Marc Evans (GTID: 903744769)
Naveen Ram (GTID: 903127600)
Tyler Lang (GTID: 903737252)

rAId.io

ABSTRACT

This report outlines regression and classification-based approaches to music analysis, both on characteristic and raw sound byte data. Our goal with this analysis is to provide novice composers the tools needed to create more successful music via insights from supervised machine learning models. To accomplish this, we approached this problem from both a genre classification and popularity classification standpoint. Overall, we found that certain models had relatively high predictive power, however all models were not able to achieve accuracy above 80%. Each model had unique results in terms of recall vs. precision, overall accuracy, and patterns in misclassification. In this project's current state, while not being able to achieve high predictive accuracy, the optimal models still can help guide novice composers through their song creation and testing process.

INTRODUCTION

Music – originating as an auditory artform in which instruments or vocals must be manipulated physically by a professional to create soundwaves that are composed in such a way as to sound pleasant or meaningful to the ear – has now become a matter of technology. Computers have grown increasingly influential in the music industry, with some forms of music, such as EDM, having become completely defined by synthetic, computer-generated sounds and mixing softwares, requiring only a laptop to become an entire band in one machine. The finesse required to compose an entire musical piece has become something more available to common society with less barriers to entry. Yet, even with composition softwares, synthetic instruments, and automated track building, the creation of a truly successful song requires industry knowledge and pure creativity, which is still one of the biggest barriers to entry. Music, among other types of art, is an industry with one of the biggest dropout rates for new entrepreneurs, in that thousands of novice music composers spring up daily, trying to make a profit with the artform, only to find that their creations receive little to no interest by the public at large.

For this reason, our goal for this paper is to utilize machine learning to identify what truly makes a song successful, which can then be taught to your novice composer to streamline his creation and testing process. Normally, a novice composer would have to create a song, release it, promote it, wait weeks or even months to see results, all with the potential that their song does not become a success. With the right machine learning modeling, however, one could, within a few hours, use data to learn what makes a song popular, create a song based on this information, and finally test their song with trained models geared toward testing on classification of potential popularity. Our hope is that, with the right data and modeling, such a tool could break even more barriers, allowing novice composers to become niche artists by giving them the tools to build and evaluate the popularity of their song and see how well it fits into a desired genre. Thus, with this project, we hope to learn about the data structures and features that can define music and sound, use modeling on both singular features and raw sound data to classify both genre and popularity, and discover more on what truly makes a song, a song.

DATA

Data Sources

Before searching for data, we defined the types of data needed to align with our final foal. We decided on two types of data features - single value features related to a song's characteristics, and raw sound data from 30-second clips of common songs. The characteristics dataset consisted of features such as tempo, danceability, energy, loudness, mode, speechiness, acousticness, instrumentalness, liveliness, valence, explicit flag, and artist (which are all defined in the appendix). Data points in this list of features included both ranked, categorical, and scaled values, which we then standardized during the scrubbing phase of data preparation. Our second dataset involved raw audio data in the form of 30-second clips, which are freely available

through the Napster API and will be used solely for research purposes. All insights reported will be on a top-level basis and not linked to any individual song or artist. The goal with these two subsets was first to see how well raw data performs in machine learning vs extracted features, as well as to explore, with the second dataset, what features can be extracted from raw, time-series sound data. This data will be discussed in more detail below.

Then, in terms of both sets of data, we wanted two subsets each: One aimed at being used to predict genre, which was balanced across 5 different genres – Blues, Hip Hop, EDM, Pop, and Metal – and the other aimed at being used to predict popularity. The subset for genre prediction was stratified over different years of release and popularity, controlling for songs over the mean in terms of popularity ranking (a score which is provided directly by Spotify's Python API – more below); while the subset for popularity classification was restricted to being released in the last 5 years. Thus, in total, we had 4 unique datasets, two per analysis goal (predicting popularity vs genre), each with its own unique methodology for analysis.

Upon having our data requirements well-defined, we decided to use two main APIs to gather our data: Spotify's python API (Spotipy), as well as Napster's python API. While there are potentially many datasets available for similar music analyses, we did not want to use a pre-gathered dataset that is perhaps overly hand-picked and cleaned to optimize for success (such as the infamous GTZAN music dataset), and had specific requirements for our analysis at hand, which we could not guarantee would be met by these datasets found pre-packaged online.

Data Collection

Our data collection pipeline was the following: first, using Spotipy to gather thousands of song titles, years released, and artists via its querying system, then stratifying and balancing the samples as mentioned above, and finally creating a list of 2000 to 3000 potential songs to pull data from. Specifically, we ran the search API from Spotipy and pulled 1000 songs each year from 2017 to 2021, based on the criteria mentioned above. As for the dataset to classify popularity, we then had to normalize the popularity scale from 0 to 100, given the unbalanced distribution of popularities, and that the popularity score given by Spotipy is based on popularity at current date. Finally, we cleaned the data by removing remasters of old songs, as this threatens our year-based stratification and filtering. To do so, we used regex code to search for songs with titles including words similar to "remaster" or "remix" as well as cross-referencing the output release date.

Then, using the song titles, we gathered the URLs from Napster that link to 30-second mp3 demos, which could be directly loaded into Python without the need to download any mp3s, using python packages urllib and pydub. The data from these MP3s is directly loaded into Python as a 1xN numpy array, which we truncated to 1x661500 for consistency. These arrays were exact numerical representations of the frequencies of the song over each sample (a sample is 1/22050th of a second in this case, as the MP3s come in with a sampling rate of 22050).

Finally, Spotipy's API conveniently comes with many characteristic features per song, and thus, to capture our other dataset, we simply scraped Spotify's API song by song, using each song's unique identifier to gather the information needed, as listed above.

Data Pre-processing

As for the raw sound data, an unexpected limitation was the size of the data being used. While knowing the size of the arrays, we had insufficient knowledge in truly how these massive arrays would clog a standard laptop's CPU, and lead to 20, even 30 GBs of storage use when using our original dataset of 4800 songs, each with 30 seconds of audio.

After spending much time creating functions and pipelines to load, read, manipulate, and finally remove from local computer memory to be able to manage the size, we eventually had to downsize our data to be able to proceed with further analysis. While our original raw dataset consisted of 4800 songs, we eventually had to downsize to a dataset of 250 songs for predicting genre, and another 250 prepared for predicting popularity. At least in the case of predicting genre, we found this small, wide dataset to be somewhat insufficient in terms of running any complex modeling on, and thus we had to find ways to grow our dataset without running into technological limitations of computation time, storage, and memory. Two ways to do so that are commonly seen in modern music analysis processes include both segmenting and augmenting.

Segmenting is a process in which a song is cut into equal parts, usually at specific points in the audio, in which the principal features of the song to be used in modeling are preserved in each clip. Thus, we found via research and our own experimentation that 5 second clips were the best balance in terms of boosting our dataset size while still preserving enough song qualities so as to contribute to meaningful feature extraction and model training.

Augmenting is a process in which single arrays are slightly manipulated so as to preserve the original array's main features, but adding enough variation so as to add value to a machine learning model by increasing the amount of data available for a model to train on and learn from, especially when it comes to feature recognition models, such as CNNs. In terms of audio data, augmentation techniques used to augment our training and validation datasets included time stretching, pitch shifting, gaussian noise, loudness changes, song reversal (for CNNs), and time masks, all of which are explained in detail in the appendix.

Feature Extraction

Spotify Data Features

The spotify dataset used for genre classification consists of 4,946 individual tracks by 1,412 unique artists. This dataset consists of 13 features, of which 10 are numerical (popularity, danceability, energy, loudness, speechiness, acousticness, instrumentalness, liveliness, valence and tempo) and 3 are categorical (explicit, key, mode). Therefore, the first step was to One-Hot-Encode our categorical features, of which explicit and mode are binary variables and

so we can remove one label at random to avoid collinearity. This results in a final feature set of 24 and our label for genre (multi-classification of Blues, EDM, Metal and Pop).

Since our dataset is small, and there wasn't much multicollinearity, we chose to use all features for modeling and so the next step was to split our data into a training (80%) and test (20%) set using a random shuffle.

Librosa Data Features

For our raw audio datasets, the next step in our pipeline was extracting features, which both served to provide more meaningful and less noisy signals to the models, as well as further reducing dimensionality into more compact arrays. To do so, we used the infamous libROSA package in Python, which is a package that runs automated feature extraction from a sound byte array. LibROSA has a wealth of possible feature extraction functions, all of which we heavily researched to choose the features most applicable to our analyses. These five main features include the Mel Spectrogram, the MFCC, the Zero Crossing Rate, the Fourier Tempogram, and the Chroma STFT, which are all explained in the appendix.

Specifically for our popularity array, we performed forward selection on each librosa feature. To do so, we used a standard Random Forest Classification model to compare accuracy on each iteration of forward selection, adding features as needed when resulting in improved accuracy. Features were first extracted and then flattened, resulting in an nxm matrix which was passed into the model, with n being the number of songs, and m being the sum of the flattened matrices for each included feature. We started forward selection with the MFCC due to its highest initial accuracy. Listed below are the best combinations found and their respective accuracies:

Features	**Accuracy**
MFCC, Zero-crossing-rate, fourier-tempogram	80%
MFCC, RMS	78.3%
MFCC, Poly_Features	78.3%
MFCC	76.6%

From this, we confirmed that the best features for popularity classification are, in fact, the MFCC, zero-crossing rate, and the Fourier tempogram. We then proceeded with analysis using only these feature sets. For genre classification, we also included the Mel Spectrogram and Chroma STFT for testing with deep learning models, especially a CNN, due to their highly visual natures.

Finally, when all data extraction was finished, data was stored in a compressed format with the package "Joblib," which stores big, but quick-to-load files on the local laptop. Due to time

limitations on loading, augmenting, and extracting features of the raw data, as a group, we were forced to work locally, and thus each member was in charge of one unique dataset.

For these extracted features, a variety of methods were used across different models, including flattening various features and stacking them horizontally, as well as passing in 3D arrays, or lists of 2D arrays, to models that could handle such a data structure, like keras models and a CatBoost model.

GENRE CLASSIFICATION

With our data fully prepared for analysis, our next step was exploration. As our main mission is to demystify music composition by providing a framework for one to be able to judge the popularity of a potential composition, genre classification has served more as an exploratory analysis to better understand our data, how various machine learning algorithms interpet or discern classifying patterns in the data, and understanding what “genre” means, in terms of sound, pitch, frequencies, and tempo. This task was performed separately with two datasets, one being the Spotify features, and the other being the raw data with all the augmentations and feature extractions discussed above.

Analysis of Spotify Features

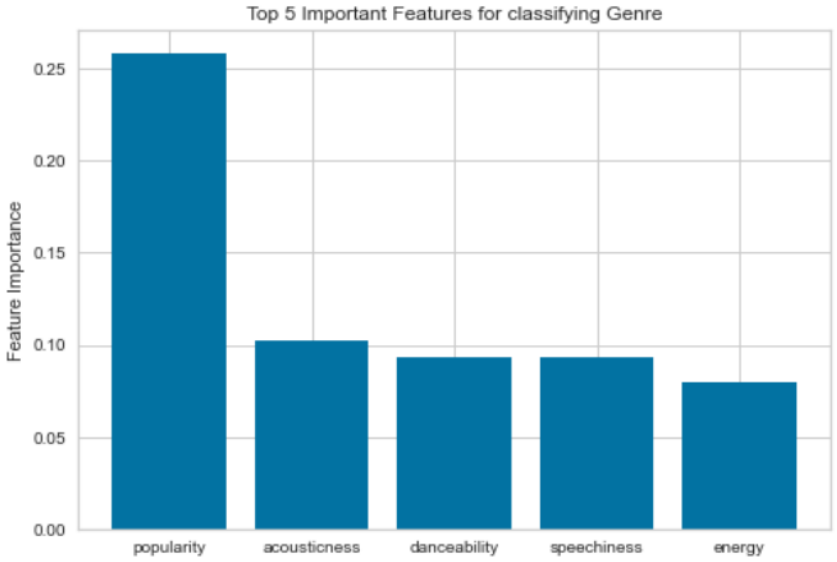
Methodology

Since the spotify dataset did not require much pre-processing other than normalizing and scaling the training data. There was no missing data and since the dataset came with a small subset of features, we could model straight away using all of the 24 available features following categorical feature encoding.

Firstly, we ran a 10-fold cross-validation on the training set to hypertune multiple classification models including: Random Forest, Logistic Regression, Ridge Classifier, Naive Bayes, and Gradient Boosting. From this, Random Forest performed the best on the cross-validation step with 100 trees and an average accuracy of 79.81%:

Model	**Accuracy**	**Recall**	**Precision**	**F1**
Random Forest	0.7981	0.8023	0.7986	0.7970
Gradient Boosting	0.7960	0.7995	0.7987	0.7959
Logistic Regression	0.7078	0.7094	0.7086	0.7059
Ridge Classifier	0.6793	0.6837	0.6802	0.6720
Naive Bayes	0.5963	0.8568	0.6022	0.5934

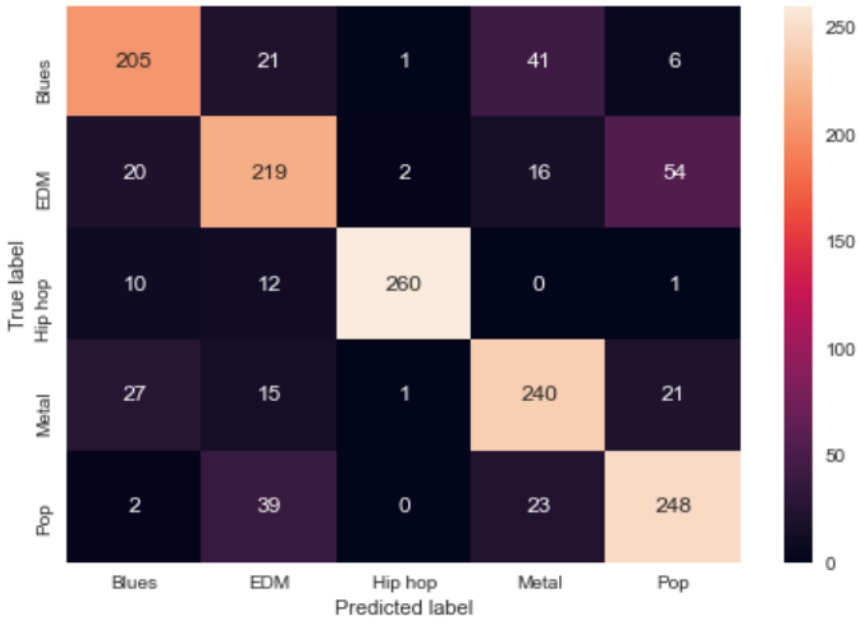
The top 5 most important features from our best performing model (Random Forest) can be seen in the below bar plot:



The key and mode (minor vs. major) features had very little effect, which seems surprising, given that we would expect these five genres are often played in opposing scales.

Results

Running the final prediction using the best model from the cross-validation step, we get an accuracy of 78.98% on the test set with the following confusion matrix:



This shows us that Hip hop was the easiest to classify. Also, Blues and Metal, and EDM and Pop were the most miss-classified pairs of predictions. It might be expected that in recent years, EDM and Pop would be miss-classified as there has been a shift in popularity of EDM/dance music. However, more unexpected is the miss-classification of Blues and Metal.

We can also see a higher miss-classification between Metal and Pop, showing that both of these genres can be harder to classify. We expect this miss-classification from Pop, as it can vary widely based on region and year of release.

Takeaway

From this analysis, the three main 'controllable' factors that are found to contribute most in genre classification are acousticness, speechiness, and danceability. Low acousticness was one of the most defining factors for identifying EDM, whereas higher acousticness helped classify Blues. High speechiness was correlated with Hip Hop, whereas low speechiness was more aligned with EDM. Finally, high danceability was correlated with Pop and EDM, whereas low danceability was correlated with Blues.

Thus, for a novice composer trying to break into a particular genre, ensuring specific levels of acousticness, speechiness, and danceability might help define a song in that genre. For acousticness, this can be as simple as adding/removing or analyzing the existence of electronic instruments within a track. For speechiness, any artist can include more singing (pop), decrease lyrics-to-beat ratio (EDM) and increase speed and frequency of spoken-word (Hip Hop) to change or break into a genre. Finally, for danceability, one can add more beat strength (e.g. bass) and repeat musical patterns to more closely align with a given genre.

Finally, as the most important factor, 'popularity' is harder to account for when analyzing genre for a novice musician, but it is unsurprising that this factor helped identify Pop and EDM, but not differentiate them.

Analysis of Sound-Byte Data

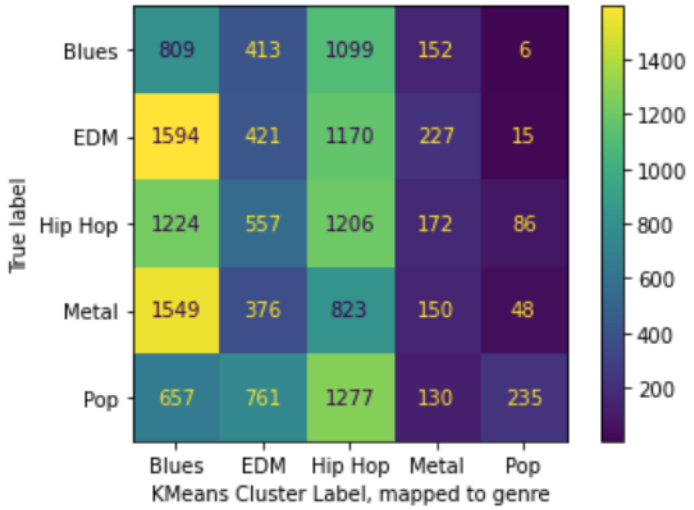
Methodology

For this dataset, the true process of extracting the final dataset involved first splitting the data into train, validation, and test sets, then segmenting each song into 5-second clips, and finally augmenting these clips using audiomentation package to grow separately the train and validation sets for model training and tuning. The test set, of course, is not augmented, as it would not accurately reflect the true accuracy of the model when passed in a real song. Finally, separate datasets are created, one per feature, for the same set of training, validation, and test sets.

Then, as this phase served to be exploratory in nature, we tried a variety of modeling techniques in hopes of learning more about the structure of the data, including unsupervised

clustering models and supervised models. Dimensionality reduction methods were also applied to flattened feature matrices, but, in all cases, seemed to degrade model performance, which is understandable due to the inherently time-based nature of the data.

To explore the separability of the data, clustering methods were tried, but had sub-standard results to be used in any useful way to support further analysis. Below is a confusion matrix comparing the true outcome variable to the label provided by a tuned KMeans model, with optimal performance at k=2 (very noisy decision boundary):



As can be seen, clustering does not seem to have a preference in terms of clustering certain genres together. And the way in which the clusters were formed is not practical to analyze, given the format of the data. Furthermore, when run with spectral clustering, all points were simply placed in cluster-0, with 10-20 outliers in clusters 1-4, meaning that clustering models have a difficult time truly finding any meaningful separation of the data in the data's M-dimensional space.

Due to the subtle differences in song sound byte data across genres, clustering did not prove to be so successful. Thus, supervised classification methods became the prime focus of this dataset. With so many variables of analysis to consider - which feature or combination of features to feed into a model, augmenting the data vs not augmenting, the various classification models that exist, and little guidance from unsupervised modeling on what type of decision boundary might perform best to help choose initial models – a variety of base models were attempted. First, to limit model selection, MFCCs were passed into the following models to judge initial performance and decide on a general type of model to explore further. The following is a table of accuracies by these base models with the MFCC data. MFCC was chosen as an initial feature due to its compressed size, and more consistent accuracy. Later, among the chosen model family, other features were attempted as well, which will be discussed.

Model	Accuracy
Linear SVM	36.59%
Logistic Regression	44.95%
Naive Bayes	50.18%
Kernel SVM (rbf)	53.31%
Random Forest	47.04%
KNN	37.63%
Catboost	49.37%
Light Gradient Boosting	51.34%

Overall, of the non-deep learning algorithms used, Kernel SVM performed the best, followed by Light Gradient Boosting, and surprisingly Naive Bayes.

The following are the confusion matrices of the Naive Bayes and Kernel SVM models:

```
[33,  9,  3,  8,  1] [23,  2,  1, 22,  6]
[ 6, 31,  3,  4, 20] [ 6, 34,  8,  8,  8]
[ 5, 12, 16, 11,  7] [ 1,  8, 35,  5,  2]
[ 9,  1,  1, 40,  4] [ 9,  4,  1, 40,  1]
[ 5, 21,  2,  2, 33] [ 5, 27,  9, 10, 12]
```

Order: Blues, EDM, Hip Hop, Metal, Pop

When looking at the results of these two models, overall both models have weak predictive power on average, but with some genres having high precision but really low recall, or vice-versa (For example, Metal). The inaccuracies are fairly scattered, and thus it is hard to detect any patterns in misclassification. In deep learning models, the biggest issue is between Pop and EDM, in which models are completely unable to discern the two apart, but achieve accuracies higher than 80% on the other three genres. This also shows to be an issue with non-deep models as well. With the final model below, this will be discussed in more depth.

One thing to note is that, very much as expected, models with linear decision boundaries did the worst, followed by KNN. Linear decision boundaries were expected to do very poorly, as the data for music is in no way linear and very intricate. Specifically, this data is very highly susceptible to overfitting, especially due to segmentation and augmentation methods used to grow the dataset. Thus, it comes as little surprise that this dataset is not apt for KNN, which is one of the most likely types of models to overfit extremely, especially with low values of K, which is necessary for a dataset like this with very delicate boundaries within the data cloud across the classes. Not a single base model trained with the MFCC was able to perform 55% or higher, and thus all models are fairly weak, with inaccuracies in classification fairly scattered. But, with a

random guess being 25% accuracy, all models do have predictive power, and thus perhaps that is why Naive Bayes, which is an overly general model, performed just as well as the top models. Boosting models were initially expected to perform even better, yet as can be seen, they did not outperform.

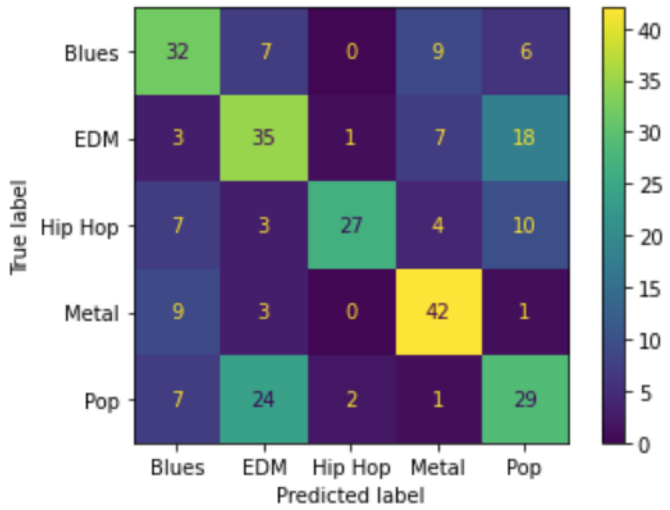
Final Model Performance

With the RBF-kernel model chosen as the final model, feature selection and processing needed to be tested.

Using the 5 features previously mentioned, models were tuned and run with each feature, as well as combinations of features. Results using dimensionality reduction are not detailed, as all tests comparing reduced dimensionality to the full dataset performed worse when reducing dimensionality to 20,60,100, and 1000. Too much information is simply lost with such an intricate time-seriestime-oriented dataset.

Feature	**Accuracy Score**
MFCC	53.31%
Mel Spectrogram	57.49%
Tempogram	45.87%
Zero-Crossing Rate	36.24%
Chroma STFT	40.07%
MFCC + Tempogram	53.01%
MFCC + ZCR	51.66%
Tempogram + ZCR + Chroma	48.59%
MFCC + Tempogram + ZCR	52.89%

As can be seen, the Mel Spectrogram performed the best of all, which is not surprising, given the wealth of information it provides, which is more subtle and reduced in an MFCC. However, it still becomes clear why MFCCs are so widely used, as they can achieve near-same accuracy, with run times 20x shorter. The kernel SVM finished training on the 11,000-row MFCC data points within 20 minutes, but took nearly four hours to run with Mel Spectrograms. Thus, for testing purposes, it becomes quite arduous for really fine-tuned testing using Mel Spectrograms on the local CPU.



Confusion matrix using an RBF-SVM model, training on Mel Spectrograms.

	precision	recall	f1-score	support
Blues	0.55	0.59	0.57	54
EDM	0.49	0.55	0.51	64
Hip hop	0.90	0.53	0.67	51
Metal	0.67	0.76	0.71	55
Pop	0.45	0.46	0.46	63
accuracy			0.57	287
macro avg	0.61	0.58	0.58	287
weighted avg	0.60	0.57	0.58	287

As can be seen, The scores are all quite scattered, with no obvious misclassification trends. It can be seen that metal overall was the easiest to classify correctly, and also was not used as much as other genres to misclassify non-metal songs. Hip Hop perhaps is the most curious case. Of all the songs that were predicted to be hip hop, almost all of them were actually hip hop (precision score of .90, the highest in the entire test set). However, many hip hop songs failed to be classified correctly (24 of the 51 total hip hop songs). Finally, pop scored the worst by far, with both low precision and recall, with pop and EDM suffering from each being misclassified as the other quite often. Pop was almost never misclassified as hip hop, but a sizable portion of hip hop songs were misclassified as pop, which is not surprising, given the evolution of many song genres to reflect tropes repeated in pop music. Thus, this analysis would be very interesting to do on older songs, where perhaps the boundaries of what made a song a particular genre were more well-defined.

Deep Learning Methods

As deep learning was not as much of a prime focus for this report and perhaps outside of the boundaries of this course, this section will mostly be summarized, however just as much time was spent investigating deep learning models used in music analysis, data preparation for

neural networks, and basic NN-architectures to achieve success in classification. Overall, a higher accuracy was, in fact, achieved with one specific type of neural network - a CNN - which is a model that scans image data, or pseudo-image data for patterns. These models include a multilayer perceptron, a convolutional neural network, and a recurrent neural network.

Due to the nature of the data being passed into these models, a single model receives one particular feature. Thus, the table below again shows the accuracy scores across the three different types of models shown for each feature. The architecture of each of the three final tuned models will also be detailed below. Explanations of each model and the way in which they are constructed can be found in the appendix.

Feature	**MLP**	**CNN**	**RNN**
MFCC	52.08%	58.89%	42.24%
Mel Spectrogram	56.25%	65.94%	44.93%
Tempogram	36.88%	36.15%	34.61%
Zero-Crossing Rate	28.60%	NA	31.10%
Chroma STFT	52.04%	45.20%	36.47%

Architectures of Models

- MLP: The final MLP model used had 4 total dense layers: the first layer with 100 neurons, the second with 64, the third with another 100, and the final layer with 5. Activation functions were all set to a “rectified linear unit” apart from the final layer, which uses softmax to provide the final prediction. All but the last layer have a regularizar and a 0.1 dropout to avoid overfitting. The final optimizer is an Adam optimizer with a learning rate of 0.0001, compiled with a cross-entropy loss function.
- CNN: The model has 3 overall 2D convolutional layers + Pooling layers, with kernel sizes of 32,16 and 8, strides of (3,3), except for the final convolutional layer, with a stride of (2,2). All activation functions remain as “relu”, and after the final pooling layer, there is a dropout layer of 0.3. Finally, the architecture ends with a standard Dense layer of 5 neurons.
- RNN: The RNN model has two LSTM layers, each with 64 units, and then a Dense layer with 64 neurons, and finally the final five-neuron dense layer.

Results

As can be seen, the best performance from deep learning came from the Mel Spectrograms using a CNN architecture, which was expected. While MFCCs run much faster and contain relatively the same information as the spectrogram, when represented as an image, the spectrogram has a much richer and detailed representation of the data as a 2D array.

Furthermore, the Chroma STFT, while also a strong representation of frequency data, Is very noisy, and the binned aspect of it (in which frequencies are represented by rectangular cells) destroys the shapes seen in a Spectrogram, making a more noisy image. Finally, tempograms probably did not perform well, as they're simply not as deterministic in terms of genre and are difficult to learn from. For the tempogram, even training accuracy seemed to reach ceiling effects around 80% accuracy, while other features could easily reach 95% or higher training accuracy when not accounting for overfitting with dropout layers.

Similarly to the other modeling efforts above, accuracies above 65% were never reached, with the biggest misclassification for deep learning happening between both EDM and Pop, in which both are equally misclassified as the other. Without these two genres, the accuracy for Mel Spectrogram CNN modeling can get as high as 81.25% for the highest 3-class accuracy. What is curious is that 30% of the blues songs in the test set are being misclassified as metal. There is only so much information that can come into features focusing on frequency distributions, but this was completely unexpected. Perhaps it could be corrected with more complex networks and input data (passing in two parallel 3D arrays as features, with different sizes, to be able to pass in a mel spectrogram, for example, along with a tempogram), but such intensive tuning for deep learning goes beyond the reach of this course, and is definitely worth investigation for next steps beyond this report.

```
array([[42,  1,  0],
       [ 3, 34, 12],
       [ 1,  7, 28]],
```

(confusion matrix for Hip Hop, Metal, Blues)

Takeaway

After running so many models on genre classification, confirming hypotheses about which models might work better than others, and which features might work best in which models, this part of the project was very illuminating in terms of understanding the capabilities of machine learning and music data, and the true complexity of such modeling with such an intricate data structure. While overall accuracy was never very high, most issues with accuracy involved the limitations of distinguishing Pop vs EDM in any algorithm, which is not surprising. These two genres are very similar in many ways, especially in terms of how Pop has evolved to include many of the tropes present in EDM over time. An interesting next step would be to understand exactly what in the data makes these two genres somewhat indistinguishable for a machine learning model, what truly makes them sound different to a human ear, and how those differences can be extrapolated in the data.

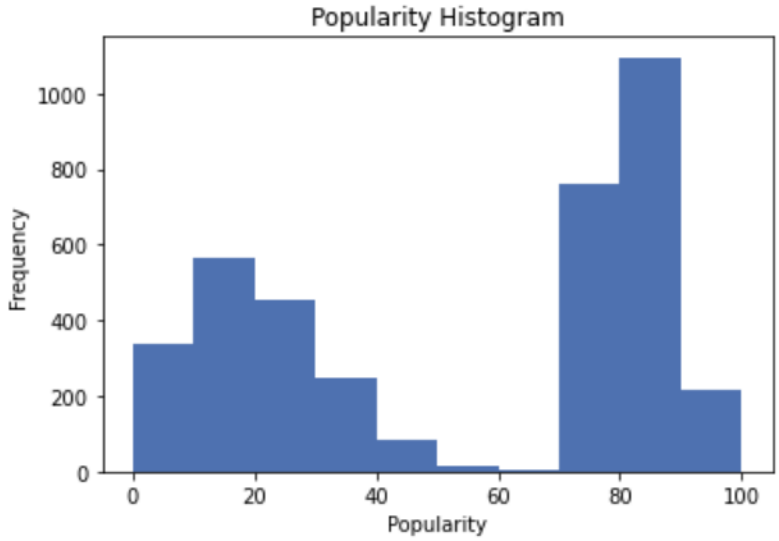
Finally, from this entire genre classification task, there have been many learnings, but primarily that, while machine learning can achieve amazing progress on detecting very distinct types of music, it struggles with more subtle differences. Our concluding hypothesis is that such modeling might be good at classifying, for example, the region of the world a song might originate from, or classifying distinct types of music, especially in other countries where musical genres are much more distinct and less fluid, like Latin American Salsa, or Brazilian Funk music,

which have very distinct and steady beats that are copied over every single song in the genre. However, even then, complex data engineering and modeling is needed to detect subtle differences among musical sound bytes, which could still be a barrier to entry for the Novice Composer.

Knowing what we now know from this part of the project, we move into the heart of our mission - determining song popularity.

POPULARITY CLASSIFICATION

As novice music producers and songwriters go through their creation process, it is important to understand what it takes to make a great song. In this section, we explored not only what characteristics of a song are important, but also explored methods to test whether the song created will be popular or not based on the actual raw wav files. There were two possible approaches to predicting the popularity of the songs. The first would be to use regression methods to predict the popularity value from 0 to 100 as given in the input. The second approach would be to take the values and set a threshold to turn popularity into a binary feature. To aid with the decision, we plotted the distribution of the popularity with a histogram. Looking at the distribution of the data, we found it to be bimodal, with one peak centered around 85 and another with a peak around 18 as pictured below. Based on this, we favored the approach of classifying popularity as a binary variable, representing popular vs unpopular songs. The binary variable was defined as “popular” for scores 75-100, and “unpopular” for scores 10-40.



Analysis of Spotify Features

As with genre classification, we use two separate datasets to model popularity. As seen in the above distribution graph, popularity as defined by Spotify is not normally distributed and hence

we will start by trying to model the first dataset with regression, and the second with classification. The first dataset uses the Spotipy data only, which contains similar features as in the genre classification (see appendix for details), while excluding genre (i.e. 23 features after encoding categorical variables).

We will also be analyzing the importance that the artist feature makes to the predictive power of the model. However, since this is not necessarily a controllable portion of any music composer's abilities, we wanted to separate it out for a portion of the analysis.

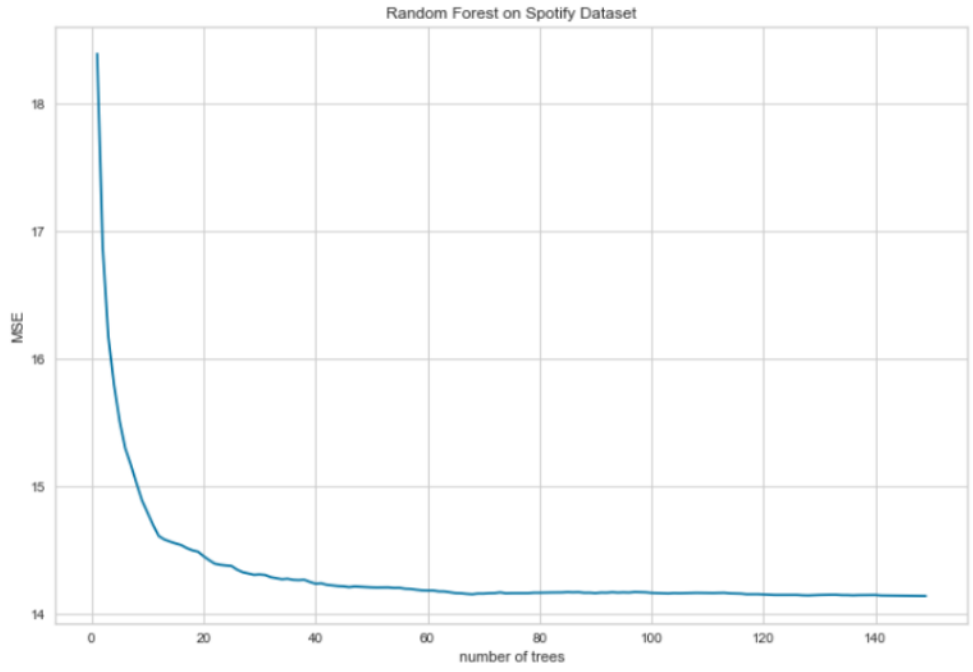
The dataset used for analyzing popularity consisted of 3767 individual tracks. As before, we normalize and standardize the data, since many of the variables are between 0 and 1.

Methodology

There were no NULL values in the dataset and the first step was splitting our data into a training (80%) and test (20%) set using a random shuffle.

The next step was to run 10-fold cross-validation on the training set against a list of models with hyper-tuning parameters. The models we tested for this part of the analysis were the following: Linear Regression (tuned , Ridge Regression, Random Forest, Gradient Boosting, Light Gradient Boosting and Bayesian Ridge.

As an example of our hypertuning process for Random Forest, we ran 10-fold cross-validation over the number of trees in the forest, from 1 to 150 and calculated the MSE and R2, resulting in the below graph:



The change in MSE started to drop-off at around 30 trees, and the tuning parameter with the best result on the CV set was n_estimators=37.

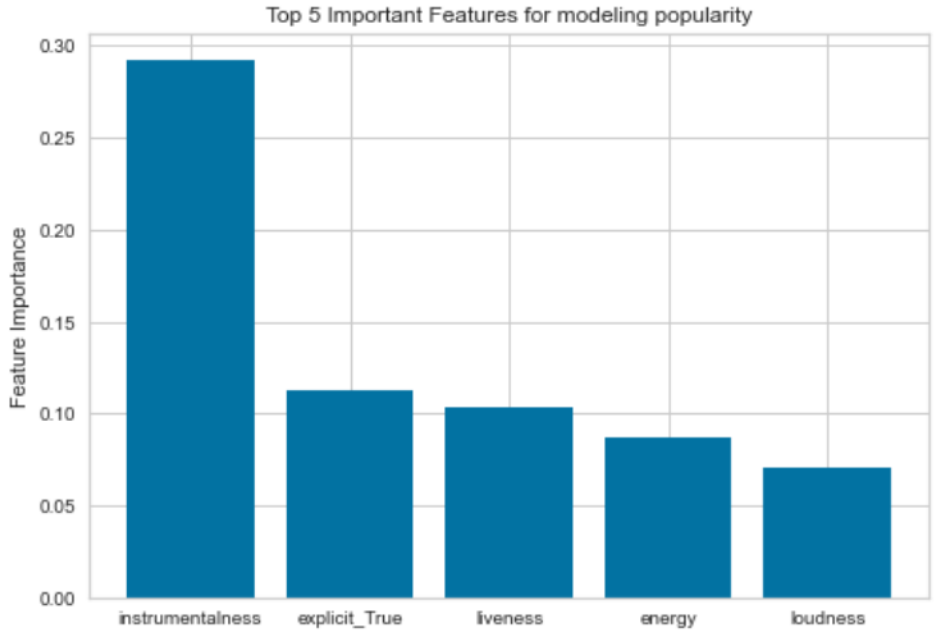
Results

The final MSE and R2 results on the cross-validation set from our different models are the following:

Model	**MSE**	**R2**
Random Forest	394.63	0.586
Light Gradient Boosting	407.87	0.581
Gradient Boosting	409.86	0.570
Ridge Regression	508.28	0.467
Linear Regression	508.35	0.467
Bayesian Ridge	508.33	0.467

As we can see, there wasn't much difference in performance among these models, and they all have relatively low predictive power on popularity (which is in the range 0-100), perhaps due to the low amount of data for these features and the distribution as discussed previously.

However, below is a graph of the top 5 important features for modeling popularity by regression:



When running a prediction on the test set from the best performing model (Random Forest), we get the following results:

Model	**MSE**	**R2**
Random Forest	450.49	0.522

Showing that there could be a small amount of overfitting on the training set, which was likely due to the dataset being small. We could either try to collect more data on individual tracks (increasing the sample size), or adding more features from different APIs/providers. For example, we could include lyrics data from Genius and run NLP algorithms for key features such as word frequencies.

As an aside, if we do include Artist as a feature (where each individual artist has it's own column), we see a significant improvement in every model on the cross-validation set, especially Ridge Regression. Below is a table on the top 3 models:

Model	**MSE**	**R2**
Ridge Regression	286.68	0.69
Bayesian Ridge	302.02	0.67
Random Forest	346.02	0.63

These 3 models are showing great predictive power, specifically Ridge Regression, which was hyper-tuned in a similar way, over different values of alpha and tolerance.

Although we have many artists which only have 1 track (821) out of the 1370 artists, we still felt this was significant to test, since there were a total of 3767 tracks, of which roughly 21% were 'unseen' artists.

Using the best performing model from our cross-validation set (Ridge Regression), running prediction on the test set gives us an MSE of 398.54 and R2 of 0.59 suggesting there may be some overfitting on the training set. This is likely due to the number of individual artists resulting in "unseen" bias in the training data.

In conclusion, regression analysis provided some interesting insights, but due to the distribution of the data, the next steps would be to run classification by first splitting the popularity numerical label into a categorical one. We could either use a clustering algorithm, SVM or simply choose a cut-off point based on the distribution above to identify what is "popular" vs. "unpopular".

Since artist is an important factor, we could also find a way to break the artist category into a small subset of categories, based on prolificity. For example, number of Spotify followers, or other metrics such as general social media influence or search statistics.

Takeaway

From the results of our analysis on Spotify features, we found that the named artist producing the song (as well as featured artists) is the most important factor when predicting popularity. This is unsurprising, as an existing fanbase as well as experience and access to specific benefits (managers, advertisement, media, etc.) paves itself to more awareness and higher active listeners. A novice musician can not account for this factor easily, but could try featuring with other artists to increase coverage across multiple regions, domiciles and industries, as well as mix popular lyrics/beats with the correct licensing.

However, because we were aiming to provide insights to novice musicians, there are other important factors to take into consideration, which also had fairly good predictive power.

For example ‘instrumentalness’, being the presence or absence of vocals, is also an important factor when predicting popularity. Higher popularity is expected from songs with words, rather than just a beat. Interestingly, our ‘explicit’ flag was also an important factor when determining popularity.

Analysis of Sound-Byte Data

Methodology

As a summary of our data section in relation to the popularity sound-byte data, we ran our data through the model with three subsets: One of 100 popular and 100 unpopular songs, 250 popular and unpopular, and the third, 100 songs split into 5-second clips. This was done as the first subset would define a baseline performance, the second subset would check for overfitting, and the last subset would test for improvement in performance with a larger subset size. Then, once the data was split, we extracted our three main features to be used for popularity classification: the MFCC, the zero-crossing rate, and the fourier tempogram. With our dataset fully prepared for analysis, we then used modeling to test for predictive power. To do so, we first ran the dataset through several models without tuning to see the baseline performance. After recording our baselines, we then turned to tuning the models.

To perform the baseline analysis, we randomly shuffled the stratified lists of popular and unpopular songs, split the X-values from the popularity label, and transformed popularity into a binary variable as outlined at the beginning of the section. Then, we split our data into training and test sets via a 30:70 split. We repeated this process for the other subsets mentioned above. As this problem now involves classification, the models we used included a random forest classifier, a decision tree classifier, xgboost, SVM, K-Nearest-Neighbors, naïve bayes, one-class SVM, adaboost, and catboost.

With each model, we trained on the X and Y training sets with the base configurations and predicted y_hat from the X test set. Then based on those results, we calculated the accuracy, precision, recall, and f1 scores. We also generated the confusion matrices to visualize classification and misclassification patterns.

Then, for each of these models, various parameters can be tuned to potentially increase accuracy. Thus, we performed a grid-search 5-fold cross validation with each combination of hyperparameters, searching for the best parameters while combating overfitting.

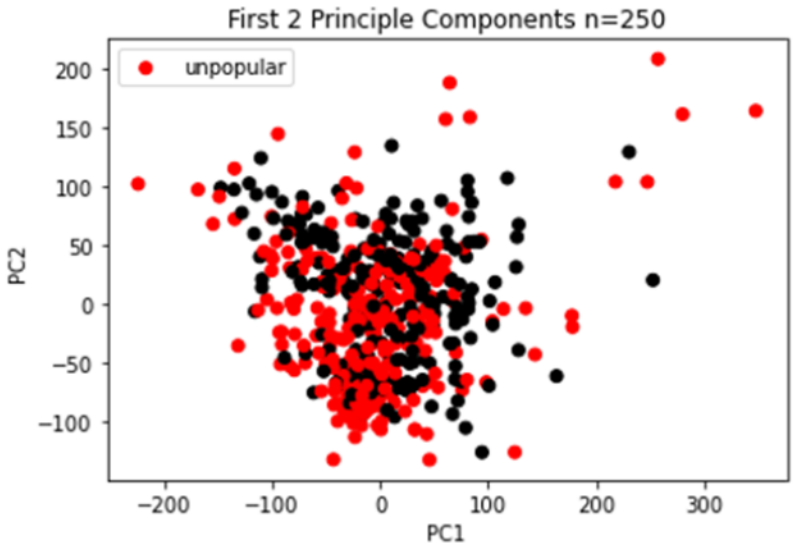
For the decision tree, we tested max_depth values from two to fifteen. This value corresponds to the maximum depth the tree will go, or how many times it will branch downwards. With the random forest model, we tuned the n_estimators value from 70 up to 160. This corresponds with the number of estimators that would be used in the forest. Next, with the naïve bayes classifier, we tuned the var_smoothing parameter from 1e^-11 to 1e^-7. This corresponds to how the variances will be augmented in the model.

As for the SVM model, we tuned the kernel. The kernel defines how the input will be transformed through matrix multiplication. For this hyperparameter, we chose to search the values in the following list: - linear, poly, rbf, and sigmoid. The linear kernel will define a hyperplane that is linear; the poly kernel will define a hyperplane that is a polynomial; the rbf kernel uses the normal curve around the dataset; and the sigmoid kernel uses the sigmoid function for the hyperplane. Next, we found the best hyperparameter for KNN, which is an non-parametric, supervised learning approach. Nevertheless, we can tune this model based on the number of neighbors used for classification per point, as well as the weights that are placed on those neighbors. We search for neighbors from 1 to 20 and weighting methods being uniform or distance based. Next, we ran the grid search with the adaboost classifier. Here, we tuned on the learning rate and number of estimators. Since adaboost is an ensemble classifier and uses multiple estimators, the n-estimators parameter specifies how many estimators to use. The learning rate states how fast the model should learn, more specifically what weights to use on each of the n classifiers. We search from 30 to 80 for the number of classifiers, and from 0.1 to 5 for the learning rate. Lastly, with the CATBoost classifier, we ran a grid search on the depth, iterations, and learning rate function parameters to tune the model. The learning rate tells the program how fast to perform gradient descent with the chosen loss function: log-loss. The depth corresponds to the depth of the tree. The iterations parameter tells the model how many times to run. Based on this, we chose values for learning rate 0.1 to 1, iterations from 1 to 10 and the depth from 6 to 10.

With each tuned final model, we then made final predictions on the second subset of data, consisting of 250 songs, to validate model performance and check again for any overfitting.

Lastly, as some models improved with introducing additional data, we repeated the process one additional time with the last subset, 100 popular and 100 unpopular songs split into 5 second clips.

As a complement to such rigorous model evaluation, we also explored potential dimensionality reduction methods, primarily PCA, to see if popularity could potentially be more separable with reduced dimensionality. The first two principal components are plotted below:



Results and Evaluation

Our first pass of model training was done with just the base models and a training and testing set. With these models we calculated the scoring metrics mentioned in the methods section. For each of the models see the results summarized in the table below.

Model	**Accuracy**	**Precision**	**Recall**	**F1**
Random Forest	80%	70%	87.5%	77.78%
XGBoost	63.3%	53.1%	70.8%	60.7%
SVM	63.3%	52.8%	79.2%	63.3%
KNN	68.3%	72.7%	33.3%	45.7%
Naïve Bayes	68.3%	57.1%	83.3%	67.8%

One Class SVM	60%	0%	0%	0%
Adaboost	68.3%	58.6%	70.8%	64.1%
Catboost	65%	56%	58.3%	57.1%
Decision Tree	45%	37.1%	54.2%	44.1%

Based on these results, we can see that the best performer was the Random Forest Model across all metrics – accuracy, recall, and f1. As can be seen, precision is generally lower than recall, which tells us that our models in general lead to more false positives. For this situation, this means songs that are not popular songs are being output as popular songs. The other thing to be cognizant of is that we used a small test set here of only 30 songs for each class. Thus, even our best model of 80% accuracy might not be fully robust with larger testing sets. To combat this, in the next step, we used tuned models using cross validation to get more robust accuracy scores with less potential for overfitting. The accuracy scores per model are detailed below, along with a description of the final optimal parameters:

Classifier	**Optimal Parameters**	**Accuracy (100)**	**Accuracy (250)**	**Accuracy (5 sec clips)**
Random Forest	n estimators: 140	75%	63.7%	72.9%
Decision Tree	Max_depth: 3	50%	57%	65.5%
Naïve Bayes	Var_smoothing: 1e^-11	68.33%	67.8%	70.2%
SVM	Kernel: poly	65%	69.7%	67.9%
KNN	4 neighbors	66.7%	57%	70.9%
ADABoost	learning rate 0.1 n_estimators : 80	65%	57%	72.6%

CATBoost	Depth: 1, iterations:2, learning_rate: 1	51.7%	55.7%	57.8%
XGBoost	N_estimators: 40	61.7%	71.1%	77.6%

Based on this output we see that the best model from the last round, the random forest, only drops 5% when tuning and cross validation are introduced, which tells us that the model with low input data still does well. However, when we train with additional data, we see that accuracy then drops an additional 12%. Another interesting result here, despite accuracy being fairly low, is the increase of the SVM model's accuracy from 65% to 69.7% with the bigger dataset. Otherwise, across the board we see a decrease in the accuracy as we increase the dataset size.

Takeaway

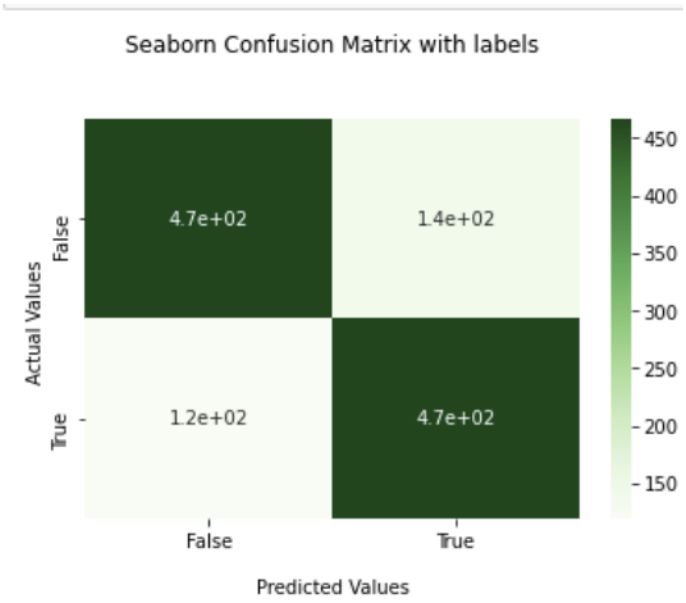
In summary, most models turned out to be somewhat weak classifiers, with generally higher recall than precision, thus showing a tendency for modeling on this dataset to generate many false positives, which is also something we found when classifying genre. The highest accuracy seen was achieved by a random forest model, obtaining 80% accuracy. One potential area of weakness could be our data collection process, in which we used API calls to mass-pull songs of a particular genre, without considering how the API might pull these songs, and if there might be any bias motivating which songs are pulled per genre. Due to the size of the dataset, and the sheer amount of music present in the world, perhaps much more time could have been taken to better tune the final list of songs to be a more balanced dataset. Furthermore, songs that may have been popular 3-5 years ago might show up in the list now as "unpopular," as popularity scores are updated for current date. Thus, this could also be skewing our results, and leads to higher levels of uncertainty with the final results of model prediction.

Looking at the effect of using a larger dataset, we saw that the random forest's accuracy dropped considerably when the dataset was increased to 250. This goes to show that the model was overfitting on the smaller dataset, despite the minimal decrease when cross validation was applied with the 100-song dataset. Thus, if we were to continue to increase the size of the dataset, the random forest's reported accuracy score will likely continue to decrease to a certain point until reaching a true, robust accuracy. Furthermore, it seems that most models resulted in a lower accuracy with the larger dataset, thus showing that most models suffered from overfitting.

However, there were two models with significant improvement, and those include the SVM model, which increased almost 5%, and the xgboost model which increased almost 10%. When we used the 5-second clips dataset, which is even larger, xgboost improved even further. It would be interesting to see how the models will improve as they gain even more data. It could be true that, as more data is added, the difference in accuracy between the datasets will be

even larger. In order to test this, I pulled an additional dataset of 400 popular and 400 unpopular songs and retrieved 5 second clips, thus resulting in almost 4000 data points total to work with. Running this with a test-train-split of 30/70 and 40 estimators with the xgboost classifier, I was able to obtain the following results:

Accuracy	**Precision**	**Recall**	**F1**
78.3%	77%	79.7%	78.3%



As our end goal is to provide a model that gives songwriters the ability to test their songs for potential for popularity, we took a 30 second clip of a well known song to see if it will be properly classified. When passing Despacito into the XGBoost model, we found that the output was 1, which aligns with how popular this song actually is. Additionally, we found a very unpopular song called Fat Idiot and the output was 0. Although this is one data point, it is interesting to see that the output matches with what we were trying to do.

DISCUSSION AND FUTURE WORK

Project Summary

While modeling on both genre and popularity classification proved to be quite complex, with accuracy in both regards reaching a maximum of around 80%, it is difficult to know whether simply more complex modeling, new data sampling, different sampling and augmentation techniques, or different data could possibly turn these final models into robust predictors. Even if most models served to be somewhat weak predictors in the end, we were still able to build

models that not only provide insight on characteristics that make a song either popular or of a certain genre, as done with the spotify characteristic features, but also created fitted models that, within a range of accuracy, can help a novice composer test their song against the model via prediction, and determine at least a possible likelihood of their song fitting within their preferred genre, and ultimately having the chance of being popular.

As for genre classification, we learned that acousticness, speechiness, energy and danceability all contribute the most to determining a particular genre. Furthermore, for truly testing the genre of a composed song, certain features related to frequency distributions within the song, such as the MFCC, can be extracted from the sound byte and tested against other songs with a convolutional neural network to determine similarity of a song to particular genres learned by the model. Furthermore, it is perhaps not surprising that model accuracy can not reach above 80%, as genres themselves are fluid definitions, and thus there is a lot of overlap among various genres, hence why models seem to have such a difficult time separating EDM from Pop.

As for providing a mechanism to test their songs, from our analysis we saw that the xgboost model was able to provide a binary popularity prediction with almost 80% accuracy. From the perspective of a songwriter, for every 5 songs they pass through it around 4 predictions will be accurate. I'd say if songwriters had this sort of insight and with this accuracy they'd be pretty happy. Additionally, with some of the future work methods below, that accuracy could improve as well. Additionally, we found there were many models that worked well and many that didn't work well. As a whole, the dataset did seem to favor classifying songs as popular. So, of that average of 1 out of 5 songs that would be misclassified, the song may be classified as popular, giving the budding music producer a boost of confidence.

Future Work

With this project, we were able to create multiple final data models that can be used to predict both genre and popularity. Overall, the final models struggled with reasonable results. For genre classification, apart from Pop and EDM, even simple deep learning models were able to predict genres with 80% accuracy. While not perfect, in future iterations of this project, it would be worth investing time into studying the reasons for misclassification, specifically across certain genres, such as pop and EDM. Furthermore, in terms of genre classification, it would be interesting to run this entire test again on songs across various global music genres, such as salsa, cumbia, samba, and others, which obviously have very unique characteristics from one another with less influence across genres, and more regulated tempos, unlike the very fluid nature of genre in American music.

As for popularity, we found that, rather unsurprisingly, the artists accredited to the song have a large influence on its popularity. For future work, it would be interesting to see if, by taking the average of previous titles, we could isolate the features that make a song more popular independent of the artist producing them. We could do this by normalizing the popularity factor based on the average, min or max popularity for any artist.

Additionally, based on our results with the sound-byte data, an additional step would be to manually scrape the internet for the popular and unpopular songs for each year, to remove any bias that would come from the spotify api output. This may help in separating the two datasets more significantly and provide better results. Additionally, from the XGBoost model, we saw that growing the dataset improved the output performance, so adding in much larger datasets beyond the computing and memory power of our machine may be useful in creating a more accurate model.

Finally, a tool that could complement this analysis very nicely would be a GAN model that is trained to compose music on its own, thus allowing a novice composer to use a machine to gain inspiration and learn from audible examples on what makes a song popular or of a specific genre, based on the output of the GAN model. In theory, GANs can serve to create entire compositions on their own, which could, if perfected, also be a valuable asset to a novice composer, who then would only have to control inputs, rather than having to learn the entire composition process on his or her own.

CONTRIBUTIONS

Tyler Lang (GTID 903737252): I was responsible primarily for feature engineering and for all analysis concerning genre classification with raw sound data. I worked more on data processing and feature extraction, as well as exploration with neural networks. As my section on genre classification was not our main finding, but served primarily to be exploratory in nature, I spent more time learning about the raw audio data, what the extracted libROSA features are and what they represent, and how these features interact with various modeling tools in machine learning and deep learning to give suitable outcomes, by testing the data among various families of models to see which performed best. I also spent a lot of time learning about and applying augmentation on the original data, and finding balances in the raw data in terms of size (row length) vs richness (extra columns) for predictive purposes, focusing as well on how augmentation and splitting must be tuned to provide valuable data points to increase model accuracy while avoiding overfitting, and avoiding augmented predictive error. I also heavily researched neural networks, and their implementation in Keras, especially in terms of analyzing audio data, focusing the most time on CNNs.

Naveen Ram (GTID 903127600): I was responsible for the data collection and cleaning steps as well as the analysis of the popularity classification with the raw sound data. I spent a good amount of time looking into past papers in the area of classification with music data. I also worked on the manual forward selection methods with all the 27 libROSA feature sets we had selected. I explored numerous classification algorithms as well as tested those with a variety of input data, sizes, and segmentation, including 200, 500, 1000, 2500, and 4000 data points. I explored how each of these impacted the output accuracy of the data. In addition, I built baseline models and then tuned those models on various hyperparameters to find the best performing models through cross validation.

Marc Evans (GTID 903744769): I was responsible for analyzing and reporting on the Spotify feature datasets for both genre classification and popularity regression, testing both forms of modeling a problem to draw valuable insights for novice musicians. I trained and hyper-tuned multiple models and found that genre was easier to classify based on the 4 opposing genres that we chose. And also that, popularity was the best factor to identify genre, and artist was significant in predicting popularity.

APPENDIX

I. Spotify Features

For this dataset we have the following table of features.

Column Name	Description	Data Type	Example
id	Unique identifier for each song	String	2Xr1dTzJee307rmrkt8c0g
explicit	Boolean of explicit content as defined by Spotify	Boolean	True
popularity	Rating of song popularity as defined by Spotify (e.g. most plays)	Float [0.0 → 100.0]	98.5
danceability	How suitable a track is for dancing based on a combination of elements including: tempo, rhythm stability, beat strength and regularity.	Float [0.0 → 1.0]	0.882
energy	Perceptual measure of intensity and activity. A high value would constitute a fast, loud and noisy track.	Float [0.0 → 1.0]	0.942
key	The key the track is in. Integers map to pitch in standard pitch map notation.	Integer [0 → 11]	0 = C
loudness	Overall loudness in	Float [typically -60 →	-6.282

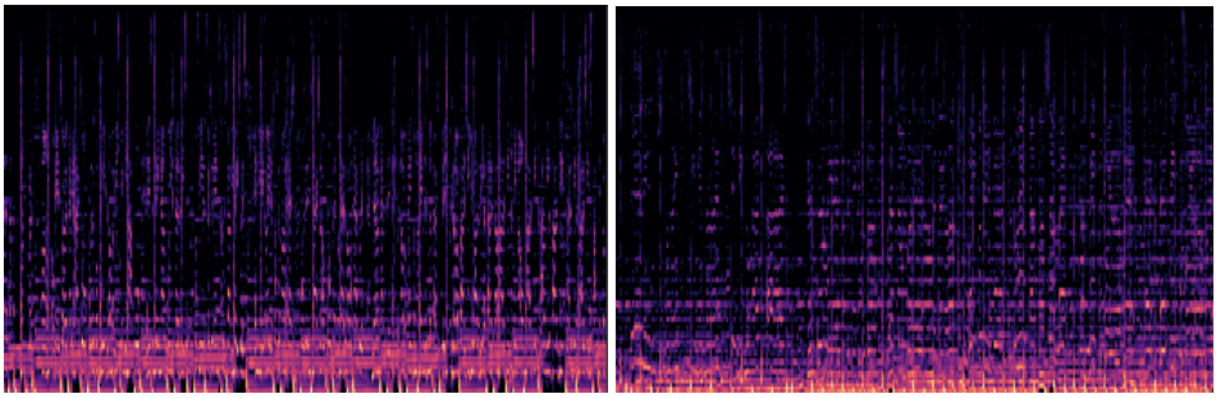
	decibels (dB). Averaged across entire track. [relative?]	0 dB]	
mode	Mode indicates the modality (major or minor) of a track, the type of scale from which its melodic content is derived.	Boolean	Major is represented by 1 and minor is 0.
speechiness	Speechiness detects the presence of spoken words in a track. The more exclusively speech-like the recording (e.g. talk show, audio book, poetry), the closer to 1.0 the attribute value. Values above 0.66 describe tracks that are probably made entirely of spoken words.	Float [0.0 → 1.0]	Values between 0.33 and 0.66 describe tracks that may contain both music and speech, either in sections or layered, including such cases as rap music. Values below 0.33 most likely represent music and other non-speech-like tracks.
acousticness	Confidence measures from 0.0 to 1.0 of whether the track is acoustic.	Float [0.0 → 1.0]	0.0241
instrumentalness	Predicts whether a track contains no vocals. "Ooh" and "aah" sounds are treated as instrumental in this context. Rap or spoken word tracks are clearly "vocal".	Float [0.0 → 1.0]	The closer the instrumentalness value is to 1.0, the greater likelihood the track contains no vocal content. Values above 0.5 are intended to represent instrumental tracks, but confidence is higher as the value approaches 1.0.
liveness	Detects the presence of an audience in the recording. Higher liveness values represent an increased probability	Float [0.0 → 1.0]	A value above 0.8 provides strong likelihood that the track is live.

	that the track was performed live.		
valence	Describing the musical positiveness conveyed by a track.	Float [0.0 → 1.0]	Tracks with high valence sound more positive (e.g. happy, cheerful, euphoric), while tracks with low valence sound more negative (e.g. sad, depressed, angry).
tempo	The overall estimated tempo of a track in beats per minute (BPM). In musical terminology, tempo is the speed or pace of a given piece and derives directly from the average beat duration.	Float	93.00
name	Name of song	String	Love nwantiti
artist	Named artist	String	CKay
url	The Spotify URI for the track.	String	https://listen.hs.llnwd.net/g3/prvw/1/5/7/5/6/2448465751.mp3
genre	Genre for the song as defined by Spotify	String	List from: - Pop - Hip Hop - EDM - Metal - Blues

II. Sound Byte Features

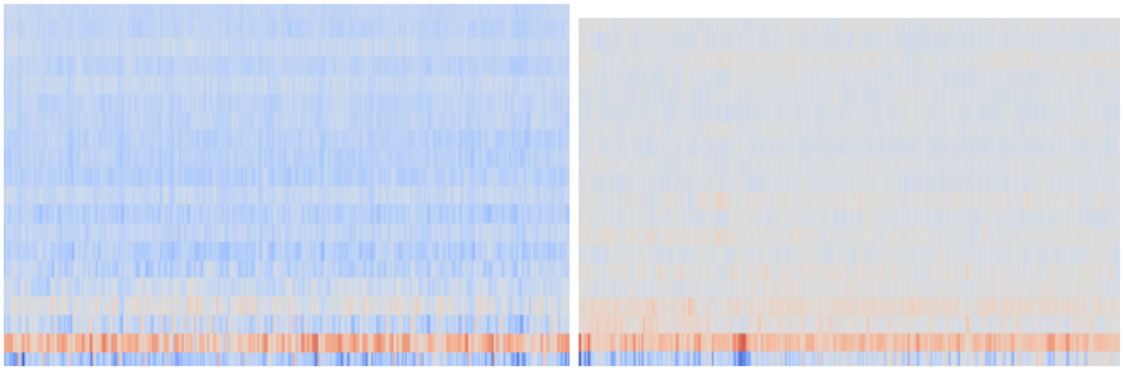
Mel spectrogram: A spectrogram is simply an image, or heatmap, of the intensity (amplitude) of all given frequencies over time. While raw sound data frequencies are simply represented by the density of certain waves within an interval, a spectrogram applies transformations to this format to plot frequency as its own axis, with amplitude thus being the "third" axis, represented by color intensity from dark purple to bright red/yellow. Then, the "Mel-frequency" is a simple transformation done to original audio, which scales audio to a shape that is more detectable by the human ear, in which an X-Hz change in a high frequency is less discernible to the human ear than the same X-Hz change in a lower frequency. This transformation serves to better differentiate music and audio, as our perception, and thus classification, of audio is highly

dependent on our own audial processing of such sound, and thus this transformation is very helpful in classification models, where otherwise such changes in frequency over the entire frequency scale might seem more noisy and less dramatic at certain frequencies compared to what we truly experience. Mel spectrograms contain all the original information of the song, but represented in a condensed way, thus only slightly reducing the size of the raw data, but providing a lot of rich information to a machine learning model, especially for models specializing in image classification, as the final mel spectrogram is a 2D array which can be plotted as an image.



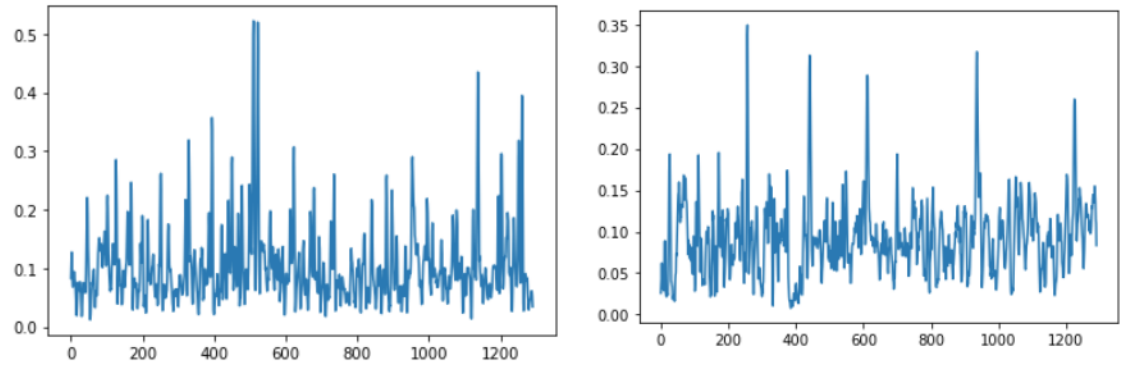
Mel-Spectrogram of "Drop it Like it's Hot" || *Mel-Spectrogram of "Something just Like This"*

Mel-frequency cepstral coefficients (MFCC): The MFCC is one of the most commonly used features in audio classification algorithms, and for good reason. It is an extremely dense representation of the frequency layers present in a song, preserving intensities in frequencies within an audio into a small 2D matrix. Thus, MFCCs are useful not only as informational buckets that describe a song's frequency, but can also be represented in an image-like manner like a Mel Spectrogram, thus making them great candidates for CNN deep learning models, or other models apt for pattern/edge-searching in 2D data structures. Formally, the MFCC is a cepstral graph, with a cepstrum being a transformation of the spectral-space. With music, or sound, being in a time-domain, in which sound is represented as frequencies and amplitudes over time, that data can be transformed into a frequency domain, in which frequency per amplitude is plotted, almost as a histogram. A cepstrum, then, is a representation of the rate of change in these frequency histograms over time. It thus serves as batches of windowed, or binned, frequency histograms, with the final x-axis being time, and the y-axis being these integer windows of frequency density. The initial spectrograms are transformed to the Mel scale, as explained above in the "Mel Spectrogram". Thus, the MFCC can be seen as tabulated Mel spectrograms over time. ‘



MFCC for "Drop it Like it's Hot" || *MFCC for "Something Just Like This"*

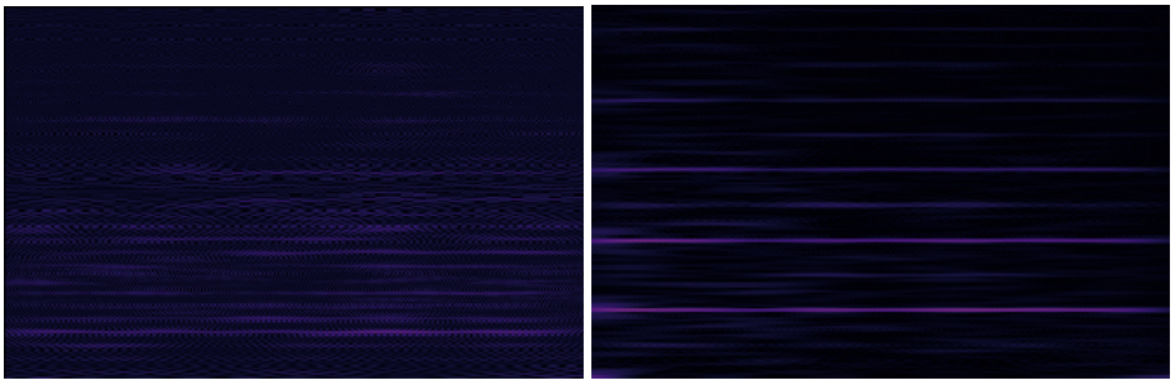
Zero Crossing Rate (ZCR): The ZCR is a very simple feature, which extracts the rate at which a signal crosses 0, or changes from positive to negative, or negative to positive. As sound is represented as waves of amplitude, vibrating in such a way that the amplitude goes from positive to negative in the form of extremely fast vibrations, the rate at which these vibrations fall and drop is essentially what is captured in the ZCR. In terms of music, while music presents a lot of variability in frequency changes across the different timbres of instruments and sounds happening all within the same time frame, ZCRs have been found to be very useful in detecting percussion and rhythm. ZCRs are perhaps the most-reduced form of the original raw data, and are a 1xN array, which makes it easy to stack across multiple songs to pass in any standard machine learning model, as it already has a flattened form. ZCRs are useful in providing higher levels of detection to genres of music with heavy and distinct percussion patterns, but alone are generally weak classifiers.



ZCR for "Drop it Like it's Hot" || *ZCR for "Something Just Like This"*

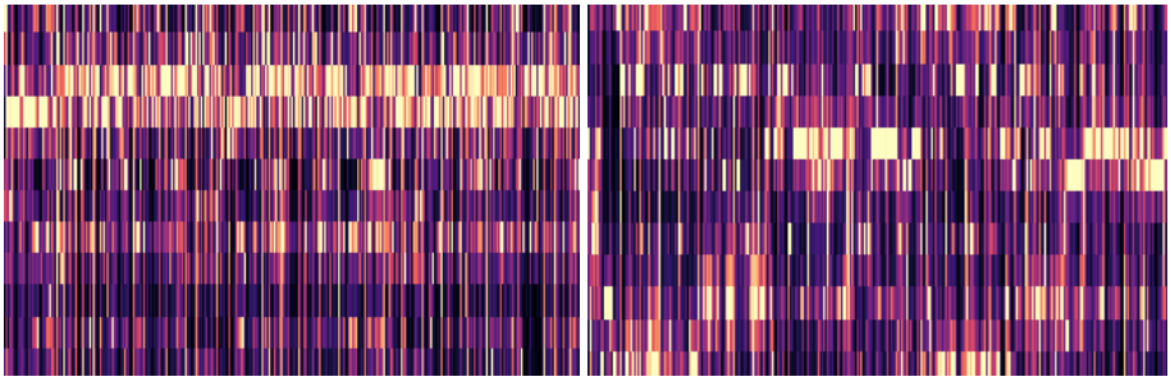
Fourier Tempogram: As tempos within a given song can vary over the length of the song, a tempogram is a matrix representation of the presence of specific tempos within the song at each time interval. A Fourier transformation is simply a transformation using mathematics that changes a function with time as its independent variable, into a graph of temporal frequency. Thus, the fourier tempogram simply shows the frequency of each tempo of the entire tempogram across all tempos within a given interval over time, with similar axes as the mel spectrogram, with the color representing the frequency of the tempo, the y axis representing tempo (and thus, the final "image" appears in the form of bands, or straight lines, as tempos are

not continuous variables), and the x-axis representing time. This feature helps strengthen modeling for genres of music with similar pitch patterns, like Pop and EDM, but which have different tempo progressions. The resulting matrix is often very large, and thus, when combined with other large features like the Mel Spectrogram, re-introduces memory limitations on a local computer. However, it has shown to be effective when combined as a feature with the highly-reduced MFCC, and more effective for longer sound bytes over shorter sound bytes, due to the length of time needed in a song to change tempo in patterns that can be discernible from other songs or genres



Fourier Tempogram for "Drop it Like it's Hot" || *Fourier Tempogram for "Something Just Like This"*

Chroma STFT: Related to the information represented in an MFCC or a spectrogram, a Chroma simply splits up the entire spectrum of sound into 12 bins, which are called "Chroma". These also can be represented as heatmaps, but are represented not by continuous variables as in a spectrogram, but as binned frequencies over set time intervals (seconds), and thus appear as heat-boxes. This representation simply provides an alternative "image" of the same data, but scaled in a different way, and with binning, shows a nicer, simpler representation of chord progressions and pitch densities.



Chroma STFT for "Drop it Like it's Hot" || *Chroma STFT for "Something Just Like This"*

III. Sound Augmentation Methods

Time Stretching: Changing the speed of an audio signal without affecting pitch. In other words, preserving the "tightness" of the waveforms in frequency over a given sample, but increasing the duration of this frequency to slow down the audio.

Pitch Shifting: Increasing or decreasing pitch by full units, or scales, thus preserving frequency patterns, but "tightening" up the frequencies to achieve higher pitches, or "stretching" them out to achieve lower pitches, while maintaining the same patterns, or shift in pitch, within a sample period.
Gaussian Noise: Simply adding random noise to the audio, with the noise fitting a gaussian distribution.
Loudness: Amplifying or dampening the loudness of an audio - in other words, changing the amplitude of each waveform over the entire sample, resulting in equal reductions in the absolute values of each observation.
Song Reversal: Reversing the direction of the audio clip, which translates to a simple reversal in the array. This helps maintain patterns within the frequencies as well, even if reversed, which is *not* useful for time-based modeling, such as RNNs, but is useful for pattern-recognition modeling, such as a CNN. As such, reversal was used very infrequently, due to its specific use-case.
Time Masks: Randomly silencing small sections of the audio, chosen as a random interval.

All augmenting was achieved with a package, "audiomentations," which was created specifically to augment raw sound data. This package allows one to create an augmentation function which, in the order defined in the code, goes through each augmentation process, picks an effect size within the minimum and maximum effect sizes defined in the function, and finally uses a probability to determine if the effect will be placed on a given song. This randomization allows a high degree of variability across all augmentation processes, across each song, thus combatting the possibility of overfitting on these augmentations themselves.

IV. Definitions of Neural Networks: MLP, CNN, RNN

Three types of neural networks were used:

- **Multi-Layer Perceptron (MLP):** These networks are standard neural nets that are made of various dense layers, each with its own activation function and quantity of neurons. The first layer receives the original training data, and this data is transformed via the activation function and passed layer to layer, until getting to the final layer, with the number of neurons equalling that of the number of classifications, which in this case, is five. Given the size of the data, many neurons were needed to achieve slight increases in accuracy per layer, but overfitting proved to be extremely problematic, thus requiring dropout layers, in which certain neurons, with a random probability, would "drop out" and stop influencing the model over one iteration, or epoch.
- **Convolutional Neural Network (CNN):** CNNs are models in which data is passed in a 4D array, in which the first dimension is the observation of data, or the song, and the last three dimensions being the image, with the last dimension being 1.. Within each data point, the CNN has multiple 2D convolutional layers, followed by pooling layers. The data passes to a convolutional layer, and with the given size, scans the 2D image with this M-by-M sized scanner, in which it applies a transformation over this MxM space for each cluster of MxM pixels in the original 2D array, and reduces the information to a reduced

space in the pooling layer, which is then passed to the next layer. In this manner, this model excels in detecting edges and other shapes in a photo. Finally, the reduced information is passed to the final layer, which is again a 5-neuron layer to predict the genre.

- **Recurrent Neural Network (RNN):** These are networks in which there is a feedback loop within the hidden layers over segments of sequential data, in which the order matters. Due to the sound byte data being sequential, or time based, RNNs are also often used in music and sound classification. During this feedback loop, learnings from previous time segments are passed in when learning the next segment, allowing the hidden layers to adjust and tune based on the order in which data is being passed in – in this case, the sequential order of our 2D arrays. Yet, RNNs have very "short term memory" as the weights of previous sequential learnings become more and more miniscule the more sequential data is passed in. Thus, it could be an issue for massive arrays, in which it consists of a large sequence.