6202 – Machine Learning 1
Final Project Proposal
Bradley Reardon, Salim Haruna
6/13/21

Music Genre Classification

Table of Contents

Introduction

Picture this: you're sitting at a bar having a great time with your friends and this amazing song starts playing. It is nothing like you've ever heard before. You think to yourself, "I wonder what genre of music this is?" but you're not even sure where to begin searching for that information. Well, worry no further. With the help of our music genre classifier, all you need is the name of the song (.mp3 file in the case of this project demo) and a few moments to spare. With the click of a button, you can find out the genre of any song you come across from here on out.

We chose to work on music genre classification as we feel it is increasingly difficult for music streaming platforms to properly classify all of the unique genre-bending music being released today. We want to help provide a solution for this issue and believe this can be solved with the help of machine learning, and more specifically with the help of a neural network. Using a training dataset found on Kaggle, we trained a multilayer perceptron classification network to classify songs originally stored as an .mp3 file. Additionally, we created an interactive graphical user interface that allows a user to select the desired .mp3 file they would like to classify into a music genre.

Dataset Overview

The training dataset we chose to work with is a music genre classification dataset found on Kaggle. This dataset contains 27 features (filename, rmse, chromasft, central_spectroid, central_bandwidth, rolloff, zero_cross, and 20 mfcc for the various frequencies within each .wav (Mel frequency, the way in which humans perceive sound)) and the label which contains 10 classes (blues, classical, country, disco, hiphop, jazz, metal, pop, reggae, rock). There are 1000 rows, each row containing the feature and target data of a single .wav file. The target balance was perfect with each class comprising 10% of the dataset *(Figure 1)*.

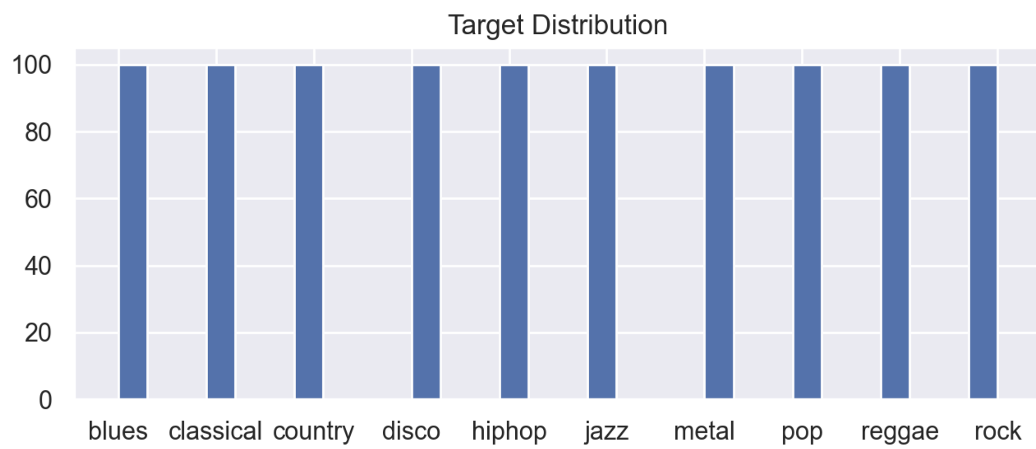

Figure 1: Target Distribution

Network and Models

The artificial neural network we chose to use is the MLPClassifier from the sklearn package. A multilayer perceptron (*Figure 2*) is a perceptron model that contains multiple layers that begins with input data being fed into the first layer, and each succeeding layer being fed the output from the preceding layer as input data. The advantages of using an artificial neural network over traditional machine learning models are their adaptability of coefficients through backpropagation, their ability to approximate unknown functions, and their ability to deal with every type of problem (regression, classification, clustering, time-series). As stated in this section of Advances in Computers by S. Abirami and P. Chitra, "The backpropagation algorithm is a form of steepest-descent algorithm in which the error signal, which is the difference between the current output of the neural network and the desired output signal, is used to adjust the weights in the output layer, and is then used to adjust the weights in the hidden layers, always going back through the network towards the inputs. Thus, although the neural network operates on the input signals to give an output in an entirely feedforward way, during learning, the resulting error is propagated back from the output to the input of the network to adjust the weights.”

We conducted performance optimization of our network with the help of [GridSearchCV] which, as described via the documentation page, is an “exhaustive search over specified parameter values for an estimator.” We included all of the possible parameter options within our hyperparameter space (barring all possible hidden-layer sizes considering there are infinite possibilities) and iterated through each parameter combination. This is a very computationally expensive approach, so in order to limit the number of times we needed to run this code, we used the *best_params_* attribute to return the parameter combination that resulted in the best performance. The following code shows the optimal MLPClassifier parameters for our specific project:

```
MLPClassifier(
      hidden_layer_sizes=(60,100,60),max_iter=10000,learning_rate='invscaling',
      solver="adam", activation='tanh', alpha= 0.0001
)
```

In order to determine if our artificial neural network would outperform a more traditional machine learning model, we also ran our data through a decision tree *(Figure 3)* model using DecisionTreeClassifier from the sklearn package. In machine learning, a decision tree is a type of supervised learning used for classification where the data is continuously split into categories based on feature parameters until the model has enough information to classify the input data.

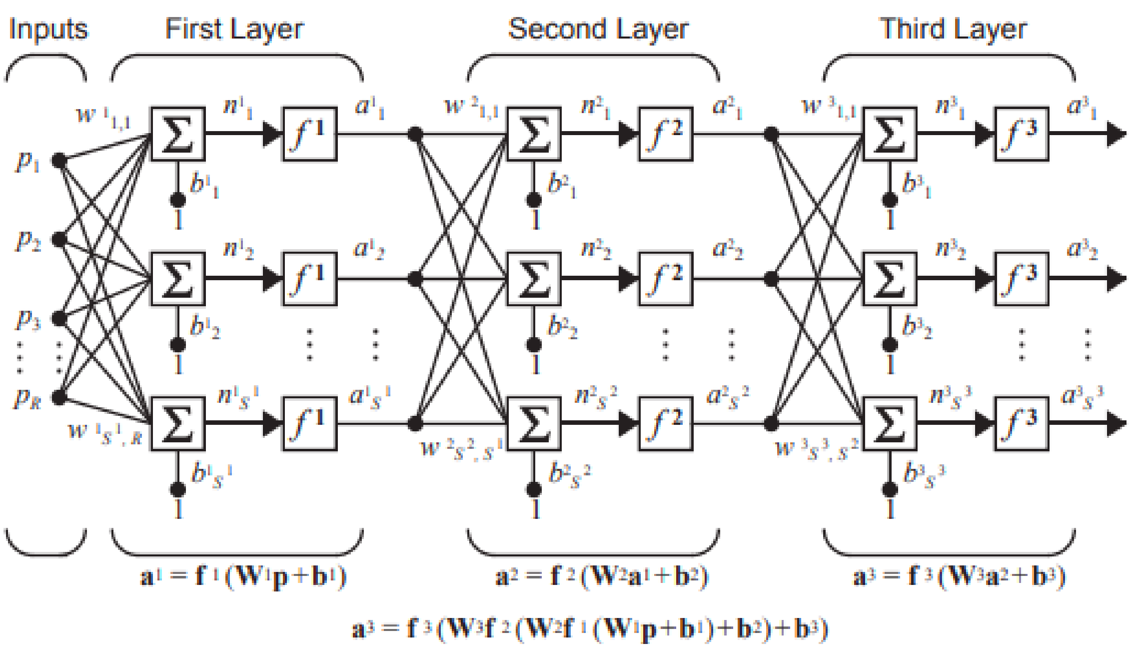

Figure 2: Multilayer Perceptron Example

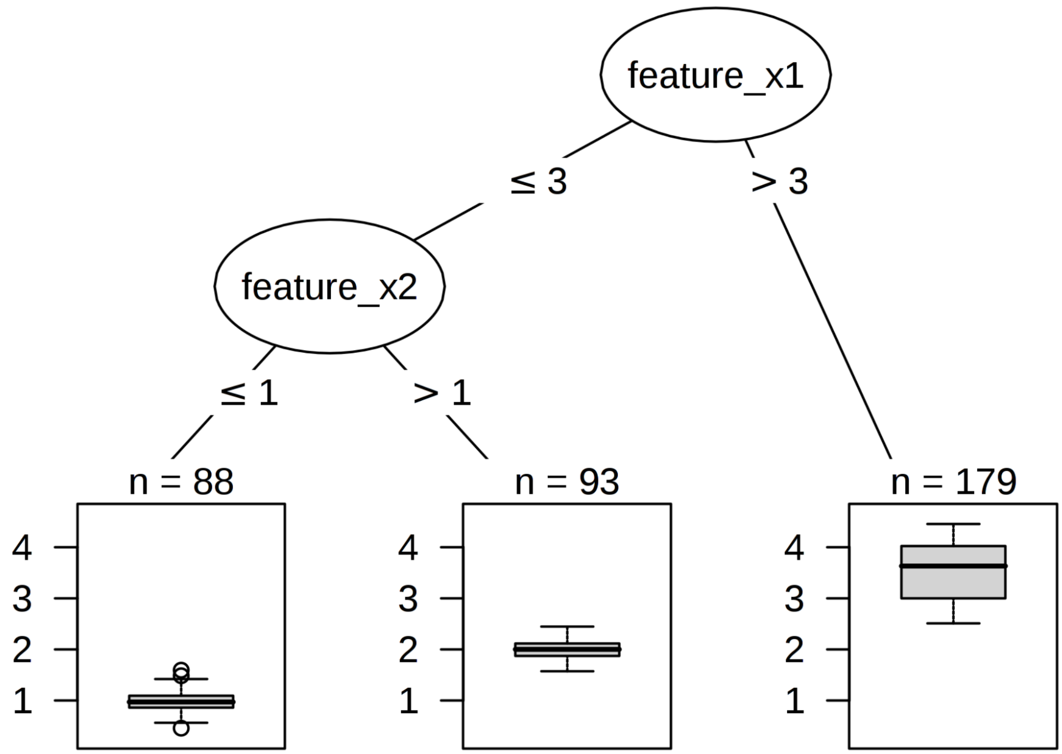

Figure 3: Decision Tree Example

Experimental Setup

Since the target data in our dataset was originally in string form, we used a label encoder to transform the targets into integer form. Additionally, we standardized our feature data to ensure all features were on the same scale and were cross-comparable. Using the *train_test_split* function from the sklearn package, we split the music genre dataset into train and test sets with a 4:1 split respectively. After training and testing our network with the optimal features, we were able to consistently receive both F1 and accuracy scores of ~65%. In order to test our model on unseen data, we needed to first convert the chosen .mp3 file into a .wav file and extract the features from said .wav file. The functions to do so, *convert_mp3_to_wave* and *wav_features*, require the use of the librosa and pydub packages and can be found in the Code folder of our github repository. The *wav_features* function creates a new dataframe consisting of the same features as the training dataset. Each row in this new .wav dataset represents a single byte of the .wav file wavelength, and the number of rows varies depending on the wavelength of the .mp3 that was converted. The non-zero mean of each feature is calculated and the single row dataset is then fed into the network and the row receives its own class label that determines the .mp3 genre as predicted by the network.

Using the [PyQT5] package, we created a GUI that allows the user to select their desired .mp3 file they want to determine the music genre of. To do this, run the [main.py] and wait for the window to open. Once open, click the *Select mp3 File* which will open your directory where you can then navigate to the desired .mp3. Select the .mp3 and wait for the file to be run through the process mentioned above. Once finished, the GUI will display a classification report for both the *MLPClassifier* and *DecisionTreeClassifier,* a music genre classification, and a plot showing the monophonic waveform of the .mp3 file.

Results

In order to arrive at the final results, we must first look at the results returned from converting an .mp3 file into a .wav file and conducting feature extraction. In *Figure 4*, we see the dataset that is returned after an .mp3 file is selected and pipelined through the *convert_to_wav* and *wav_features* functions. As mentioned earlier, each row represents a single byte of the .wav file wavelength.

.	chroma_stft	rmse	spectral_centroid	spectral_bandwidth	rolloff	zero_crossing_rate	mcff1	mcff2	mcff3	mcff4	mcff5	mcff6	mcff7	mcff8	mcff9	mcff10	
...	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	-529.10089	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.
...	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	-529.10089	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.
...	0.93994	0.00000	5512.49994	3185.74988	9377.70996	0.00000	-529.10089	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.
...	0.44115	0.00000	5678.89222	3056.50287	9313.11035	0.21777	-529.10089	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.
...	0.70805	0.00000	5361.18569	3064.92454	9054.71191	0.38281	-529.10089	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.
...	0.82350	0.00001	4802.39228	3216.80506	8796.31348	0.46826	-529.10089	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.
...	0.60006	0.00002	4318.46333	3309.27007	8559.44824	0.48682	-529.10089	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.
...	0.79990	0.00004	3483.50141	3408.31884	7988.81836	0.27490	-529.10089	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.
...	0.62671	0.00007	2620.33720	3294.50589	7202.85645	0.11182	-528.79077	0.43820	0.43731	0.43582	0.43375	0.43108	0.42784	0.42401	0.41962	0.41464	0.
...	0.68632	0.00011	2008.32484	3077.75282	6244.62891	0.02734	-527.45721	2.31599	2.29047	2.24832	2.19012	2.11663	2.02886	1.92798	1.81534	1.69241	1.
...	0.54665	0.00015	1545.11068	2787.94296	4877.27051	0.01318	-525.89618	4.51953	4.48180	4.41934	4.33279	4.22302	4.09113	3.93844	3.76648	3.57695	3.
...	0.51823	0.00024	1318.34394	2639.79371	3919.04297	0.00879	-525.14532	5.58221	5.54691	5.48838	5.40702	5.30344	5.17839	5.03276	4.86762	4.68412	4.
...	0.64215	0.00031	994.34481	2334.38658	1421.19141	0.00879	-522.88074	8.76673	8.67737	8.52962	8.32521	8.06655	7.75666	7.39918	6.99823	6.55844	6.
...	0.45984	0.00038	884.45728	2215.99874	333.76465	0.00830	-522.16919	9.76613	9.65611	9.47423	9.22269	8.90455	8.52362	8.08448	7.59237	7.05309	6.
...	0.16414	0.00041	960.06231	2288.33100	904.39453	0.00781	-522.44226	9.37844	9.26424	9.07559	8.81493	8.48568	8.09210	7.63926	7.13295	6.57960	5.
...	0.41382	0.00043	850.20524	2146.75109	312.23145	0.00977	-521.38928	10.86087	10.72671	10.50516	10.19922	9.81307	9.35190	8.82193	8.23019	7.58455	6.
...	0.33165	0.00041	880.53545	2216.09804	312.23145	0.01172	-521.71289	10.40563	10.27851	10.06848	9.77830	9.41173	8.97354	8.46936	7.90566	7.28960	6.
...	0.62237	0.00049	799.93302	2112.65480	279.93164	0.01367	-520.50372	12.08891	11.88276	11.54484	11.08335	10.50940	9.83648	9.08005	8.25693	7.38471	6.
...	0.29923	0.00063	662.89770	1880.09483	258.39844	0.01318	-518.33258	15.12630	14.82193	14.32343	13.64364	12.79991	11.81345	10.70859	9.51196	8.25160	6.
...	0.34909	0.00077	498.01452	1595.95793	193.79883	0.01270	-516.67798	17.47368	17.19068	16.72543	16.08737	15.28937	14.34735	13.27989	12.10768	10.85308	9.
...	0.39692	0.00098	459.21371	1519.95102	183.03223	0.01074	-516.26294	18.05709	17.76353	17.28049	16.61716	15.78609	14.80286	13.68575	12.45525	11.13361	9.
...	0.45517	0.00130	404.79293	1407.08609	183.03223	0.01025	-513.96021	21.26886	20.84291	20.14599	19.19709	18.02179	16.65129	15.12132	13.47078	11.74042	9.
...	0.75818	0.00185	331.72571	1205.09499	183.03223	0.00928	-510.84933	25.59777	24.96401	23.93273	22.54013	20.83438	18.87336	16.72187	14.44866	12.12323	9.
...	0.58436	0.00213	260.29502	1018.40441	150.73242	0.00830	-508.03717	29.53022	28.76546	27.52433	25.85514	23.82194	21.50120	18.97785	16.34100	13.67950	11
...	0.38707	0.00243	266.53030	1040.25391	172.26562	0.00928	-507.24017	30.62309	29.75799	28.35791	26.48297	24.21255	21.64082	18.87155	16.01244	13.16936	10
...	0.57163	0.00269	247.28033	997.40442	150.73242	0.00781	-507.82108	29.83179	29.05579	27.79891	26.11364	24.06926	21.74797	19.24030	16.64017	14.03992	11
...	0.47392	0.00255	261.83398	1022.88742	150.73242	0.00830	-507.40305	30.41983	29.63422	28.36056	26.65031	24.57143	22.20473	19.63938	16.96825	14.28298	11
...	0.46362	0.00298	251.89269	983.44430	161.49902	0.00830	-506.61523	31.50892	30.64956	29.25799	27.39277	25.13107	22.56430	19.79288	16.92075	14.04975	11
...	0.79689	0.00321	229.15735	901.05962	161.49902	0.00830	-504.93976	33.80588	32.73438	31.00665	28.70582	25.94064	22.83844	19.53712	16.17667	12.89084	9.
...	0.40423	0.00332	228.53342	861.34349	183.03223	0.00977	-502.18408	37.58883	36.18439	33.93279	30.96045	27.43041	23.52999	19.45685	15.40474	11.55024	8.
...	0.17391	0.00378	253.37690	915.81119	204.56543	0.01074	-501.38419	38.66582	37.10230	34.59746	31.29454	27.37835	23.06115	18.56686	14.11483	9.90472	6.
...	0.25591	0.00376	246.20275	882.60088	204.56543	0.01221	-501.22665	38.89997	37.36938	34.91563	31.67654	27.82972	23.57901	19.13922	14.72063	10.51473	6.
...	0.40655	0.00367	229.98729	818.37265	204.56543	0.01172	-500.52823	39.91838	38.47433	36.14706	33.05037	29.33307	25.16954	20.74882	16.26301	11.89609	7.
...	0.59033	0.00383	241.28080	839.99425	236.86523	0.01123	-498.94885	42.07455	40.40487	37.72287	34.17213	29.93906	25.23985	20.30571	15.36751	10.64125	6.
...	0.61516	0.00427	225.87106	813.26123	236.86523	0.01123	-497.34061	44.22132	42.18718	38.95680	34.75290	29.85538	24.57425	19.22089	14.08107	9.39268	5.
...	0.41121	0.00456	211.20642	725.51225	236.86523	0.01074	-495.76105	46.42645	44.30723	40.93861	36.54876	31.42536	25.88839	20.26115	14.84257	9.88414	5.

Figure 4: .wav file dataset after conversion from .mp3

This dataset is passed through the *predict_music_genre* method (within the *predict_music_genre_window* class) where the dataset feeds into the *MLPClassifier* network and

DecisionTreeClassifier model. The GUI output, represented in *Figure 5*, returns a classification report for both classifiers, the music genre that the artificial neural network classified the .mp3 as, and the plot of the monophonic waveform representing the wavelength, amplitude, pitch and the frequency of the .mp3 file.

To our surprise, the multilayer perceptron consistently outperformed the decision tree model. On average, the *MLPClassifier* returned an accuracy score of 0.65 meaning it correctly classified the .mp3 files 65% of the time. As we can see in *Figure 5* based on the F1 scores being near or below 0.50, the *MLPClassifier* struggled with classifying some classes more than others, particularly with classifying classical, disco, pop, and reggae observations. The remaining classes were often classified with high accuracy, with F1 scores being greater than 0.70.

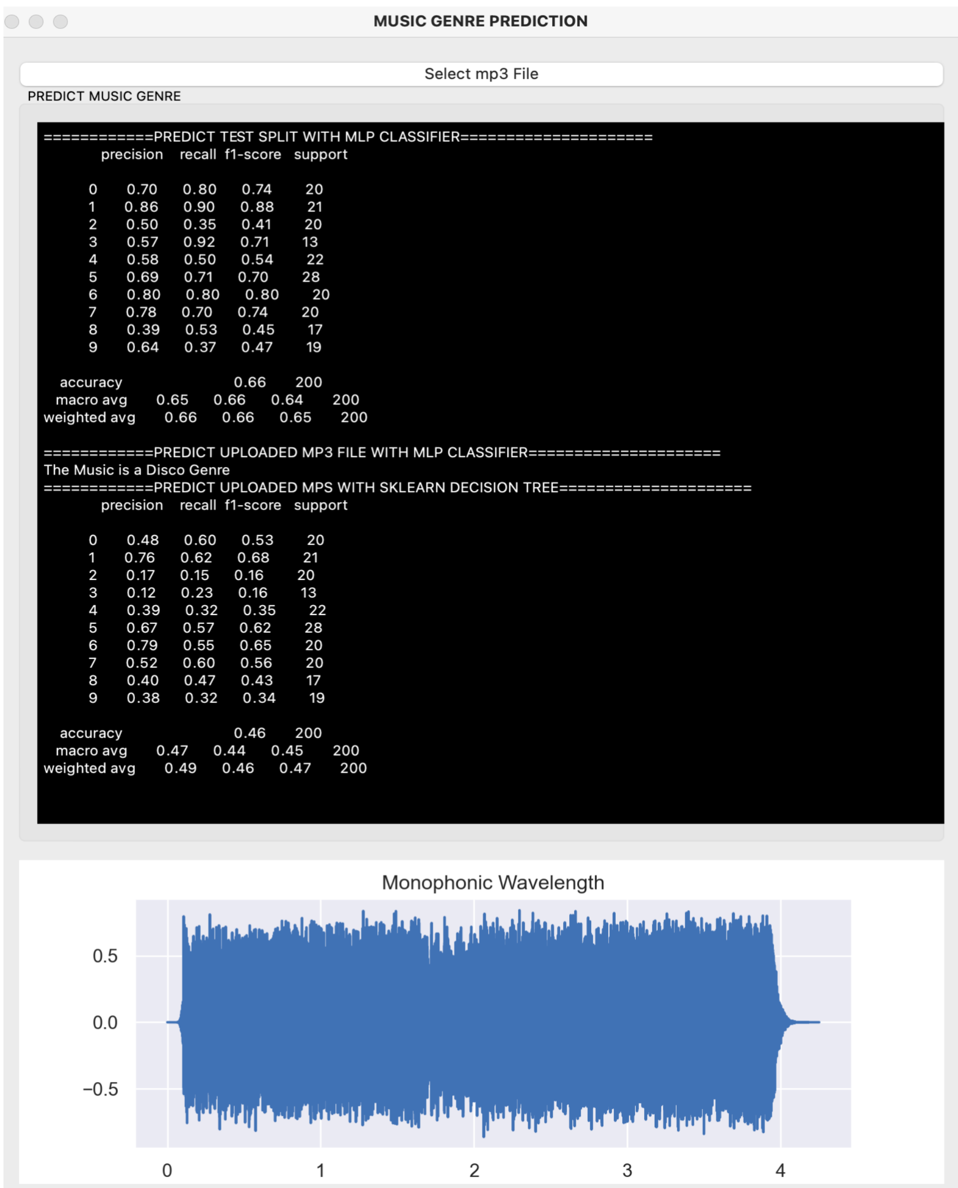

Figure 5: GUI Output Example with results

Conclusion

Given that the dataset we used to train the network and model was relatively small, we assumed that the multilayer perceptron might not outperform the decision tree model considering artificial neural networks often require larger datasets to perform well. This is why traditional models are still constantly used in industry, as many companies are not working with large enough datasets for artificial neural networks to be the optimal choice. Overall, the *MLPClassifier* outperformed the *DecisionTreeClassifier* with its accuracy score being 0.20 higher on average than that of the *DecisionTreeClassifier*. Additionally, the highest variance amongst *MLPClassifier* F1 scores was 0.40 while that of the *DecisionTreeClassifier* was 0.50, meaning that the *MLPClassifier* was more efficient in correctly classifying each class.

A difficulty we noticed is that songs might not always perfectly fit into one genre. This causes high single-label classification precision to be difficult to achieve consistently, and is due to modern music overlapping and being influenced by multiple genres. The features of a song that fits this description will fall into different genres throughout a single .mp3 file, and thus cause noise in the classification process.

A caveat we dealt with during this project is that our training data was limited to 10 core music genres. In an ideal setting, our training data would include observations from a much larger array of genres which would allow us to more precisely train a network and ultimately improve its classification ability. In the future, we would like to continue adding as much .wav genre data into our training dataset as possible to achieve higher accuracy when classifying songs into genres.

Citations

Abirami , S., & Chitra, P. (n.d.). *Multilayer Perceptron*. Multilayer Perceptron - an overview | ScienceDirect Topics. https://www.sciencedirect.com/topics/computer-science/multilayer-perceptron.

GridSearchCV. scikit. (n.d.). https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html.

Jiaaro. (n.d.). *jiaaro/pydub*. GitHub. https://github.com/jiaaro/pydub.

librosa. (n.d.). https://librosa.org/doc/latest/index.html.

MLPClassifier. scikit. (n.d.). https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html.

Natarajan, H. (2020, November 3). *Music Genre Classification*. Kaggle. https://www.kaggle.com/harish24/music-genre-classification.

Appendix