Extracting and Analyzing the Data in Sound Files to Train a Music Genre Classification Model

Tom Constertina

Case Studies in Machine Learning
University of Texas at Austin

Project Repository: https://github.com/Tominatoo/ai_csml_final/tree/main

ABSTRACT

Music classification has become increasingly important over recent years, with the widespread use of streaming services that provide music suggestions for its users. We have developed a simple music classification model that has been trained on the songs from the free-to-use GTZAN Dataset to classify sound files into a musical genre based on their features [1]. For each song, we decomposed the song into different musical features using the Python library of LibROSA [2]. Musical features such as MFCCs, spectral bandwidth, chroma features, and tempo were extracted and analyzed. After that, we used all of these features to train a model and evaluate its performance. We trained on pre-built models from the Python library of *sklearn* and our own model with a custom neural network architecture.

INTRODUCTION

Music is an incredibly diverse and rich form of art, with many different genres and styles from numerous cultures throughout thousands of years. Even with the culmination of the experience and development of music throughout the years, music is hard to quantify. What makes a good singer good or bad if they both sing on-key? How does emotion change the way music is perceived? How does vibrato change the perception of a song [3]? With the introduction of recording equipment and sound analysis, some of these questions are becoming easier to define. What was once a very abstract art form can now be analyzed, stored as data, and decomposed to determine what makes up a sound. Generative AI models have taken great strides in interpreting music and then being able to generate unique music.

Though AI is no replacement to human-created music, it can play a key role in understanding sound and shaping the next era of music [5]. With technology playing a role in assisting music classification and generation, this paper aims to learn what features make up a sound, how to create a model on the data extracted from a song, and the best-performing model in classifying music into a genre. While in this paper we are only focused on genres, this approach can also be used for more abstract classifications, like finding other songs that are similar to a user's preference in melody, rhythm, instrument types, and more.

BACKGROUND

There has already been a great deal of research and work gone into developing AI models for music tasks. Oord et al. explored how transfer learning can be utilized to overcome the issue of lack of availability of free music [6]. In this paper, it discusses being able to transfer the knowledge from a model for a different, but related task, and utilize it for a new task with a different dataset. This helps to overcome the issue of scarcity of copyright-free music as well as the large file sizes and lengthy training times that go into model development.

Jia discussed creating a convolutional neural network to train a model to classify the overall 'emotion' of a song [7]. The input features for this model were the MFCC features of a song, and we will also use MFCC features in our model. MFCC stands for Mel-Frequency Cepstral Coefficient and has become instrumental in many aspects of speech processing [8]. MFCC will be helpful in our project as it can represent aspects of a song such as its rhythmic, harmonic, and timbral properties.

Lazaro et al. uses Audio Spectrum Centroid (ASC), Audio Spectrum Flatness (ASF), and Audio Spectrum Spread (ASS) to classify a song's tempo [9]. ASC is used to determine the most prominent frequencies at a certain point in time. ASF is used to measure the 'flatness' of a song at a point in time. A high flatness would mean there are a lot of frequencies present in similar strengths, such as white noise. A low flatness would indicate just a few frequencies dominating the sound. ASS is used to quantify how variant the frequencies are in a song, in relation to the ASC. If a song has several prominent frequencies which are vastly different from each other, the spectral spread will be very large [10]. In our project, we used ASC and ASF as features in our project, and the combination of them help to identify the tempo of a song.

Das et al. used a similar approach to our music classification task to develop a model to classify urban noises such as a siren or a jackhammer [11]. They created a model that combined both a convolutional neural network (CNN) and a long short-term memory (LSTM) model and compared their performance to popular pre-built models. While we have come up with a different set of features, the architecture of theirs is going to be heavily used as a basis for our own model's architecture. One of the features they used was Chroma STFT, which we also used. This represents the intensity of different pitch classes, or notes, of a sound at a particular time.

METHODOLOGY

1) Dataset

The music dataset I used for model training was the GTZAN Dataset [1]. This dataset provides 30 second clips of different songs for 10 unique genres, all which are free-to-use and publicly available. The 10 different genres are Blues, Classical, Country, Disco, Hip-Hop, Jazz, Metal, Pop, Reggae, and Rock. There is a total of 1000 songs, as there are 100 song clips for each genre. They also provide a list of features, similarly extracted from LibROSA, but their features were not used. The path to a song is defined as '/Data/genres_original/{genre}/{genre}.000{00-99}.wav'. One minor caveat to be mentioned is that the 'jazz.00054.wav' file was corrupted, so it was not used in training the model.

2) *Audio Features*

a. *Chroma STFT*

The first feature extracted from the songs was the Chroma STFT. STFT stands for short-term Fourier transform, and maps a signal into a function of time and frequency [12]. We can use this on a song and then use this to generate a chroma feature matrix that will represent the frequency as a musical pitch, or note (C, D, E, F…), and portray music notes that dominate the sound with a brighter light to represent what percentage of the sound is made up by frequencies of that musical note. We then pass both the mean and variance of this Chroma STFT into our model as a feature.

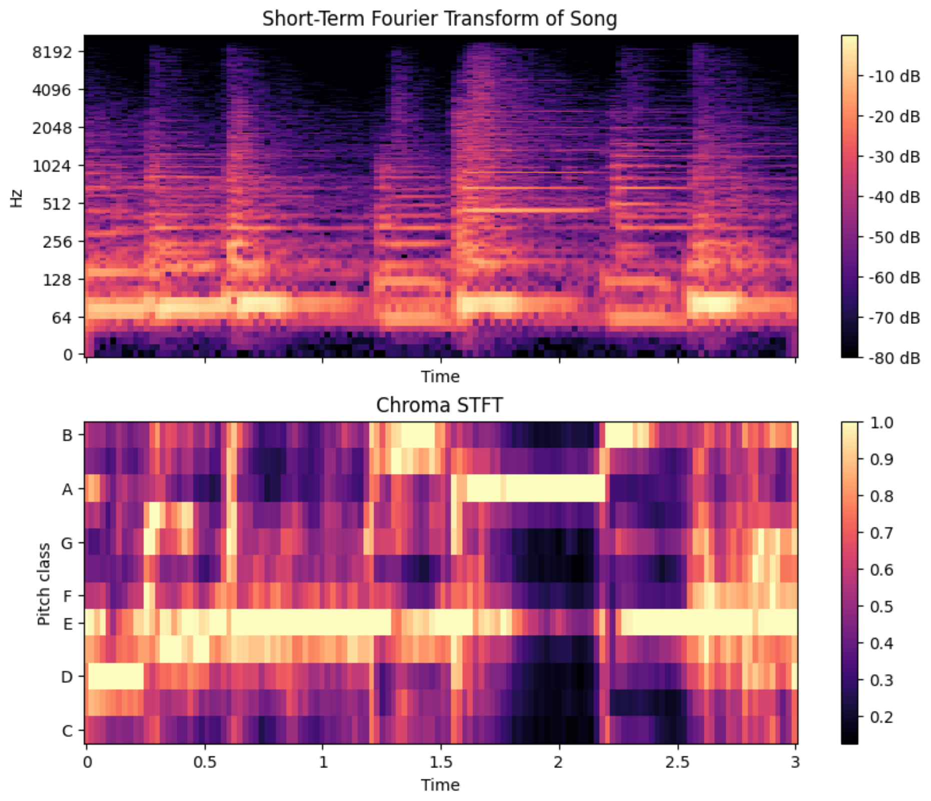
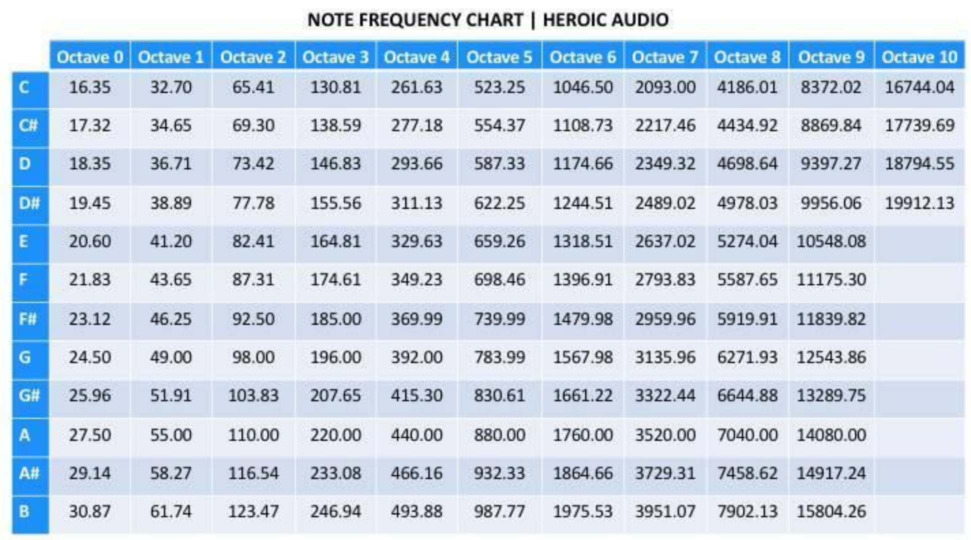


Figure 1. A STFT representation of a 3-second song clip (upper) and a Chroma STFT representation of the same sound clip – The first 3 seconds of blues.00003.wav.

Note that a higher frequency does not necessarily correlate to a higher pitch class, because of the presence of octaves. There are multiple frequency ranges for one particular note, each capturing the note at a different octave. The Chroma STFT does not capture octaves, but only the type of note [13].



NOTE FREQUENCY CHART | HEROIC AUDIO

	Octave 0	Octave 1	Octave 2	Octave 3	Octave 4	Octave 5	Octave 6	Octave 7	Octave 8	Octave 9	Octave 10
C	16.35	32.70	65.41	130.81	261.63	523.25	1046.50	2093.00	4186.01	8372.02	16744.04
C#	17.32	34.65	69.30	138.59	277.18	554.37	1108.73	2217.46	4434.92	8869.84	17739.69
D	18.35	36.71	73.42	146.83	293.66	587.33	1174.66	2349.32	4698.64	9397.27	18794.55
D#	19.45	38.89	77.78	155.56	311.13	622.25	1244.51	2489.02	4978.03	9956.06	19912.13
E	20.60	41.20	82.41	164.81	329.63	659.26	1318.51	2637.02	5274.04	10548.08	
F	21.83	43.65	87.31	174.61	349.23	698.46	1396.91	2793.83	5587.65	11175.30	
F#	23.12	46.25	92.50	185.00	369.99	739.99	1479.98	2959.96	5919.91	11839.82	
G	24.50	49.00	98.00	196.00	392.00	783.99	1567.98	3135.96	6271.93	12543.86	
G#	25.96	51.91	103.83	207.65	415.30	830.61	1661.22	3322.44	6644.88	13289.75	
A	27.50	55.00	110.00	220.00	440.00	880.00	1760.00	3520.00	7040.00	14080.00	
A#	29.14	58.27	116.54	233.08	466.16	932.33	1864.66	3729.31	7458.62	14917.24	
B	30.87	61.74	123.47	246.94	493.88	987.77	1975.53	3951.07	7902.13	15804.26	

Figure 2. A Note Frequency Chart, representing the frequency of notes at different octaves [14].

b. Spectral Centroid

As mentioned in the Background section, the Audio Spectrum Centroid gives the frequency point where most of the energy is centered around. In our case, energy is the decibel level. To determine this, all of the frequencies are given a weight based on their energy level, and the average is found from this to determine the spectral centroid. A higher frequency spectral centroid means a more higher-pitched and bright sound, where a lower spectral centroid means a lower-pitched and deeper sound [15].

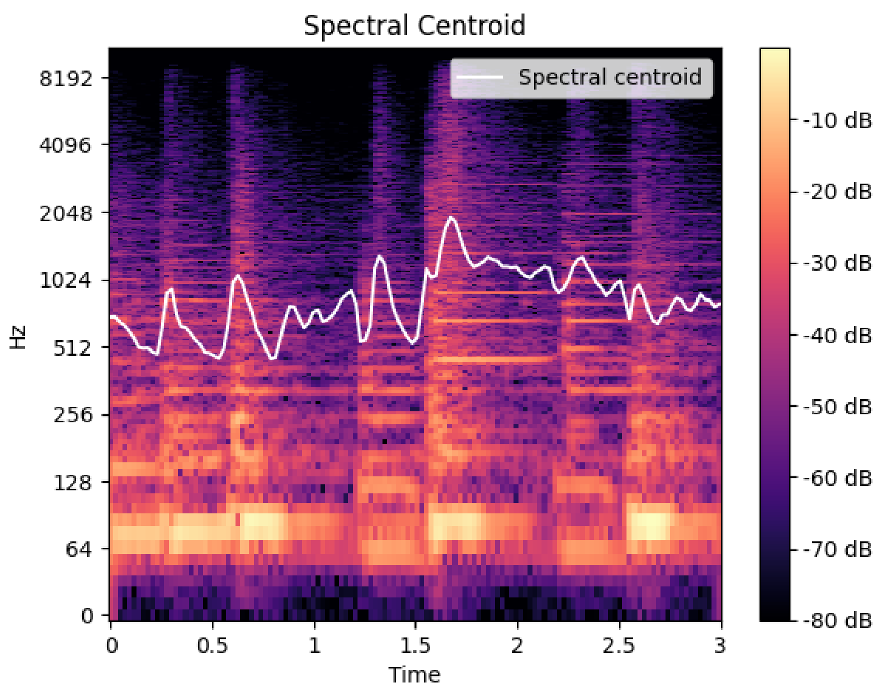


Figure 3. The spectral centroid at each point in time of the first 3 seconds of sound clip blues.00003.wav.

c. *Spectral Flatness*

As mentioned in the Background section, the spectral flatness measures how even the distribution of energy is. For our case looking at sound, a purely flat sound would be all frequencies having equal decibel levels. Whereas if a sound only has a few frequencies dominating, it would be a much less flat spectrum. The way this is measured is the ratio of the geometric mean of the spectrum to the arithmetic mean of the spectrum. A very high spectral flatness would be closer to 1 [16].

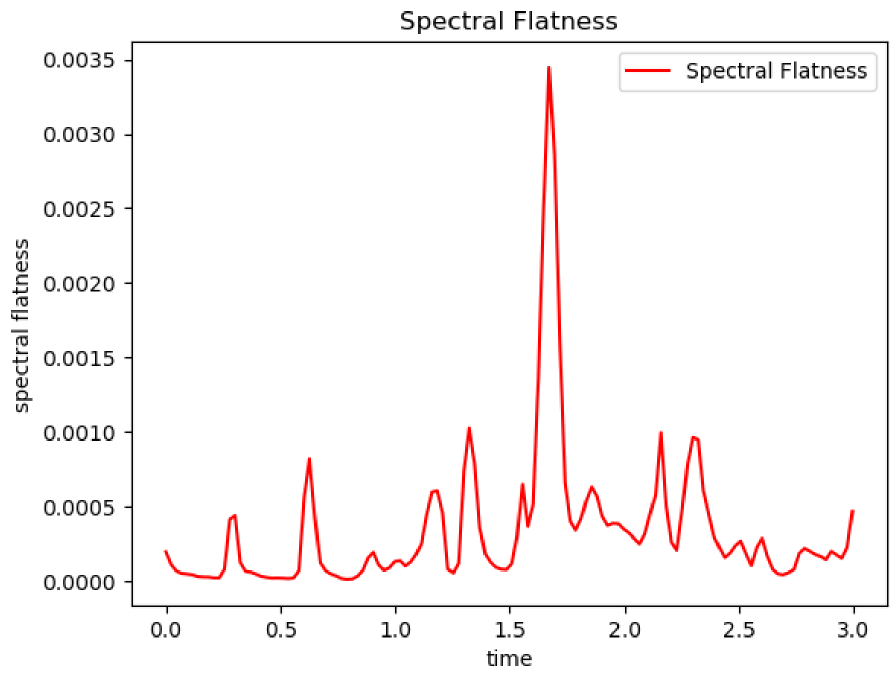


Figure 4. The spectral flatness at each point in time of the first 3 seconds of sound clip blues.00003.wav.

4

d. Roll Off Frequency

The roll-off frequency measures the frequency point in which all the combined energy below that point is equal to the threshold – a percentage of the entire energy. For our project, we used a roll off threshold of 0.85, which means that we will be finding the point on the spectrogram that all the energy below it will sum to 85% of the total energy [17].

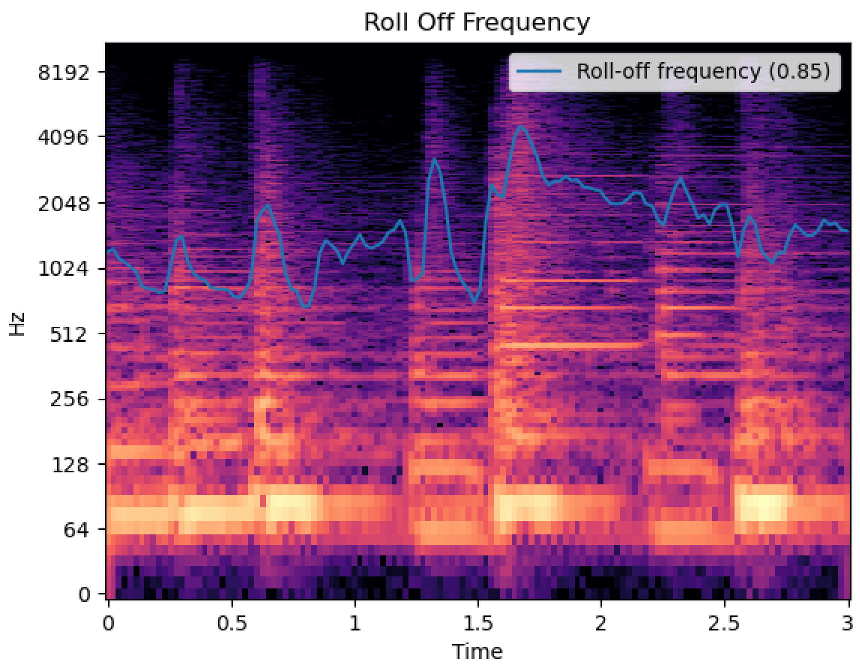


Figure 5. The Roll Off Frequency of the first 3 seconds of sound clip blues.00003.wav, with the threshold set to 0.85.

e. MFCCs

The Mel Frequency Cepstral Coefficients (MFCC) are widely used in audio processing to detect the acoustic properties of a sound, beyond that of just pitch or loudness. For example, a trumpet and a piano playing the same note at the same octave and at the same sound level might have a similar pitch and decibel level, but their sounds are very different from one another. MFCCs are used in measuring the properties that make a sound unique, such as the timbre of a sound. A high number of coefficients used will represent the finer and more precise aspects of the sound, whereas a low number of coefficients will represent just the most important spectral information. In our project, we used 20 coefficients.

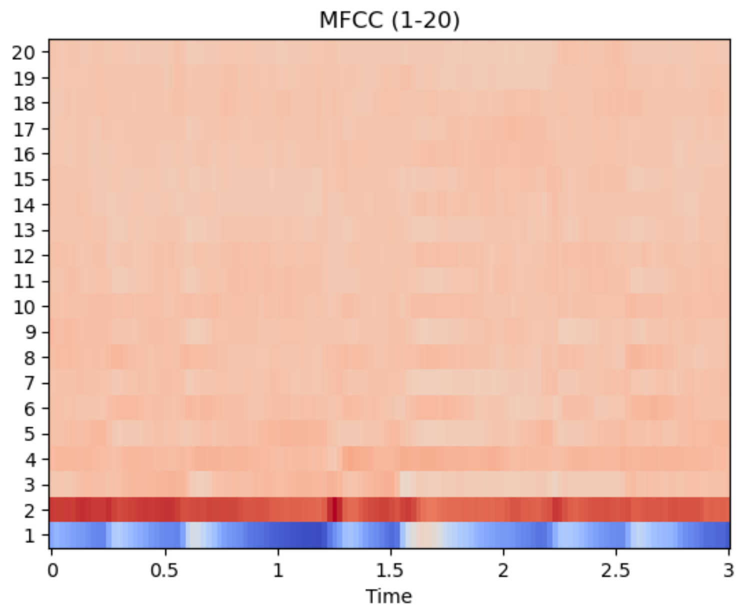


Figure 6. A graph representing the 20 MFCC coefficients of the first 3 seconds of the sound clip blues.00003.wav.

3. *Model Setup*

For our project, we decided to use all the features described in the above section. Initially, we set out to capture the temporal information of each of these features, but quickly realized this was unfeasible for the scope of our project. Each 30 second song clip had ~661,794 frames of sound associated with it. Given that there were ~1000 different song clips in the dataset and we wanted to analyze 24 different features, it was clear that the amount of computation and storage space needed to train a temporal model such as a convolutional neural network or a recurrent neural network would be immense [19]. Instead, we decided to split each 30 second song into ten 3-second sound clips, and take the average and variance of each of the features. For extracting all features and the visualization of them, code was used from the libROSA documentation [20].

First, we separated each of the songs into ten 3-second sound clips. To do this, we calculated how many frames was in a 3 second clip using the sampling rate of the audio. We found a sampling rate of 22050, or 22.05kHz. This means that the audio is sampled 22050 times per second, so each second would have that many sound frames [21]. Thus, we split the songs into 10 66150 frame sound clips. Next, we used the libROSA library to get the Chroma STFT, the Spectral Centroid, Spectral Flatness, Spectral Rolloff, and 20 MFCC signals. For each of these 24 features, we found the mean and variance for each one and stored this as a CSV file.

We then set out to create a custom model that will predict the genre of a song given the list of 48 features. First, we separated the data into training and testing sets. We isolated the 'label' column from the CSV file to use our labels. For labels, we then used a label encoder to transform the genre strings into a corresponding encoded representation (0, 1, 2…9). Then, we used sklearn and its provided train_test_split() function to separate the data into training and testing data. We decided that 80% of the data would be for training, and then 20% would be used just for evaluating the model [23]. Making sure to include this validation set will help detect overfitting on the model – which is where a model becomes too dependent on training data that it can't make predictions on new data. After that, we converted all the input (X) and label (y) data to tensors, and fed them into PyTorch's DataLoader so that they could be batched [24]. We used a batch size of 64, meaning that 64 examples would be run simultaneously.

Since the temporal data was lost from just storing the statistics of each song, for our model we decided to use a fully connected network, or a Multilayer perceptron network [22]. This will take an input vector of a certain length, pass it through multiple hidden layers of different sizes, applying an activation layer after each hidden layer. Finally, there will be an output layer with x number of features, where x is the number of possible classes, and each feature corresponds to the logit of a particular class. These logits can then be transformed to find the probability of each corresponding class being the correct classification. In our case, we have 48 input features, and 10 possible classes (genres) that a song can be defined as.

For our hidden layers, we decided to choose hidden linear layers of size 128, 64, and 32. For our activation function, we chose the ReLU function. This is defined as $f(x) = \max(0, x)$, and will return the variable if it is greater than 0, or else it will return 0 [25]. This introduces non-linearity to our model, and helps to mitigate vanishing gradients – when gradients start shrinking through backpropagation [26]. Thus, our model's architecture was defined as:

$$input \rightarrow Linear\ (128) \rightarrow ReLU \rightarrow Linear\ (64) \rightarrow ReLU \rightarrow Linear\ (32) \rightarrow ReLU \rightarrow Linear\ (\#\ classes)$$

Where our input was initially defined as 48 features, and our '# classes' was 10.

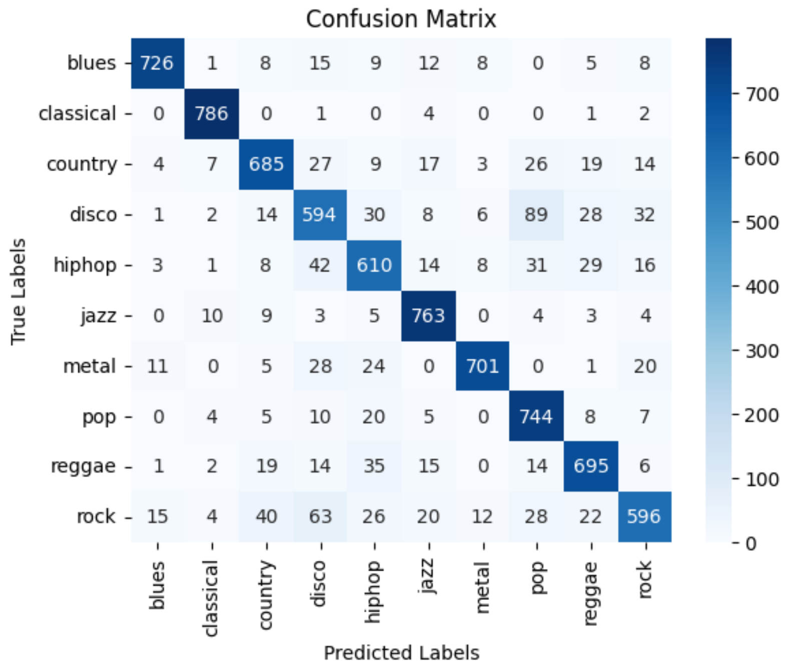
Finally, we set up the process to train this model. The documentation from PyTorch was used heavily in setting up the training [27]. For the optimizer, we used ADAM with a learning rate of 0.001, and for the loss function we used Cross Entropy Loss [28 29]. For our number of epochs, we used 250.

<u>RESULTS</u>

A. Multi-Layer Perceptron Performance

When initially running our model with all 48 features (24 features, with each feature having a mean and variance), we found that our performance was incredibly suboptimal. Our model was converging to ~10% success rate, which is roughly equal to randomly guessing the genre every time. Instead, we tried running the model with 24 features, separately running it on just the means and just the variances. While the model still achieved about 10% accuracy with the variance feature set, the model was able to achieve 86.42% accuracy with the mean feature set. Therefore, in our final model architecture we only pass the mean feature set into the model.

To find which genres the model was best and worst at classifying, we used the provided confusion_matrix function from the sklearn.metrics module [30].



	blues	classical	country	disco	hiphop	jazz	metal	pop	reggae	rock
blues	726	1	8	15	9	12	8	0	5	8
classical	0	786	0	1	0	4	0	0	1	2
country	4	7	685	27	9	17	3	26	19	14
disco	1	2	14	594	30	8	6	89	28	32
hiphop	3	1	8	42	610	14	8	31	29	16
jazz	0	10	9	3	5	763	0	4	3	4
metal	11	0	5	28	24	0	701	0	1	20
pop	0	4	5	10	20	5	0	744	8	7
reggae	1	2	19	14	35	15	0	14	695	6
rock	15	4	40	63	26	20	12	28	22	596

Figure 7. A confusion matrix of our Multi-layer Perceptron Model.

From this, we can see that the model performed worst on the genres of rock, hip-hop, and disco. The most common error is when a song is disco, and the model predicts its genre as pop. If we were to look for ways to improve the performance of this model, we would study the genres that the model performed worst on and try to find some trend or additional musical feature that could better identify each.

B. Other Model Performances

We wanted to see how our model performed against default, pre-built models provided by sklearn. The first model we decided to compare against was Random Forest Classifier. This model builds many tree-structure classifiers at the basis of a training sample set, with a random subset of features being considered. For classification, the collection of trees ‘vote’ on which class they think is the correct class, and majority vote is used to determine which class it outputs [31]. The second model we compared our model to was the Gradient Boosting Classifier. This model is similar to the Random Forest Classifier, but instead of using many different trees to vote on a class, it instead builds its trees iteratively, with each tree improving on the previous model. The final tree is created by weighing each tree based on their performance and summing them [32].

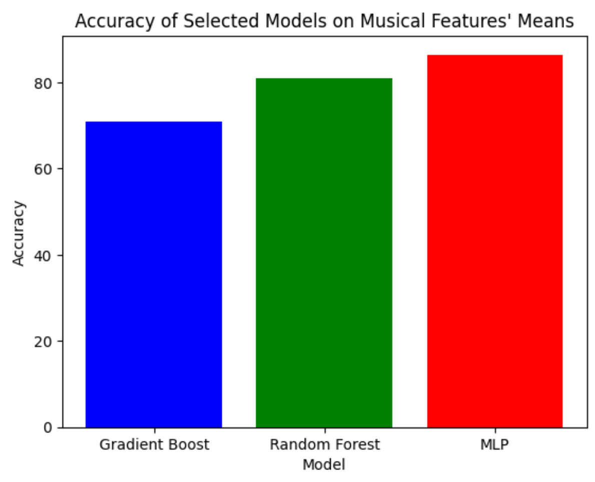


Figure 8. Comparison of the performance of models trained on the means of musical features.

The gradient boost model managed an accuracy of 70.80%, and the random forest model received an accuracy of 80.92%.

<u>CONCLUSION & NEXT STEPS</u>

While the scope of the project ended up being smaller than we initially set out to achieve, we were able to extract musical features from audio files, build and train a custom neural network on this data, and achieve a notable improvement over several default models provided by sklearn. What must be noted is that all 3 models' performances can likely be improved by adjusting their hyperparameters. Furthermore, though the variance of the musical features caused a large dip in performance, it is still useful data, and finding a way to incorporate this data will likely be our next step.

Another feature in our project that we were not able to include due to its large complexity was the temporal data of the songs. We instead used aggregated statistics, considering the mean and variance of 3-second clips of each song. Because of the large computation and space requirements needed to train our model while considering every frame of the song and its connections to previous frames, this was unfeasible to be included in our project. An initial idea we may try to introduce temporal data while keeping the complexity relatively small is to skip frames. For example, we could take every 100th frame of a song and turn this into a new song with effectively 1/100th of the sample rate.

Furthermore, one of our limitations was that the premise of our model defines that each song is only available to be assigned to 1 genre, and there are only 10 genres available. Some of the songs may belong to multiple of the 10 genres we defined or may belong to none of them. In addition, there are hundreds of genres that we did not define in our project. Perhaps a more accurate structure of our model would be to define an independent probability for a song to belong to a genre. If a genre's probability exceeds a certain threshold, we could assign the song to that genre. This would allow a song to belong to none or multiple genres. This would require a lot more work in labeling songs, however.

Finally, our model was only trained and evaluated on one song dataset [1]. Perhaps there was some bias introduced in this dataset's songs that introduce a commonality in each genre that might not exist in other datasets. A next step to evaluate the robustness of our model would be to evaluate our model on other datasets, or if it is able to correctly identify the genre of songs that are not copyright-free.

Overall, this project achieved the goal of learning to create a music classification model by extracting musical features using libROSA. Our custom model architecture managed a high accuracy of ~86%, which is significantly more than the performance of several default model architectures provided by sklearn.

REFERENCES

* Indicates a non-scholarly source

*[1] GTZAN Dataset. (2023). *Kaggle.* Retrieved from https://www.kaggle.com/datasets/andradaolteanu/gtzan-dataset-music-genre-classification.

[2] McFee, Brian, Colin Raffel, Dawen Liang, Daniel PW Ellis, Matt McVicar, Eric Battenberg, and Oriol Nieto. "librosa: Audio and music signal analysis in python." *In Proceedings of the 14th python in science conference*, pp. 18-25. 2015.

[3] Seashore, C. E. (1937). The Psychology of Music. *Music Educators Journal*, *23*(4), 30-33.

[4] Barenboim, G., Debbio, L.D., Hirn, J. *et al.* Exploring how a generative AI *interprets* music. *Neural Comput & Applic 36*, 17007–17022 (2024).

*[5] Semanick, Alex. (2024). *How AI is transforming the creative economy and music industry*. OHIO News. Obtained from https://www.ohio.edu/news/2024/04/how-ai-transforming-creative-economy-music-industry.

[6] Oord, A.V., Dieleman, S., & Schrauwen, B. (2014). Transfer Learning by Supervised Pre-training for Audio-based Music Classification. *International Society for Music Information Retrieval Conference*.

[7] Jia, Xiaosong, A Music Emotion Classification Model Based on the Improved Convolutional Neural Network, *Computational Intelligence and Neuroscience*, 2022, 6749622, 11 pages, 2022.

[8] Tiwari, V. (2010). MFCC and its applications in speaker recognition. *International journal on emerging technologies, 1*(1)*,* 19-22.

[9] A. Lazaro, R. Sarno, R. J. Andre and M. N. Mahardika, "Music tempo classification using audio spectrum centroid, audio spectrum flatness, and audio spectrum spread based on MPEG-7 audio features," *2017 3rd International Conference on Science in Information Technology (ICSITech)*, Bandung, Indonesia, 2017, pp. 41-46.

[10] Giannakopoulos, T., & Pikrakis, A. (2014). Chapter 4 - Audio Features. In T. Giannakopoulos & A. Pikrakis (Eds.), *Introduction to Audio Analysis* (pp. 59–103).

[11] J. K. Das, A. Ghosh, A. K. Pal, S. Dutta and A. Chakrabarty, "Urban Sound Classification Using Convolutional Neural Network and Long Short Term Memory Based on Multiple Features," *2020 Fourth International Conference On Intelligent Computing in Data Sciences (ICDS)*, Fez, Morocco, 2020, pp. 1-9.

[12] Yang, Y., Peng, Z., Zhang, W., & Meng, G. (2019). Parameterised time-frequency analysis methods and their engineering applications: A review of recent advances. *Mechanical Systems and Signal Processing, 119***,** 182–221.

*[13] Ultimate Guide to Musical Frequencies. *iDrumTune*. Obtained from https://www.idrumtune.com/ultimate-guide-to-musical-frequencies/.

*[14] Understanding Note Frequency Charts (And Why You Should Be Using One). *Produce Like a Pro*. Obtained from https://producelikeapro.com/blog/note-frequency-chart/.

*[15] Delisle, Julie (2019). Spectral centroid. *Timbre and Orchestration Resource.* Obtained from https://timbreandorchestration.org/writings/timbre-lingo/2019/3/29/spectral-centroid.

*[16] Cook, John (2021). Spectral flatness. *John D. Cook Consulting.* Obtained from https://www.johndcook.com/blog/2016/05/03/spectral-flatness/.

*[17] Mahajan, Kaavya (2023). Extracting audio features using Librosa. *Medium.* Obtained from https://kaavyamaha12.medium.com/extracting-audio-features-using-librosa-3be4ff1fe57f.

[18] Ghosal, Dr. Arijit & Saha, Sanjoy & Dhara, Bibhas & Chakraborty, Rudrasis. (2012). Music Classification based on MFCC Variants and Amplitude Variation Pattern: A Hierarchical Approach. *International Journal of Signal Processing, Image Processing and Pattern Recognition. 5*. 131-150.

[19] Ketkar, N., & Moolayil, J. (2021). Convolutional Neural Networks. In *Deep Learning with Python: Learn Best Practices of Deep Learning Models with PyTorch* (pp. 197–242).

*[20] Feature extraction. *Librosa.* Obtained from https://librosa.org/doc/main/feature.html.

*[21] Hahn, Michael (2024). What is Sample Rate? Audio File Quality Explained for Producers. *LANDR Blog.* Obtained from https://blog.landr.com/sample-rate/.

[22] Popescu, M. C., Balas, V. E., Perescu-Popescu, L., & Mastorakis, N. (2009). Multilayer perceptron and neural networks. *WSEAS Transactions on Circuits and Systems*, *8*(7), 579-588.

*[23] How to split the Dataset With scikit-learn's train_test_split() Function. *Geeks for Geeks.* Obtained from https://www.geeksforgeeks.org/how-to-split-the-dataset-with-scikit-learns-train_test_split-function/.

*[24] Datasets & DataLoaders. *PyTorch.* Obtained from https://pytorch.org/tutorials/beginner/basics/data_tutorial.html.

*[25] Krishnamurthy, Bharath. An Introduction to the ReLU Activation Function. *Built In.* https://builtin.com/machine-learning/relu-activation-function.

*[26] Vanishing Gradient Problem in Deep Learning: Understanding, Intuition, and Solutions. *Medium*. https://medium.com/@amanatulla1606/vanishing-gradient-problem-in-deep-learning-understanding-intuition-and-solutions-da90ef4ecb54.

*[27] Training with PyTorch. *PyTorch*. https://pytorch.org/tutorials/beginner/introyt/trainingyt.html.

*[28] Vishwakarmk, Neha (2024). What is Adam Optimizer? *Analytics Vidhya*. Obtained from https://www.analyticsvidhya.com/blog/2023/09/what-is-adam-optimizer/.

*[29] Hughes, Chris (2024). A Brief Overview of Cross Entropy Loss. *Medium*. Obtained from https://medium.com/@chris.p.hughes10/a-brief-overview-of-cross-entropy-loss-523aa56b75d5.

*[30] confusion_matrix. s*cikit-learn*. Obtained from https://scikit-learn.org/dev/modules/generated/sklearn.metrics.confusion_matrix.html.

[31] Liu, Y., Wang, Y., & Zhang, J. (2012). New machine learning algorithm: Random forest. In *Information Computing and Applications: Third International Conference, ICICA 2012, Chengde, China, September 14-16, 2012. Proceedings 3* (pp. 246-252). Springer Berlin Heidelberg.

[32] Natekin, A., & Knoll, A. (2013). Gradient boosting machines, a tutorial. *Frontiers in neurorobotics*, *7*, 21.