1 Data Exploration

We began by listening to an audio file from each of the 10 genres to get a general idea of the audio. We then counted the ground-truth labels to ensure there were 80 audios for each genre in the training data. Later, when plotting Principal Component Analysis (PCA) results of the raw data, we realized there were duplicate tracks, and eventually found 12 identical pairs, shown in Table 1.

File Names of Pair	**Genre**
train006, train179	Hiphop
train020, train785	Metal
train049, train431	Pop
train050, train288	Metal
train060, train437	Metal
train082, train390	Metal
train113, train261	Metal
train116, train165	Pop
train123, train395	Metal
train270, train312	Jazz
train293, train737	Metal
train343, train584	Hiphop

Table 1: File Names and Genres of Identical Audio Pairs

We removed one track from each duplicate pair when performing and plotting some of our PCA results due to errors generated by the matplotlib library when adding color markers to identical points. However, we kept the duplicates in all our models' training data since removing duplicates would result in notably less instances of metal tracks than other genres (a majority, 7/12, of the duplicate tracks are metal), which may have potentially caused the models to prioritize accurate metal classification significantly less than other genres.

Next, we identified the sampling rates of the training data (22050 Hz) and the sample counts and the number of tracks with those counts. The least and greatest sample counts respectively were 660000 and 675808, corresponding to 29.932 and 30.649 seconds of audio. Since all tracks were close to 30 seconds long, we adjusted the durations of all tracks (including testing data) to 30 seconds by appropriately truncating the end of the tracks that were too long and adding zeros to the end of those that were too short). Making the tracks into same-length vectors made them compatible data for our models.

Training and validation data splitting is discussed individually for each model in the Model Selection section.

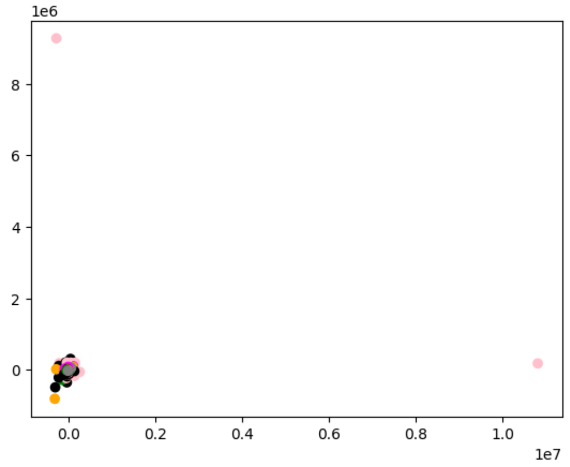


Figure 1: PCA (2 components) Projection of Centered Raw Data

2 Feature Extraction

2.1 Raw Data

We first examined the viability of having the raw data be our features. Specifically, each 30-second track would be a data point, and each sample from the track would be an element in the vector representing the data point. Thus, all the training data would be converted to 800 661500-length vectors. We plotted this converted data (after centering) via PCA (with 2 components) (see Figure 1) to examine whether data of the same genre were clustered together or formed some pattern, which our models might effectively utilize for successful genre classification. Note that in all PCA scatter plots, the color of the points indicate genre.

The PCA algorithm works by finding a projection of the data into some k dimensions which maximizes the variance of the data. The k vectors which are the axes of the projection are the eigenvectors corresponding to the greatest eigenvalues of the data covariance matrix, defined as the transpose of the data matrix times the data matrix. The resulting projection provides lower-dimensional features which best describe the overall spread of the data, which is useful for feature selection and visualization of patterns and trends. In this case, we use PCA to visualize and identify genre-specific patterns and clusters in the data.

After plotting Figure 1, we immediately noticed 2 pink-colored outliers on the plot, and re-plotted after removing them, to better visualize the data, shown in Figure 2.

Even after removing outliers, we still noticed a lack of clustering or patterns in the data, except that some of the pink and black data points appeared to be slightly distant from the center of the main cluster. However, we observed numerous pink and black points that were within the cluster center, and could not identify patterns relating to other genres, and thus were pessimistic about the separability of genres that the raw data features would provide.

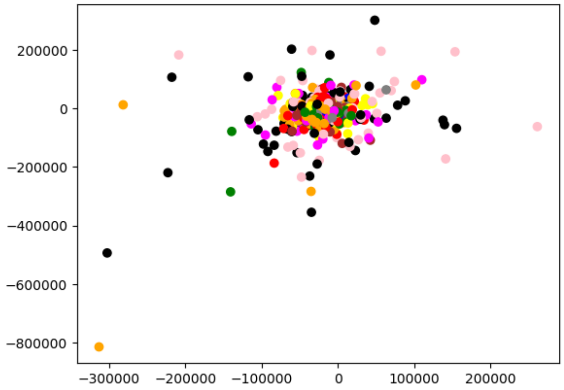


Figure 2: Plot after Removing Outliers from PCA (2 components) Projection of Centered Raw Data

Suspecting that the 2 outliers from earlier contributed significantly to the variability of the PCA, and that therefore the PCA results didn't maximize the projection variability of the non-outlier data, we performed and plotted PCA on the non-outlier raw data (all data except those two outliers), shown in Figure 3. Observing yet 2 more outliers, we removed them from the plot to obtain Figure 4.

From the Figure 4 plot, we observed the same pattern of many of the pink and black genre data points being distant from the main cluster, however there was no apparent pattern to be observed for the other genres, suggesting that a model trained on these features may be unable to obtain sufficient useful information to classify numerous genres. Due to this apparent insufficiency, we examined further feature options.

2.2 3-Second Audio Clips

We next examined the viability of splitting each 30-second clip into 10 3-second audio clips, and using the raw data as our features, where each data point is a 3-second clip. This would yield 8000 66150-length vectors as training data. When listening to a few 3-second segments, we felt that we could usually hear distinctive tunes corresponding to certain genres, and that therefore 3-second segments would contain sufficient information to be classified into a genre by a model. Further, dividing the 30-second tracks accordingly would increase our training data 10-fold, which would likely reduce over-fitting and improve the generalization of our models to test data.

We plotted a PCA (2 components) projection (see Figure 5) of the centered 3-second data to examine whether it contained clusters or patterns for each genre, which would suggest that our models may be effective at classifying genres using such features.

We observed that similar to the PCA plots of the raw data (e.g. Figure 4), the black and pink data points were distinctly far from the center of the main cluster, and in fact seem to be far from the cluster in 2 differing dimensions, which is an improvement over the PCA plots of the raw

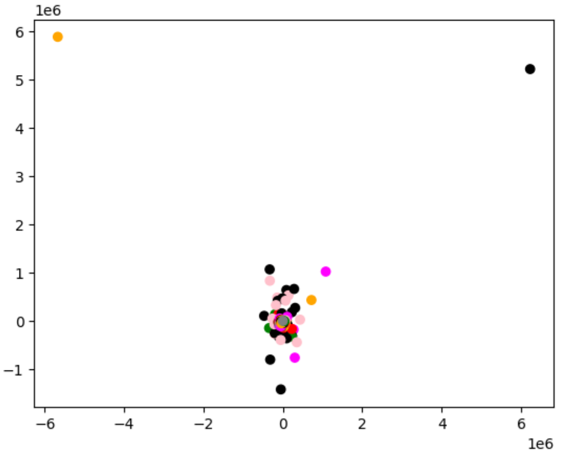


Figure 3: PCA (2 components) Projection of Centered Non-Outlier Raw Data

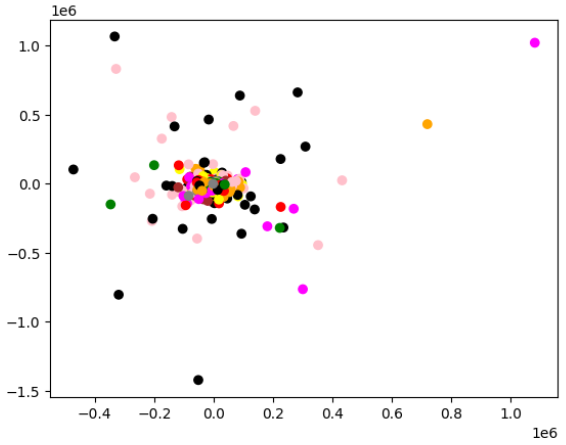


Figure 4: Plot after Removing Outliers from PCA (2 components) Projection of Centered Non-Outlier Raw Data

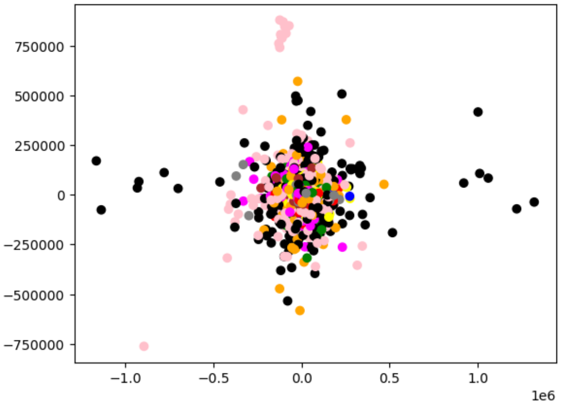


Figure 5: PCA (2 components) Projection of Centered Raw 3-Second Clips

data. However, many pink and black data points are in the center of the cluster, and no pattern is discernible for the other colors (genres). Therefore, we were unsatisfied with the potential of these features, and examined another set of features.

2.3 MFCCs of 5-Second Audio Clips

We were interested in examining Mel-Frequency Cepstral Coefficients (MFCCs) as our features, as they are often used as features when working with audio signal classification. Also, after observing that the PCA plot of the 3-second raw data (Figure 5) did not appear to extract much distinguishing information for each genre, we decided to examine 5-second clips of the audio, which presumably would contain more information distinct to the audio's genre, potentially improving genre separability. Therefore, we decided to examine MFCCs of 5-second audio clips.

The MFCC computation for an audio time signal is computed by performing a Fourier transform of it to obtain information about different frequencies. Then, the logarithm of the absolute value of the transform is computed, which serves to provide more information about different frequencies and information about the harmonic components of the original signal, which are reflected in periodic structures in this log spectrum. Next, this spectrum is transformed according to the mel scale, which serves to scale frequencies such that the difference between frequencies correspond more to the actual perceptual difference heard by human listeners, which is more relevant to genre classification, which is a human-listener-determined metric. Then, the discrete cosine transform is computed for this mel-scaled log power spectrum, which results in n MFCCs for each time step (where n is generally set to an integer between 13 or 20 for audio tasks). Each MFCC provides information about the spectral envelope (i.e. the shape) of the transformed power spectrum (i.e. the transformed spectrum of the original signal's frequencies) at a certain time step. Since in practice, MFCCs used as model features are averaged across time, we averaged ours over time as well, resulting in 20 MFCCs. We chose $n = 20$ because it was the default for the MFCC library (librosa) we used, and we reasoned that the library would choose an appropriate default value that generally provides good results for MFCC-related tasks.

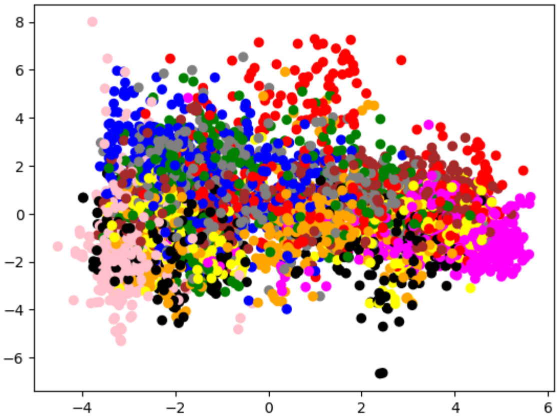


Figure 6: PCA (2 components) Projection of Centered MFCCs of 5-Second Clips

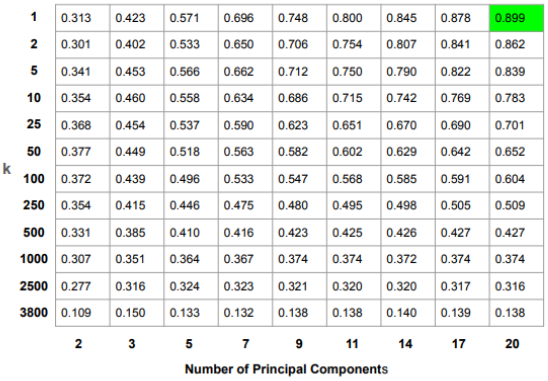
After computing the MFCCs on the 5-second clips, we had 4800 20-length vectors as our training data, where each vector contains the 20 averaged-across-time MFCCs for a 5-second clip. To examine whether the features yielded genre-specific clusters or patterns, we plotted the PCA (2 components) projection of these centered MFCC features, shown in Figure 6.

We were happy with the results of this PCA plot because the different genres (colors) appeared to be clustered in different sections of the plot. For example, the purple points are clustered on the right, the blue points are clustered in the top left, the orange points are clustered in the center, and the light pink points are clustered in the bottom right. This distinct clustering of different genres suggested that the MFCC features provided significant separability of the genres into different clusters, which would likely be reflected more in higher dimensions, as the clusters wouldn't be as cramped together as in a 2-dimensional projection. This genre clustering would be amenable for our models, likely enabling successful classification of different genres, so we used these MFCC features in our models.

3 Model Implementation

3.1 K-Nearest Neighbors

The next model we tried was a k-nearest neighbors model on the PCA projection of the centered MFCC features of the 5-second clips. Upon observing apparent genre clustering in the PCA (2 components) plot of the MFCC features (see Figure 6), we thought a k-nearest neighbors model would be the most intuitive way to take advantage of this clustering, since a data point of a certain genre would be close to the cluster of its genre, so its nearest neighbors would presumably be clips from its genre, leading to a correct classification. Further, we expected the genre-specific clustering to improve as the number of projection dimensions increased, since the clusters may be less crammed together than they were in 2-dimensions (see Figure 6). So, we wanted to try increasing the PCA dimensions to see if the genres would be more separable.



k	2	3	5	7	9	11	14	17	20
1	0.313	0.423	0.571	0.696	0.748	0.800	0.845	0.878	0.899
2	0.301	0.402	0.533	0.650	0.706	0.754	0.807	0.841	0.862
5	0.341	0.453	0.566	0.662	0.712	0.750	0.790	0.822	0.839
10	0.354	0.460	0.558	0.634	0.686	0.715	0.742	0.769	0.783
25	0.368	0.454	0.537	0.590	0.623	0.651	0.670	0.690	0.701
50	0.377	0.449	0.518	0.563	0.582	0.602	0.629	0.642	0.652
100	0.372	0.439	0.496	0.533	0.547	0.568	0.585	0.591	0.604
250	0.354	0.415	0.446	0.475	0.480	0.495	0.498	0.505	0.509
500	0.331	0.385	0.410	0.416	0.423	0.425	0.426	0.427	0.427
1000	0.307	0.351	0.364	0.367	0.374	0.374	0.372	0.374	0.374
2500	0.277	0.316	0.324	0.323	0.321	0.320	0.320	0.317	0.316
3800	0.109	0.150	0.133	0.132	0.138	0.138	0.140	0.139	0.138

Number of Principal Components

Figure 7: Table of Validation Accuracies of Hyperparameter Combinations for PCA-projected kNN Model

K-nearest neighbors works by computing the k closest data points (via 2-norm distance) and outputs the classification that is the most common category of the k points.

We employed 5-fold cross validation on our training data, a validation algorithm which splits the training data into 5 groups, and trains and validates it 5 times, with a different group being the validation group each time. The overall training and validation scores are the average of those obtained in each of the 5 iterations. We did this to maximize the truth of our validation score (since multiple validation sets are used) and minimize over-fitting (since all the training data is used for training, not a subset).

The hyperparameters we tuned were the number of nearest neighbors to evaluate (k) and the number of components of the PCA projection. After trying numerous combinations (see Figure 17), we the one which produced the greatest validation accuracy (0.899) was $k = 1$ with 20 principal components, which is equivalent to $k = 1$ on the original MFCC 5-second data. In general, increasing number of components and decreasing number of nearest neighbors increased validation score, suggesting that very small local clusters (but not big clusters) often had the same genre, which only became more true in higher dimensions.

Although this model had strong validation accuracy on the 5-second data, when applied to the Kaggle public and private test data (for each 30-second clip, it outputs the mode predicted category of its 6 5-second clips) with the optimal hyperparameters, it scored 0.32 and 0.46 respectively, which is over twice as bad as it performed on the validation data. Therefore, although this model performs well on small data, it was unable to generalize well when combining predictions for the actual 30-second test data. Furthermore, this model is likely slower in predicting than other models (except perhaps the Random Forest model), since it would need to compute l2 distances over all 4800 20-length vectors for a prediction of a single 5-second clip. On the other hand, our other models simply require a single feed-forward traversal of the model, not an explicit comparison with all of the training data.

3.2 Random Forest

Our final model was a random forest classifier on the MFCC features of the 5-second clips. We wanted to use a random forest model since tree-based models like random forests and XGBoost are often very accurate in practice, so we thought it might be worthwhile to try them. Unlike the k-nearest neighbors model, we didn't experiment with using the PCA projected features of the MFCC features since we already found them to have worse performance in the kNN model. Further, random forests already have numerous potential hyperparameters to tune, so we wanted to avoid the tuning time being extremely long from having to tune many hyperparameters.

The random forest algorithm generates a fixed number of decision trees.

Each decision tree is created by randomly sampling the training data (in order to reduce overfitting) and finding the optimal feature value such that, after dividing the data sample into 2 groups based on whether they have a less or greater value in that feature, the 2 groups have maximally different categories (e.g. genres). It continues this division process with each of these 2 groups (which are 2 "branches") until further divisions do not significantly increase the difference in categories or the maximum tree depth is reached, which completes the tree's creation.

The tree predicts a category for a data point by starting at the tree top and moving left and right down its branches based on which sides of the feature divisions the point falls under. When the point gets to the tree bottom, it is probabilistically classified based on the categories of the training data points which ended there in the creation stage.

In the random forest, the classification probabilities of all the trees are averaged to get the overall classification probabilities, the largest of which is the overall prediction of the model.

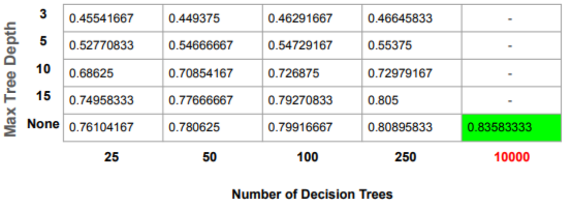
The two hyperparameters we chose to vary for this model were the number of decision trees and the maximal depth of the tree. We did not want to tune more than two because tuning would take extremely long otherwise.

We chose number of decision trees because increasing number of trees generally only improves accuracy (since there are more trees working together to predict) and mainly only loses time and space efficiency. So increasing the number of trees as much as possible would be beneficial for our model.

We chose maximal tree depth because it controls over-fitting and under-fitting (i.e. a deeper tree may overfit and a shallow one may underfit), and we wanted a hyperparameter which controls this crucial aspect of the model (which is also the biggest aspect controlled by a hyperparameter in other models, e.g. l2 regularization parameter, number of nearest neighbors, etc.).

After performing 5-fold cross validation on numerous combinations of values for number of trees and maximal depth (see Figure 18), we noticed a general trend of greater validation score as both hyperparameters increased. Since the maximal depth was "None", we subsequently tried to increase number of trees as much as possible, achieving 0.836 validation accuracy.

We tested this final tuned model on the public and private Kaggle test data, but only achieved 0.47 and 0.36 accuracy respectively. This quality of performance is similar to the kNN model (see Section 4.3). The random forest model appeared to again perform significantly better when applied to 5-second clips than the actual 30-second data, possibly because some of the 5-second validation clips were from the same 30-second tracks as the 5-second training clips, causing data contamination and causing a misleading validation score. We forgot to realize this until writing the report, and



Max Tree Depth	25	50	100	250	10000
3	0.45541667	0.449375	0.46291667	0.46645833	-
5	0.52770833	0.54666667	0.54729167	0.55375	-
10	0.68625	0.70854167	0.726875	0.72979167	-
15	0.74958333	0.77666667	0.79270833	0.805	-
None	0.76104167	0.780625	0.79916667	0.80895833	0.83583333

Number of Decision Trees

Figure 8: Table of Validation Accuracies of Hyperparameter Combinations for Random Forest Model

this should have been avoided.

Some comparisons between this and other models are that the random forest is likely slower than all other models since it has to traverse 10,000 decision trees for each 5-second clip. However, it is likely more interpretable than neural networks like the LSTM and CNN, since the most common MFCC values at which the trees lead to certain genres can be computed, which would reveal what MFCC values the model explicitly looks for when identifying certain genres.

4 References

2. Mel-Frequency Cepstral Coefficients Explained Easily
3. Intuitive understanding of MFCCs
4. MEL
5. MFCC Technique for Speech Recognition
6. Random Forests
7. A Beginner's Guide to Random Forest Hyperparameter Tuning