracy score and suggests that Spotify's musical features do not sufficiently capture the subjectivity and complexity of genre labels. Still, the most distinct genres lend Spotify's musical features some predictability. Moreover, genres that sound similar in name are not necessarily similar in musical features.**

1. Introduction

Music genres are culturally and perceptually constructed categories that attempt to organize songs with shared characteristics. While intuitive to humans, genres can be ambiguous and fluid, making them difficult to classify computationally. Our project investigates whether the genre of a song can be predicted using Spotify-provided audio features, which represent acoustic, rhythmic, and structural characteristics. We also explore whether unsupervised clustering of songs reveals natural genre groupings.

We use supervised models to classify songs into one of 82 genres and unsupervised methods to identify patterns in the data without genre labels. Our input includes features like tempo, energy, acousticness, mode, key, time signature, and speechiness. The intended output is the predicted genre label (in supervised models) or a set of meaningful clusters (in unsupervised models) that align with genre patterns.

2. Related Work

Previous studies have shown success in music classification tasks using features like lyrics, chords, and acoustic attributes. For instance, a 2019 USC study demonstrated high genre classification accuracy using lyrics and chords [1]. We build on this by focusing purely on Spotify's numerical and categorical audio features, which are both accessible and scalable.

Spotify's API includes engineered features (e.g., danceability, speechiness) derived from proprietary models. While these may already incorporate some genre-related patterns, our project evaluates how effectively these features alone can predict genre or reveal genre-like clusters. Previous works on deep learning for tabular data, such as fast.ai's guide [2], inform our neural network design, especially for handling categorical variables [3].

3. Dataset

We use the Spotify 1M Tracks dataset [4], which contains over 800,000 tracks. Each track includes numerical features (e.g., tempo, loudness) and categorical features (e.g., key, time signature). Our target variable is the genre label, with 82 unique genres.

We filtered the data to include only tracks released in 2022, resulting in 50,000 examples. This subset provides modern music and manageable computational complexity. Our preprocessing steps included one-hot encoding of categorical variables for our logistic regression models and neural networks, as well as z-score normalization of continuous variables. We investigated min-max normalization of our continuous features to make sure that they contained values between 0 and 1, but this did not result in a noticeable difference, so we decided to stick with z-score normalization. Also, we performed minimal feature engineering but added interaction terms in some logistic regression models to study their effects.

Furthermore, we used each of our model structures on three different versions of the data set described above. The first one was the exact same as the data set above, with all 82 genres contained. Next, we ran the models with 77 genres from the same dataset, removing five genres that we deemed as illegitimate: French, German, Spanish, Swedish, and Indian. We decided that these genres were illegitimate because they seemed to contain songs that were from multiple genres but were simply in the language denoted by the name of the Spotify-assigned genre. Finally, in our third dataset, we keep all modifications we made to the second dataset, but we only include the 26 most dominant genres we ascertained from running the k-means algorithm with k=30. To obtain the most dominant genre in each cluster, we find the genre that has the highest proportion of (count in that cluster)/(total count in dataset). This results in 26 unique genres. This third dataset was then used in our various structures of predictive models.

To build a dataset for unsupervised learning, we exclude all categorical variables because they are unfit for all 3 of our clustering algorithms. This leaves us with a dataset of 10 quantitative features (danceability, energy, loudness, speechi-

ness, acousticness, instrumentalness, liveness, valence, tempo, duration). We then perform several modifications to arrive at four different versions of the dataset.

- Version 0: all features are Z-score normalized
- Version 1: all features are Z-score normalized, and all non-musical genres are excluded. These include French, German, Indian, Swedish, and Spanish, all of which are based on Language and not musicality.
- Version 2: all features undergo min-max scaling, and all non-musical genres are excluded.
- Version 3: same as version 2, but includes the 26 dominant genres described in the above section.

In sum, we included various numerical and categorical musical attributes in our predictive models as features. However, in our clustering analyses, we did not include any categorical features, which is something that it may be beneficial to look at in future studies. The numerical features that we included were danceability, energy, loudness, speechiness, acousticness, instrumentalness, liveness, valence, tempo, and duration. The categorical features included in our predictive models were key, mode, and time signature. We one-hot encoded these categorical features using `OneHotEncoder` from `sklearn`. More detailed information on what our features measure and how they are calculated can be found here [4]. It is worth noting that these features described above are not all of the variables in the data set. We did not use variables that were not audio features, such as popularity and the release year of a song.

4. Methods

A. Supervised Models

*1) **Multinomial Logistic Regression Experiments**.* To understand the impact of different factors on model performance, we conducted ten experiments using multinomial logistic regression. Each model was evaluated on a validation set and a separate test set, reporting accuracy, log loss, and detailed classification results.

Before model training, we examined multicollinearity by calculating the Variance Inflation Factor (VIF) for each feature. Features with a high VIF (typically above 5 or 10) indicate strong multicollinearity and were considered for PCA. VIF was computed using the formula:

$$\text{VIF}_i = \frac{1}{1 - R_i^2}$$

where R_i^2 is the coefficient of determination when regressing the i-th feature on all other features.

To further capture potential nonlinear relationships and interactions between features, we expanded the feature space using pairwise interaction terms and polynomial features. This was done using the `PolynomialFeatures` transformer from `scikit-learn` with degree 2 and `include_bias=False`. This process increased the number of features significantly (from 30 to 495), allowing the model to account for second-order interactions and squared terms. After expansion, VIF was re-evaluated to identify and address any new multicollinearity introduced. All models are done

We then conducted the following experiments:

a) Unregularized Models. These models disabled regularization (`C` $=$ `1e10`), allowing the models to learn without constraints.

- **Experiment 1: Unregularized (Base)**
  A fundamental multinomial logistic regression model was trained directly on the scaled and one-hot encoded base features. This setup applied 5-fold stratified cross-validation on the training set to provide a more robust and less biased estimate of the unregularized model.
- **Experiment 2: Unregularized + Interaction Terms**
  Combining the approaches from Experiment 2 and 3, 5-fold stratified cross-validation was performed on the training data that included interaction terms. This assessed the generalization performance of the unregularized model in an expanded feature space.
- **Experiment 3: Unregularized + Interaction Terms + PCA**
  In this configuration, `PCA` was applied to the features after the generation of interaction terms. The unregularized logistic regression model was then trained on this dimensionality-reduced feature set, with its performance evaluated using 5-fold stratified cross-validation.

b) Regularized Models. These models included regularization to prevent overfitting using L2 and Elastic Net Penalties (both L1 and L2).

- **Experiment 4: Regularized (Base)**
  Logistic regression with L2 regularization, tuned using `GridSearchCV`.
- **Experiment 5: Regularized + Interaction Terms**
  Applied L2 regularization to feature set with interaction terms.
- **Experiment 6: Regularized + PCA**
  Trained on PCA-transformed features to reduce dimensionality and manage multicollinearity.
- **Experiment 7: Regularized + Interaction Terms + PCA**
  This experiment highlights the combined application of interaction terms and `PCA` before L2 regularization. The setup is identical to Experiment 8, with the L2 regularized model trained on the `PCA`-transformed features (from interaction terms), and optimal `C` found via `GridSearchCV` and 5-fold stratified cross-validation.
- **Experiment 8: Regularized + Elastic Net**
  This advanced regularized model utilized the Elastic Net penalty, which offers a balance between L1 (Lasso) and L2 (Ridge) regularization. The `saga` solver was specifically employed to support this penalty. `GridSearchCV` with 5-fold stratified cross-validation was used to comprehensively tune both the regularization strength `C` and the `l1_ratio` (ranging from 0 for pure L2 to 1 for pure L1 regularization).

2) Neural Networks. We investigated how various different structures of Keras Sequential (reference) neural networks performed in classifying songs in our dataset into the correct genres. Sparse categorical cross entropy was used as our loss function for all of our neural networks. Our chief evaluating tool was the average test accuracy after cross validation and we also paid some attention to model overfitting. The experiments for different structures of neural networks that we investigated are as follows:

- **Experiment 1: Unregularized Neural Network**
  Our first neural networks included no regularization. This was done to give us a baseline to improve upon and compare to. We were not expecting any good performance at all with this model structure.
  The basic structure was a Keras Sequential model with an input layer, two dense layers, and an output layer. The shape of the input layers matches the shape of data frame of features. The first dense layer includes 128 nodes and uses the ReLU activation function. Directly after that is the second dense layer, which included 64 nodes and also uses the ReLU activation function. Then, there is the output layer, which includes 82 nodes (one for each genre) and uses the SoftMax activation function to help us get interpretable probabilities for each genre. K Fold Cross validation for this structure of model was performed in an effort to optimize the learning rate and batch size. Using five as the value of K, we found that the optimal learning rate for this structure was 0.001 and that the optimal batch size was 64. Our final neural network with this model structure and these hyperparameter values produced an average test accuracy of 28.88 percent on our full dataset (with 82 genres). On the dataset excluding the five illegitimate genres, this structure achieved an average test accuracy of 30.48 percent. Finally, on the heavily reduced dataset with 26 genres, this model structure achieved an average test accuracy of 60.36 percent. Each set of training consisted of 30 epochs. This model structure resulted in overfitting to the training data and high variance, as the training accuracy was consistently higher than the validation and testing accuracy by three to five percent. Additionally, in training the validation loss plateaued and then began to increase as the amount of epochs went on, while the training loss decreased consistently over all of the epochs. This is another sign of overfitting. At the end of training, the training loss would typically be about 0.3 higher than the validation loss.
  This overfitting seen in this structure is to be expected since there is no regularization present. Furthermore, the lower level of accuracy seen here compared to the other experiments (see below) is also to be expected due to the lack of regularization, as regularization generally increases the generalizability of a model.
- **Experiment 2: L2 Regularized Neural Network**
  Our second set of neural networks included L2 regularization on top of the structure described in Experiment 1. We chose to include L2 regularization because it was the form of regularization that we were the most familiar with and we thought that it would increase the accuracy and decrease overfitting. We thought that this would be done by penalizing large weights.
  L2 regularization was included through the *kernel_regularizer* argument in each Dense layer being set to *keras.regularizers.l2(l2 = regularizationStrength*. Through K-Fold cross validation with k equal to 5, we found that the optimal L2 regularization strength was 0.0001. On our full dataset, our final neural network with this structure achieved an average test accuracy of 29.2 percent. On the dataset without illegitimate values, this structure was able to achieve an average test accuracy of 30.94 percent. Then, on the heavily reduced dataset with only 26 genres, this model structure achieved an average test accuracy of 60.5 percent. Once again, each set of training consisted of 30 epochs. We investigated training for 50 epochs some with this structure, but it did not result in noticeable improvement, so we did not up the epochs from there. This model structure resulted in a similar degree of overfitting to the structure from Experiment 1. The training accuracy was consistently higher than the validation and testing accuracy by an amount similar to Experiment 1 (three to five percent). Also, in training the validation loss plateaued and then began to increase as the amount of epochs went on, while the training loss decreased consistently over all of the epochs. The difference at the end of training between the training and validation loss was again about 0.3 typically like in Experiment 1. This lack of reduction in overfitting when adding L2 regularization could be due to a number of reasons. This could in part be due to a few features being highly correlated, such as loudness and energy, as L2 regularization does not solve the issue of choosing among highly correlated features. This could also be due in part to outliers in the data, which is not something that we explored, among other reasons. Furthermore, the slight increase in accuracy seen with the L2 regularization while overfitting does not improve may certainly be due to highly correlated features. This is because L2 regularization penalizes models for large weights, so it may spread influence across correlated features, helping to smooth out the model slightly, hence the slight accuracy improvement.
  The slight accuracy improvement still renders this structure better than that of Experiment 1. However, we thought that this structure would do a lot better at reducing overfitting, but thinking about the data and how L2 works, we understand why this did not happen.
- **Experiment 3: L2 and Dropoout Regularized Neural Network**

  Our third and final set of neural networks added dropout regularization onto the structure from Experiment 2. We

chose to add dropout regularization thinking that it would increase the accuracy and decrease the overfitting of our model by helping the model learn more generalizable paths.

Dropout regularization was done by adding Dropout layers after each Dense layer in the neural network. We found the optimal random dropout rate for nodes to be 0.1 after performing K-Fold cross validation with K equal to five. With the full version of our dataset, this network structure was able to achieve an average test accuracy of 29.76 percent. On the version of the dataset without illegitimate genres, this network structure was able to achieve an average test accuracy of 30.96 percent over 30 epoch training sessions, 31.52 percent over 50 epoch training sessions, and 32.05 percent on 100 epoch training sessions. Despite the increase in accuracy when the number of epochs was increased, symptoms of overfitting also increased when the number of epochs was increased. However, the overall level of overfitting decreased compared to Experiment 1 and Experiment 2, with even the 100 epoch trained models having less difference in training and validation loss, as well as accuracy. With the heavily reduced version of the dataset with 26 genres, this structure achieved an average test accuracy of 60.79 percent with 30 epoch training sessions. With all three versions of the dataset and 30 epoch training sessions, overfitting dramatically decreased. The typical difference between training accuracy and validation and test accuracy was less than a percent at the end of training. Furthermore, the difference in training and validation loss typically less than 0.1 at the end of training. Furthermore, the overall level of overfitting decreased compared to Experiment 1 and Experiment 2 in even the 100 epoch trained model. This had less difference in training and validation loss (0.15 on average after training), as well as accuracy(two percent on average). So, dropout regularization did a better job handling the high correlation between some features. By randomly dropping nodes, dropout regularization allows for the model to learn more general spread out or distributed representations of the features. This hereby encourages independence among the nodes, ensuring that the network does not rely on a few correlated inputs being present in every pass. Thus, it follows that dropout regularization would help reduce overfitting with this data, while L2 regularization might not. Additionally, the test accuracy improved slightly in this experiment likely due to discouraging reliance on specific paths or inputs, especially if there are highly correlated inputs, since dropout forces models to work even when some features are missing. Therefore, it forces the model to learn to use correlated features independently.

The slight increase in test accuracy as well as the large reduction in overfitting render this network structure as superior for our data compared to the structures presented in Experiments 1 and 2. We expected this structure to be better than the first structure by virtue of adding some regularization, but we did not originally anticipate that this would reduce overfitting by so much compared to the second structure. However, what we discovered is in line with current machine learning theory and has helped us build a better understanding of different regularization techniques.

B. Unsupervised Models

For unsupervised learning, we used three clustering algorithms:

K-means clustering: essentially, this algorithm groups the data points into k clusters such that the inertia, or the sum of the squared Euclidean distances between each data point and its cluster center, is minimized. It works by proceeding though the following steps. First, it uses the K-means++ algorithm to assign each data point a label, which designates which cluster the data point belongs to. The assigned labels won't minimize the inertia, but it serves as a good starting point to begin the clustering process, which consists of (1) compute the center of each cluster, and (2) for each data point, reassign it to the cluster where its center is nearest. The algorithm repeats (1) and (2) until the difference in error between the previous iteration and the current iteration is below the set threshold.

DBSCAN: this algorithm is suitable for finding nested clusters (clusters that spatially envelop another cluster - imagine a donut-shaped cluster enveloping another cluster). Unlike k-means, this algorithm does not require the user to specify the number of clusters k, but instead the user specifies epsilon and min_samples. It works by finding all the "core points" - points that contains at least min_samples points within the radius of epsilon. Points that are within the radius of a core point are assigned to the same cluster as that core point. When the algorithm finishes running, it assigns points that are connected via core points the same cluster, and label points that lack adjacent points within its radius as "noise" - the outliers that doesn't belong in any cluster.

Hierarchical clustering. It works by finding the two elements in the dataset with the lowest Euclidean distance between them, then group these two elements together. It repeats the process again, and if it has to compute the Euclidean distance between two groups of elements, then it finds the minimum distance between any two points across the two groups. This method of computing the difference between two groups is also known as the Nearest Point Algorithm.

5. Experiments and Results

A. Supervised Models

1) Multinomial Logistic Regression. We experimented with three main feature sets:

- Base: Scaled numeric features and one-hot encoded categorical features. This provides a simple baseline with no transformations.
- inter.: Base features with added interaction terms via polynomial features expansion (degree 2, interaction

only). This was hypothesized to improve the performance by capturing nonlinear feature interactions.
- PCA: Dimensionality reduction applied to base, using PCA to reduce noise and multicollinearity. PCA was introduced to solve any potential overfitting from interaction terms, especially when features space grew large.

For experiments, we trained multinomial logistic regression models using combination of the above feature sets, with and without regularization (L2 and elastic net). For hyperparameters, in regularized models, we used `GridSearchCV` to tune regularization strength `C` and elastic net mixing parameter `l1_ratio` when applicable. And polynomial interactions used `PolynomialFeatures(interaction_only=True)`. For evaluation, We used test accuracy as the primary metric, supplemented by log loss and classification reports during model development.

Results from the multinomial logistic regression:

- For accuracy performance across 82 genres (Figure 1), the best-performing model for multinomial logistic regression was the version with interaction features and L2 regularization, achieving a test accuracy of 27.79 percent. Additionally, for this best model, across those 82 genres, the F1-score for comedy is 0.92, followed by sleep at 0.89, grindcore at 0.54, both salsa and tango at 0.53, and romance at 0.52.
- The best-performing model across 77 genres (Figure 2), excluding language-related ones, is the regularized model with interaction terms, achieving a test accuracy of 29.62 percent. Moreover, across those 77 genres, the F1-score for comedy is 0.91, followed by sleep at 0.90, grindcore at 0.56, and drum-and-bass and salsa both at 0.52.
- For test accuracy across 26 genres (Figure 3), the best-performing model is the regularized model with interaction terms, achieving a test accuracy of 60.61 percent.
- For all three versions of the dataset, the models with interaction terms alone yielded the highest test accuracy. This suggests that including interaction features significantly enhances the model's ability to capture relationships between input variables and genre labels.
- PCA slightly reduced performance when combined with intermediate features, suggesting that it may have removed useful variance.

2) Neural Networks.

- The best performing structure is for all 82 genres, the 77 genre subset, and the 26 genre subset was the one with L2 and Dropout regularization, which recorded average test accuracies of 60.36, 60.5, and 60.79 percent respectively.

Results:

	Models	Test Accuracy
Multi	Unreg (Base)	24.59
	Unreg w/ (inter.)	27.16
	Unreg w/ (inter. + PCA)	26.36
	Reg (Base)	24.56
	Reg w/ (inter.)	27.79
	Reg w/ (PCA)	26.50
	Reg. w/ (inter. + PCA)	26.50
	Reg. w/ (elastic)	24.81
NN	Unreg	28.88
	L2 Reg	29.2
	L2 + Drop Reg	29.76

Figure 1: Test Accuracy comparison of Multinomial Logistic Regression and Neural Network across 82 genres.

	Models	Test Accuracy
Multi	Unreg (Base)	26.38
	Unreg w/ (inter.)	29.13
	Unreg w/ (inter. + PCA)	28.61
	Reg (Base)	26.45
	Reg w/ (inter.)	29.62
	Reg w/ (PCA)	28.59
	Reg. w/ (inter. + PCA)	28.59
	Reg. w/ (elastic)	26.48
NN	Unreg	30.48
	L2 Reg	30.94
	L2 + Drop Reg	32.05

Figure 2: Test Accuracy comparison of Multinomial Logistic Regression and Neural Network on the analysis of 77 genres, excluding language-related genres.

	Models	Test Accuracy
Multi	Unreg (Base)	57.18
	Unreg w/ (inter.)	58.14
	Unreg w/ (inter. + PCA)	58.70
	Reg (Base)	57.48
	Reg w/ (inter.)	60.61
	Reg w/ (PCA)	58.60
	Reg. w/ (inter. + PCA)	58.60
	Reg. w/ (elastic)	57.28
NN	Unreg	60.36
	L2 Reg	60.5
	L2 + Dropout Reg	60.79

Figure 3: Performance comparison of Multinomial Logistic Regression and Neural Network on the analysis of 26 genres, excluding the non-dominant genres from the K-mean clustering.

B. Unsupervised Models

*1) **K-means clustering**.* We ran sklearn's k-means algorithm on each version of our dataset, with max number of iterations=10000, random_state=76584, and the following values for k:

Dataset version	Values of k used	Number of genres
0	[10,20,30,40,50,60,70,80,81,82,83]	82
1	[10,20,30,40,50,60,70,76,77,78]	77
2	[10,20,30,40,50,60,70,76,77,78]	77
3	[5,10,15,20,25,26,27,28,29]	26

Notice that the values for k increments in equal steps up to a certain point. That certain point is the number of distinct genres in that dataset version. We do this to find the "knee", which is the k that gives the best balance between low inertia and low number of clusters.

Dataset version	Approximate knee
0	30
1	30
2	30
3	15

To evaluate how the K-means algorithm perform on each dataset version, we used the following metrics:

- **Inertia**: sum of squared Euclidean distances between each data point and its cluster center. This represents the "error" at the end of the clustering process.
- **Silhouette score**: a metric that compares how close the data points are to other members in the same cluster, and how close the data points are to the nearest point in a different cluster. The value ranges from -1 to 1, where a higher value indicates that the data points are well-clustered. Formally, the score is calculated as:

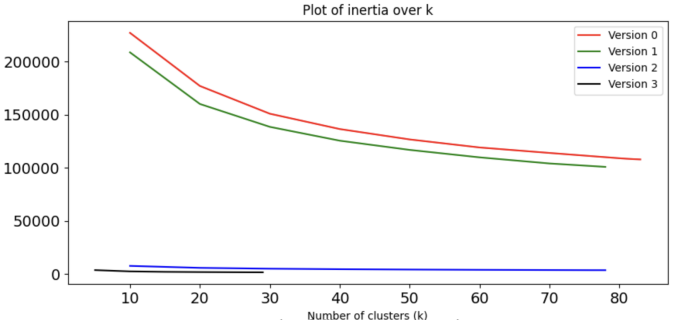
  Silhouette score $= mean(\frac{a(i)-b(i)}{max(a(i),b(i))})$

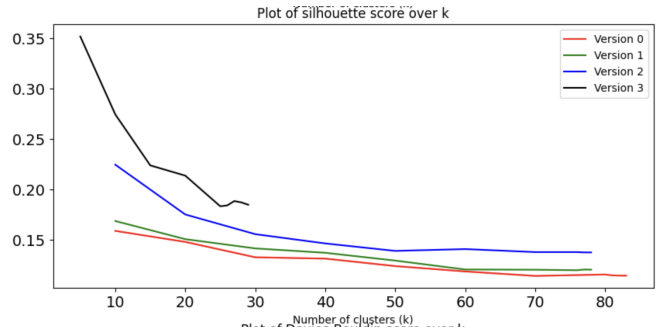
  where a(i) is the average distance from point i to other data points in the same cluster, and b(i) is the is the smallest average distance from i to data points in a different cluster.
- **Davies-Bouldin score**: a metric that also indicates how well-clustered the data points are, but unlike the silhouette score, it finds the average "similarity" between each cluster and its most similar one, where similarity is the ratio of intra-cluster distances to inter-cluster distances. The range is from 0 to positive infinity, where a lower score indicates a well-clustered data. Formally, the score is calculated as:

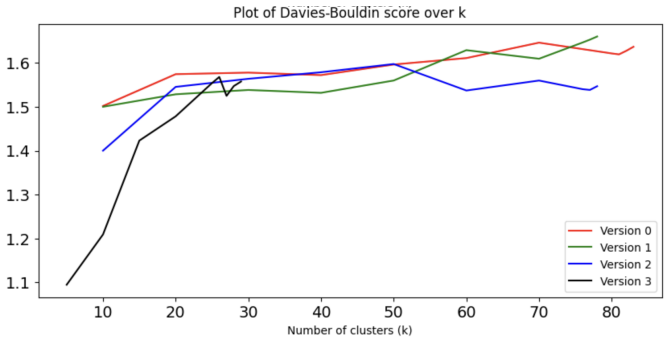
  Davies-Bouldin score $= \frac{1}{k}\Sigma_{i=1}^{k} max_{i \neq j}(\frac{R_{ii}+R_{jj}}{R_{ij}})$

  where where k is the total number of clusters, R_{ii} is the compactness of cluster i, R_{jj} is the compactness of cluster j, and R_{ij} is the distance between cluster i and j.

The plots below display how the evaluation metrics change with the number of clusters (k) used, across four different versions of the dataset.







The plots show that as the number of clusters (k) increases, the inertia (error) decreases, but the data becomes less well-clustered, according to the silhouette score and the Davies-Bouldin score plots.

Regarding the differences between each dataset version, these results show that removing the non-musical genres (dataset 0 to 1) reduces the inertia and improves the clusteredness by a marginal amount, according to the inertia and the silhouette score plot respectively. We think this is due to two reasons: first, the feature values for those genres vary greatly from song-to-song, second, the less number of classes we have in general, the less likely it is to misclassify a song's genre.

Next, using min-max scaling instead of Z-score normalization (dataset 1 to 2) significantly reduces the error and improves the silhouette score by a larger margin. The decrease in error is explained by the fact that min-max scaling scales all feature values to a range from 0 to 1, while Z-score normalization does not. This entails that the squared distances in dataset 2 is much smaller compared to dataset 1. The decrease in silhouette score from dataset 1 to 2 is explained by the fact that in dataset 1, the range of the duration feature is approximately 40, which is much larger than the range of other features. Accordingly, the k-means algorithm gives the duration feature more weight. For example, a difference of 10 in duration is treated as larger distance than a difference of 0.9 in valence, despite the fact that 0.9 covers most of the valence range, even more than 10 covers the duration range. In this picture, the k-means algorithm form clusters to minimize the duration component in the Euclidean distance more than any other features. The decrease in silhouette score from dataset 1 to 2 suggests that giving more weight to duration leads to worse clustering results.

Using only the 26 dominant genres (dataset 2 to 3) significantly improves both the silhouette score and the Davies-Bouldin score. This is because those 26 genres are the most distinct genres in our dataset. The genres are selected from the most dominant genre in each cluster when we ran k-means with k = 30. If those dominant genres end up in different clusters, then their songs are musically different from each other, and the k-means algorithm should form better clusters.

In addition to the evaluation metrics above, we computed the *accuracy* of the k-means algorithm. However, this is difficult to calculate because once the k-means algorithm returns the cluster labels for each data point, we do not know which genre best maps to each cluster label. To resolve this, we use the Hungarian algorithm to assign the best possible genre to each cluster label, where the best possible assignment minimizes the *cost*.

First, we construct a cost matrix of shape (k, k), where each cell (i, j) represents the cost if cluster label i were assigned to genre j. The cost is defined as the number of data points in label i but not in genre j, plus the number in genre j but not in label i. This cost matrix is then fed into the Hungarian algorithm.

Once the Hungarian algorithm returns the best possible assignment, we can compute the accuracy by comparing the clusters generated from k-means to the "true" clustering where one genre maps perfectly to one cluster. This can be done by using sklearn's function for adjusted Rand index, which is a metric that compares the similarity between two clusterings - the one obtained from k-means and the "true" clustering. The index ranges form 1 to -0.5, where a higher value indicates higher similarity between two clusterings. The following table displays the adjusted Rand index for each version of the dataset.

Dataset version	Adjusted Rand Index
0	0.06421
1	0.06922
2	0.07629
3	0.26317

The results indicate that removing non-musical genres (dataset 0 to 1) increases the index by a marginal amount of 0.00501. The same is true for using min-max scaling instead of Z-score normalization (dataset 1 to 2) which increases the index by 0.00707.

Interestingly, removing the number of genres from 77 to 26 dominant genres (dataset 2 to 3) leads to a significant gain in accuracy of 0.18688, indicating that there is some degree of predictability of genre from musical features, but that some genres are too musically similar to distinguish, or that Spotify's musical features do not capture the difference between genres. We think that it is a combination of both - some genres are defined by the nature of the lyrics more than the musical aspects, meaning that two songs could be musically similar, while remaining distinct genres due to their lyrical differences. Meanwhile, Spotify's features could fail to capture musical parameters like timbre, polyphony, and form, which suggests that, musically speaking, there is more to genre than Spotify's musical features.

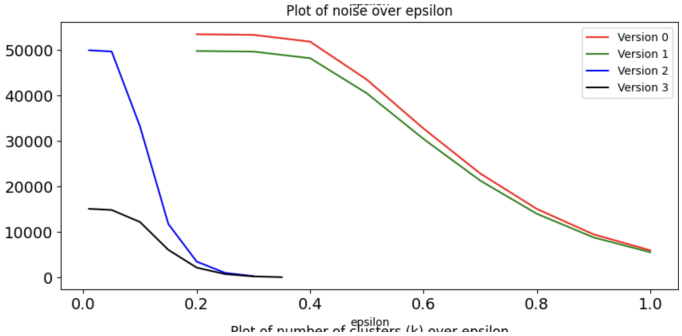
*2) **DBSCAN**.* We ran sklearn's DBSCAN algorithm on each version of our dataset, with min_samples=5, and the following values for epsilon:

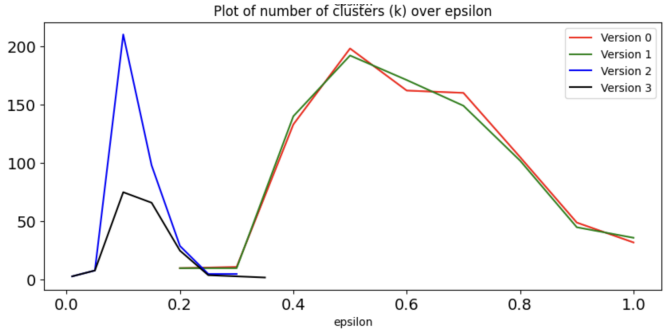
(0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0)

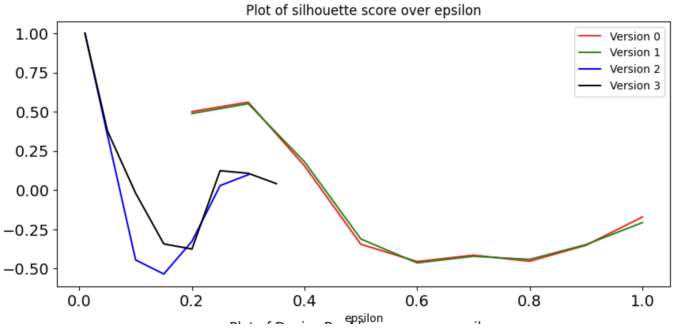
To evaluate how DBSCAN performs on each dataset, we used four different metrics:

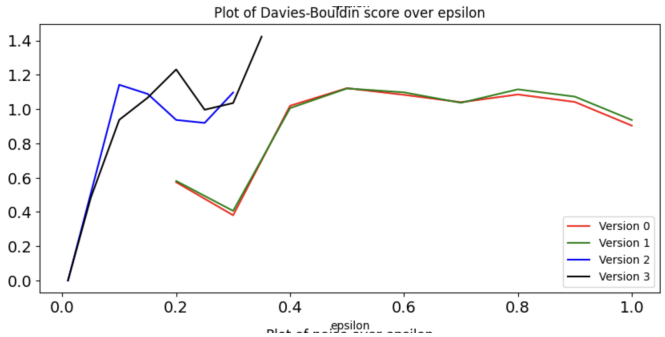
- **Noise**: number of data points that are not assigned to any cluster
- **Silhouette score**: explained in the above section
- **Davies-Bouldin score**: explained in the above section
- **Number of clusters**: the number of clusters that the DBSCAN algorithm is able to find.

The plots below illustrates how the evaluation metrics changes over epsilon.









Noticeably, version 2 and 3 exhibit similar shaped lines compared to version 0 and 1, but the number of epsilons for version 2 and 3 is lower. This is because min-max scaling is used for version 2 and 3, which limits the values to a range of 0 to 1, and the distances between points hold a smaller value that warrants a smaller radius (epsilon). Performance-wise, there is little to no difference between dataset versions 0 and 1, but it is difficult to interpret this particular result. With versions 2 and 3, however, differences emerge in almost all metrics: both the number of clusters and noise for version 3 is lower because there are fewer number of data points compared to version 2, and both the silhouette score and Davies-Bouldin score for version 3 seems better than version 2 but only at lower epsilons.

3) ***Hierarchical clustering.*** We used SciPy's cluster.hierarchy functions to perform hierarchical clustering, with method = single. The datasets we performed the hierarchical clustering on are modified so that each element is the average of all the songs in one genre. We do this for two reasons: one, the runtime is dramatically reduced compared to running with over 50,000 elements; two, this helps us easily find what genres are similar to each other. Dendogram visualization is done via the dendogram function, which is provided in the package.

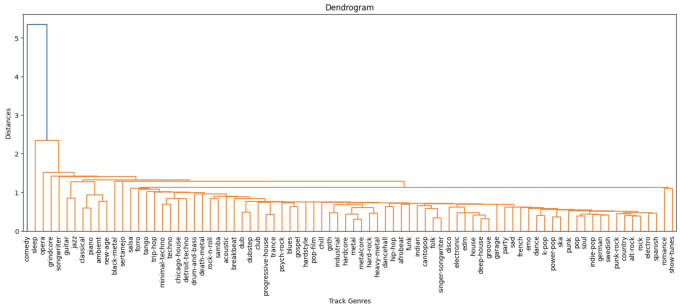


Fig. 1: Dataset version 0

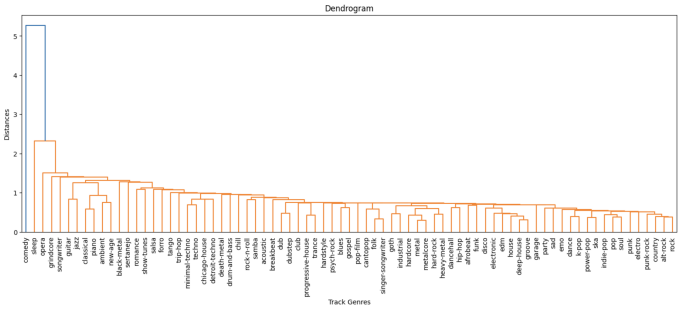


Fig. 2: Dataset version 1

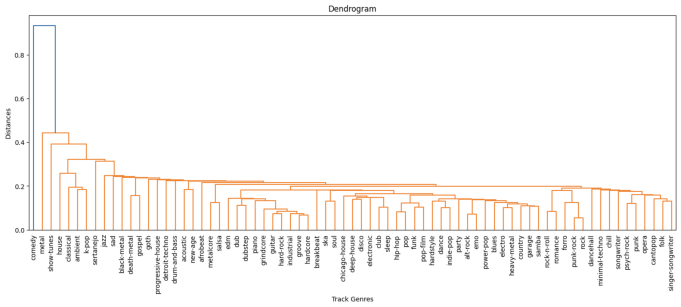


Fig. 3: Dataset version 2

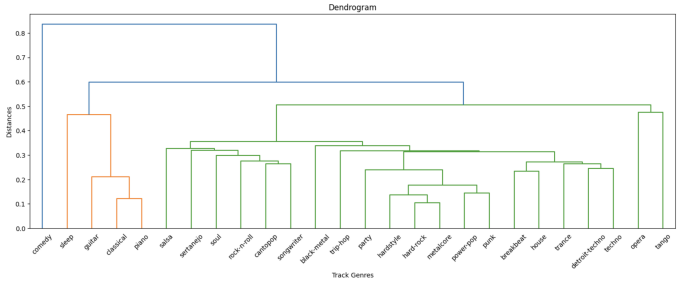


Fig. 4: Dataset version 3

These results suggests that genres that are similar in name (eg.heavy-metal and black-metal) are not necessarily similar in terms of Spotify's musical features. This indicates that there is more to genre than musicality, or that Spotify's music features are insufficient to group similar genres together. Regardless, when working with the 26 dominant genres, the results of the hierarchical clustering seems intuitive. For example, piano is most similar to classical and guitar, while hard-rock is most similar to metalcore.

6. Discussion and Future Work

One noticeable aspect of our project is how our neural networks only slightly perform better than our logistic regression models. It is possible that the relationships in the data are mostly linear and/or that our neural networks are too simple. This also could be due to a lack of feature embedding, which we planned on implementing originally, but had trouble with and did not have enough time to fix. It would also definitely be worth looking into making the neural networks more complex. However, altogether our mediocre performance (a lot better than random chance, but not really usable for any sort of good

accuracy) is most likely due to many genres being hard to distinguish, and this could definitely still impact our models if the above changes are explored.

Future Work for Supervised Learning

Some things general additions that may help our model performance are incorporating lyrics or tags for the "mood" of a song. To achieve better performance, it would also be worth looking at transformer-based models and/or ensemble methods. One other thing that it would be worth exploring is L1 regularization in the neural networks, which can be useful when there are irrelevant features, which there may be for predicting certain genres. Something that would be interesting to investigate is the model performance with genre classification for years besides 2022 and examining if there are any temporal trends in performance.

Future Work for Unsupervised learning

Spotify's musical features do not map well to genre, but that may be because there are so many genres available. Once we narrowed down to 26 most distinct genres, the k-means algorithm is able to produce a clustering with better, but still mediocre, adjusted Rand index score. Using min-max scaling is also crucial for performing a fair clustering, since it's important not to give any one feature undue weight.

For unsupervised learning, one possible direction is to find an evaluation metric that allows a fair comparison with the K-means algorithm. The rand index won't work with DBSCAN because the number of clusters it returns is unpredictable, and it does not make sense to find the accuracy score if the number of returned clusters does not match the number of genres. Perhaps it would be more appropriate to use an evaluation metric that indicates how dominant the dominant genres in each cluster are. Another possible direction is to use the hierarchical clustering results to collapse similar genres together to increase overall accuracy. However, as indicated by the dendograms, the genres that sound similar in name (eg. black-metal and metalcore) are not necessarily similar in terms of musical features. As such, it is unclear what the new labels should be after the collapse. The other direction is to collapse similar-sounding genres together, but this may not increase the overall accuracy.

References

[1] "AI tool characterizes a song's genre and provides insights regarding perception music," ScienceDaily, https://www.sciencedaily.com/releases/2019/08/190812160536.htm#:~:text=08/190812160536.htm-,An%20artificial%20intelligence%20tool%20can%20characterize%20a%20song%27s%20genre%20and,systems%20that%20impact%20human%20emotions

[2] R. Thomas, "An introduction to deep learning for Tabular Data," fast.ai, https://www.fast.ai/posts/2018-04-29-categorical-embeddings.html

[3] C. Guo and F. Berkhahn, "Entity embeddings of categorical variables," arXiv.org, https://arxiv.org/abs/1604.06737

[4] A. Joshi, Spotify 1M Tracks, Kaggle, https://www.kaggle.com/datasets/amitanshjoshi/spotify-1million-tracks

[5] GeeksforGeeks, "How to handle categorical variables in regression," GeeksforGeeks, https://www.geeksforgeeks.org/how-to-handle-categorical-variables-in-regression/

[6] Pascalwhoop, "An overview of categorical input handling for Neural Networks," Medium, https://medium.com/data-science/an-overview-of-categorical-input-handling-for-neural-networks-c172ba552dee

[7] "Mastering batch size in Deep Learning: Unlocking Optimal Model Performance," LUNARTECH, https://www.lunartech.ai/blog/mastering-batch-size-in-deep-learning-unlocking-optimal-model-performance

[8] Comment et al., "DBSCAN clustering in ML - density based clustering," GeeksforGeeks, https://www.geeksforgeeks.org/dbscan-clustering-in-ml-density-based-clustering/

[9] "DBSCAN," scikit, https://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html

[10] GeeksforGeeks, "Clustering metrics in machine learning," GeeksforGeeks, https://www.geeksforgeeks.org/clustering-metrics/

[11] "The assignment problem," HungarianAlgorithm.com - Solve the Assignment Problem, https://www.hungarianalgorithm.com/

[12] D. * S. + R, "Matching clustering solutions using the 'Hungarian method': R-bloggers," R, https://www.r-bloggers.com/2012/11/matching-clustering-solutions-using-the-hungarian-method/.

[13] E. Keany, "The Ultimate Guide for Clustering Mixed Data," Medium, https://medium.com/analytics-vidhya/the-ultimate-guide-for-clustering-mixed-data-1eefa0b4743b

[14] J. M. Ph.D. and E. Kavlakoglu, "What is multicollinearity?," IBM, https://www.ibm.com/think/topics/multicollinearity

[15] "Multinomial logistic regression using SPSS statistics," How to perform a Multinomial Logistic Regression in SPSS Statistics — Laerd Statistics, https://statistics.laerd.com/spss-tutorials/multinomial-logistic-regression-using-spss-statistics.php

[16] R. W. Nahhas, "Introduction to regression methods for public health using R," 6.9 Interactions, https://www.bookdown.org/rwnahhas/RMPH/blr-interaction.html

[17] B. Subhash, "Explainable AI: Saliency maps," Medium, https://medium.com/@bijil.subhash/explainable-ai-saliency-maps-89098e230100

[18] "Lime: Local interpretable model-agnostic explanations," C3 AI, https://c3.ai/glossary/data-science/lime-local-interpretable-model-agnostic-explanations/#:~:text=LIME%2C%20the%20acronym%20for%20local,to%20explain%20each%20individual%20prediction.

[19] "Welcome to the shap documentation," Welcome to the SHAP documentation - SHAP latest documentation, https://shap.readthedocs.io/en/latest/

[20] GeeksforGeeks, "Dropout in neural networks," GeeksforGeeks, https://www.geeksforgeeks.org/dropout-in-neural-networks/

[21] GeeksforGeeks, "Regularization by early stopping," GeeksforGeeks, https://www.geeksforgeeks.org/regularization-by-early-stopping/