ME597 Artificial Intelligence in Thermal Systems

**Final Report Format**

Title: Distinguishing between Marine Mammals and Ocean Vessels with Artificial Intelligence/Machine Learning

Team members with affiliations: Austen Suqi, MS in Mechanical Engineering Student

Project Advisor: Prof. Veeraraghava Raju Hasti, School of Mechanical Engineering, Purdue University, West Lafayette, IN 47907

**Abstract**

Submarines and unmanned undersea vehicles (UUVs) use Sound Navigation and Ranging (SONAR) to safely navigate the ocean and avoid collision with ships and other submersibles. In order to complete the task the submarine or UUV must distinguish manmade noises from natural ocean noises to prevent collision. This work tests the ability to use eight open source Artificial Intelligence and Machine Learning (AI/ML) algorithms to build models that can accomplish the task. Overall, supervised machine learning quickly and effectively produced models that would keep submarines and UUVs aware of their environment and help avoid costly collision.

**Keywords** Machine Learning, Artificial Intelligence, Unmanned Vehicles, Submarines, SONAR

**Introduction**

Driving a ship in the ocean is similar to driving a car as there are standard rules that prevent collision and are described ad nauseum in Farwell’s Rules of the Nautical Road. However, ship drivers cannot avoid collision with objects they cannot see, like submarines. For this reason, it is the responsibility of the submarine to avoid hitting other vessels. Submarines use Sound Navigation and Ranging (SONAR) to prevent collision since there are no visual cues unless they are operating on or near the surface of the ocean. There is a team of personnel who operate the SONAR systems, primarily passive SONAR. Passive SONAR is only listening to the noises in the ocean without transmitting noise for object detection. The SONAR team trains constantly to ensure they can distinguish manmade vessels, ships and other submarines, from marine mammals since the marine mammals can avoid the submarine unlike the manmade vessels.

The use of Artificial Intelligence / Machine Learning (AI/ML) will improve safety on Submarines by aiding SONAR operators and will be the primary safety feature for Unmanned Undersea Vehicles (UUVs). The use of UUVs is rapidly expanding for research, military, and private uses. While most commercial UUVs are small and would cause minimal damage to recreational boaters, they are expensive investments. AI/ML can protect UUVs by preventing collision, similar to how SONAR operators prevent collisions on Submarines.

This research uses open source sound clips of ships, submarines, and marine mammals and develops machine learning models to distinguish between manmade and biologic noises and eventually distinguish between different marine mammals and ship types. In order to create a model to classify the data, five commonly used features are extracted from the clips: mel-frequency cepstral coefficient (MFCCs) and spectral features. These details will characterize the sounds and allow for implementation of AI/ML methods to classify them.

**Literature survey**

Artificial intelligence (AI) and machine learning (ML) are used to classify sound for many applications from music genre classification. The most well-known use of AI/ML for sound classification is music genre classification. Frederick Heerde et al compare three fuzzy rule based techniques to classify music [1]. While fuzzy rules and related techniques are not used in this study, their work shows that there are many properties of sound to evaluate as it used 245 features to describe each song [1]. In addition to music, AI is used to analyze heart and lung sounds in medicine. Rajkumar Palaniappan et al analyzes 52 scholarly articles to summarize the methods used to evaluate lung sounds, showing that this is a popular application of AI/ML [2].

Preprocessing Techniques

Rajkumar Palaniappan et al state that preprocessing the data to extract the critical characteristics of the sounds is vital in order to effectively implement AI/ML [2]. The most common preprocessing methods for respiratory sounds in the 52 reviewed articles were the autoregressive model, mel-frequency cepstral coefficient (MFCC), energy, entropy, spectral features, and wavelet [2]. The Mel Scale is described by Neha Singh to relate to human hearing as sounds appear equally different if they are the same distance apart on the scale [3]. The article continues by providing a contrast from the Hertz (Hz) scale where a difference of 500 Hz is obvious from 500 to 1000 Hz, but not from 7500 to 8000 Hz [3]. This makes the data more analogous to what humans hear.

Other preprocessing tasks can be used besides pulling specific information from the sound clips. For example, in the work by Dariusz Kucharski et al, the study of heart sounds used normalization, applied a logarithmic function, and pulled the most applicable segment of data to improve the results [4]. In another study of lung sound classification by Dalal Bardou et al, dimensionality reduction and input whitening were tested to improve the models [5]. Their work also removed repeated data and reduced dimensionality due to having a 3840-dimension vector [5].

Artificial Intelligence / Machine Learning Methods

There are many AI/ML classifiers that can be applied to sound analysis. Examples found in other bodies of work include Support Vector Machines (SVM), K-Nearest Neighbors (KNN), Gaussian, logistic regression, decision trees, Convolution Neural Networks (CNN), and Naïve Bayes.

In Dalal Bardou et al’s study, three classifiers, SVM, KNN, and Gaussian were compared against each other and against the use of a CNN. Alone, SVM was the top performer, but adding CNN allowed for the best performance [5]. Meanwhile, Husevin Coskun used wavelet transform and Mel-frequency cepstral coefficients to extract the data built SVM and Artificial Neural Network (ANN) models [6]. SVM out performed ANN with success rates of 93.3% with wavelet transform and 100% with MFCC [6]. These works show high accuracy is possible with AI/ML methods for sound classification.

**Description of the dataset**

This dataset is a compilation of audio clips found online at maritime.org and the Watkins Marine Mammal Sound Database [7], [8]. Sound clips are reviewed and shortened with the aim of having each between one and ten seconds, separating voice overs from the sounds that are being classified, and producing equal numbers of manmade and biologic noises. In order to process the data in Python, MP3 files were converted to WAV using free online software. Subsequent breakdown of the data is described in the methodology.

**Methodology**

This work is based on example codes found online as well as homepages for python libraries [9], [10]. The complete code used in this work can be found in Appendix A.

Importing the Data

To start, all of the training and test data is loaded into Google Colab in two directories – manmade noise and biologic noise. Manmade sound clips are from maritime.org and marine mammals are from the Watkins Marine Mammal Sound Database [7], [8]. This separation facilitates labeling the data for the supervised machine learning algorithms.

The data is imported into two arrays using a for loop for each directory. The Librosa library is used to extract the audio files as a floating point time series [11]. The load function in Librosa samples the audio file at discrete times based on the sampling rate it is given in Hz. In addition to the floating point time series, the amplitude of the audio file at the time points is converted to dB to support displaying the spectrogram during data visualization.

Data Visualization

Once the data is loaded into Google Colab, the waveform and spectrograms are plotted using the librosa.display library. Waveform plots display the amplitude of the sound at specific times based on the sampling rate that is used when importing the files as described above. Figure 1 and Figure 2 display waveforms for an Atlantic Spotted Dolphin and a submarine respectively. It is useful to note the discrete sounds at random amplitudes and intervals for the dolphin while the noise from the submarine appears sinusoidal in nature. Figure 3 and Figure 4 show spectrograms of a Walrus and a surface ship. Similar to the dolphin noises’ amplitudes, the walrus’s call changes frequency fluidly throughout the sound clip. Meanwhile the ship has discrete and consistent frequency bands for its noise.

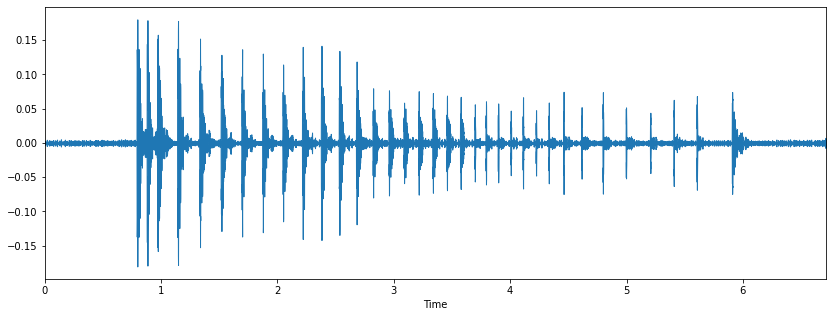


Figure 1. Atlantic Spotted Dolphin Waveform

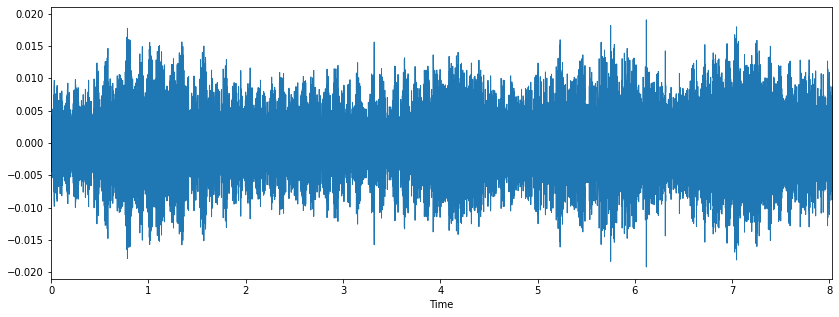


Figure 2. Submarine Waveform

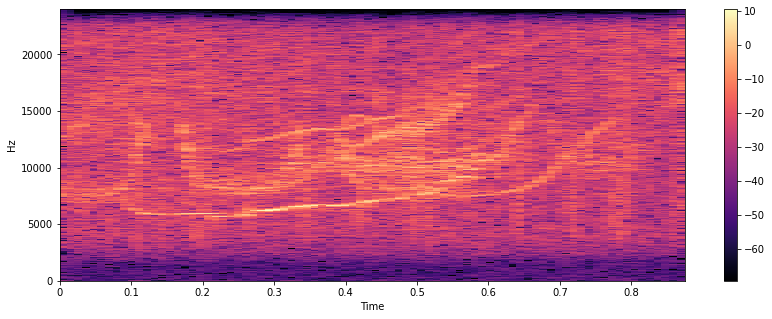


Figure 3. Walrus Spectrogram

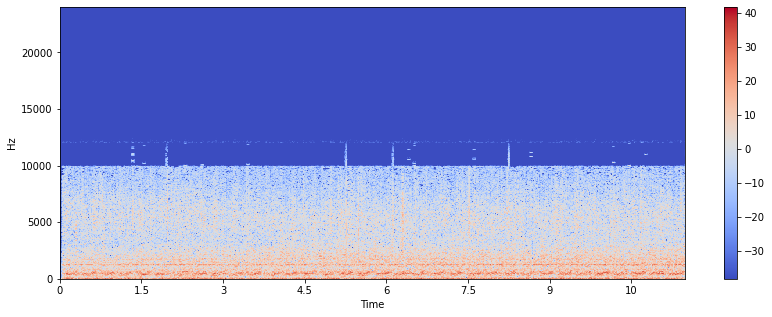


Figure 4. Ship Spectrogram

Feature Extraction / Pre-processing

In order to train the classification models audio features are extracted from the floating point time series. The features are extracted using Librosa and include spectral centroids, spectral rolloff, chromagrams, and MFCCs. These characteristics were chosen based on success in other works and simplicity to extract using the Librosa library [10], [11].

The spectral centroid is the average of the normalized magnitude of the spectrogram frequencies. Spectral rolloff is the frequency for which 85% or more of the energy is the sound clip is in this bin or bins below it. A chromagram is a feature of sound that is related to the twelve pitch classes in music [12]. MFCCs are coefficients that represent the short term power spectrum of a sound [13].

The data is saved as a csv file to verify proper alignment of the biologic and manmade data features. When reloaded, data is saved as a pandas dataframe. The data is normalized to remove any negative values for use in the AI/ML models.

The raw and normalized data are each split into training and test data with an 80/20 split. A random state of 15 is used to allow for reproducibility of the results, to keep the same split of raw and normalized data, and to ensure that the test data includes a mix of biologic and manmade data.

Model Building

The following models were chosen for their ease of implementation due to the time constraints on this work and the additional time required to acquire and prepare the data. Additionally, these classifiers have been proven to be effective in audio and other classification problems.

Support Vector Machines are high performing classifiers. The SVM aims to create a boundary between two classes using a prescribed type of curve. In this work, the polynomial and radial basis classifier kernels. The polynomial classifier used was the default cubic polynomial.

Logistics Regression is a classification algorithm used to predict binary outcomes. The algorithm creates an equation based on the inputs to predict the outcome. Unlike linear regression, the input variables can be categorical or continuous.

Keras is a deep learning model that is part of TensorFlow 2.0. Keras uses common neural network blocks such as layers, objectives, activation functions, and optimizers[14]. This algorithm can support standard, convolutional, and recurrent neural networks[14]. The algorithm can be used for classification and is built by choosing a model, adding layers for the analysis, and compiling the program. Once compiled, it is similar to other algorithms in that it can be fit to a dataset and then used to predict results from other inputs.

Naïve Bayes classifiers use statistics to minimize the probability of misclassification. In this work, the multinomial and Gaussian Naïve Bayes classifiers are used. The multinomial Naïve Bayes classifier requires all inputs to be positive; therefore, this portion of the work uses the normalized input values. Contrarily, the Guassian Naïve Bayes can work with the raw data.

K-means is an unsupervised clustering algorithm that attempts to divide data into a specified number of groups based on their proximities to each other. This work paired K-means with three different normalization techniques and Principal Component Analysis (PCA). PCA was used instead of Latent Dirichlet Allocation (LDA) based on the results displayed in Figure 5 below. As shown, LDA did not reduce the data into distinguishable groups, while PCA may be useful in some models. In future work PCA should be considered with a radial basis SVM classifier.

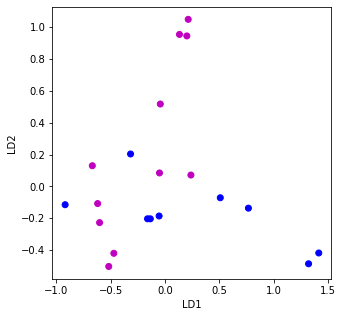
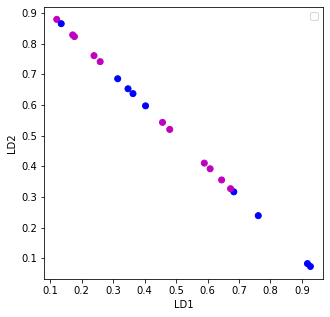


Figure 5. Reduction to two dimensions - LDA (left) and PCA (right)

K-nearest neighbors is a classifier that bases the classification of a data point on the closest known data points to it. In this work, the data was tested for one to 15 closest points and the lowest k value with the highest accuracy in training and testing was used. Figure 5 shows the optimal K value was 11.

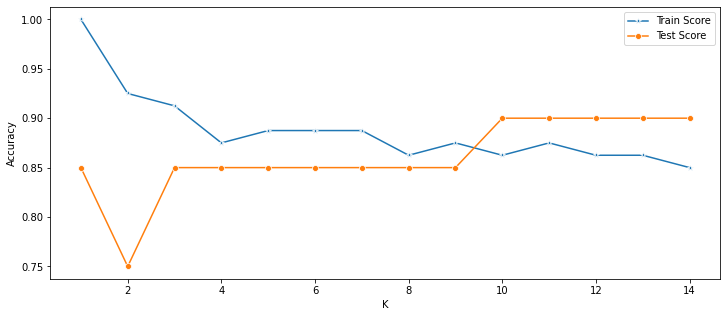


Figure 6. K-Nearest Neighbors analysis

Finally, a decision tree model is implemented. A decision tree creates multiple classification questions and uses them to determine the classification of the data. Figure 6 shows the decision tree output from the model.

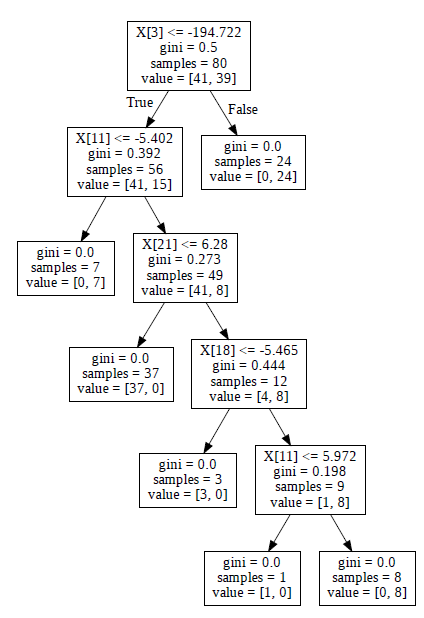


Figure 7. Decision Tree

Once all the models are created and their test data classifications are predicted, the accuracy and confusion matrix for each model is calculated and displayed. These results will be reviewed in the next section of the paper.

**Results and Discussion**

This project produces eight models to classify sounds from marine mammals and manmade ocean craft. Each model is tested on 20 sound clips, due to the random breakout 11 are manmade and nine are marine mammals. This was a relatively even spread which allows for an unbiased test. In addition to the accuracy of the models, the direction in which misclassification is predominate is evaluated. This is viewed as how conservative each model is. For example, it is safer for a submarine to classify a dolphin as a ship than it is to classify a ship as a dolphin. If a dolphin is classified as a ship the submarine will steer clear of it and not have any issues, but if a ship is classified as a dolphin the submarine may ignore it and cause a collision. Future would should focus on adjusting thresholds to cause the models to fail toward classifying sound as manmade so that collisions can be avoided.

The first two models were Support Vector Machines using a cubic polynomial kernel and a radial basis classifier kernel. Both SVM models had 75% accuracy on the test data. All misclassified sound clips were marine mammal noises that were classified as ships. This is conservative and makes for safe algorithms.

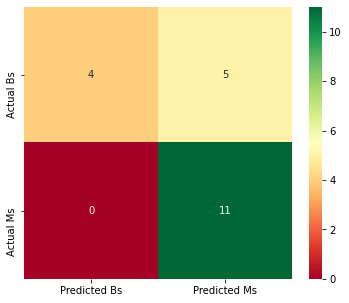
****

Figure 8. SVM Confusion Matrix

The logistics regression model has the highest accuracy of all models in this study with 95%. The misclassified sound clip is a ship that is classified as a marine mammal, which is not conservative. Based on the small training data set and minimal adjustments to the basic logistics regression model, this is likely the best starting point to implement AI/ML for UUV and submarine SONAR classification. Moving forward, the class weight and solver should be tested to find an optimal model. Additionally, this would be the ideal starting point to try classifying the types of marine mammals and distinguishing between surface ships and submarines as this would be the next challenge to implement these models into UUVs and submarines.

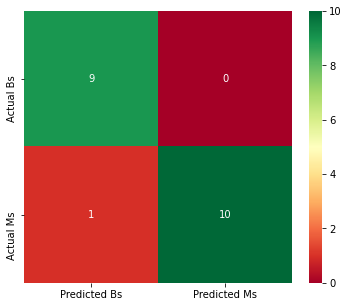


Figure 9. Logistics Regression Confusion Matrix

The keras model used in this work uses the sequential model and has three layers. This structure could be adjusted to optimize the algorithm, but is chosen for the simplicity due to using minimal enhancements to other algorithms in this work. This model used three layers. The first two layers used the relu activation and were 12 and eight units respectively. The last layer was a single unit and used the sigmoid activation. The model was compiled using the adam optimizer, binary cross-entropy loss, and accuracy as the metric. To fit the data, 150 epochs were used with a batch size of ten. In all, Keras produces an 85% accuracy, but all three misclassified test data points were manmade. This makes the algorithm non-conservative.

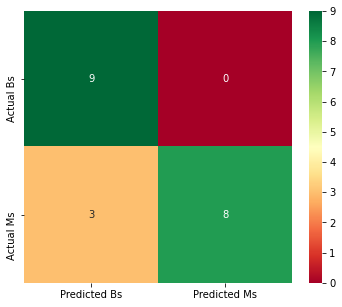


Figure 10. Keras Confusion Matrix

Two Naïve Bayes models are used in this work – multinomial and Gaussian. The Gaussian model outperformed the multinomial model with 85% accuracy over the 75% accuracy of the multinomial model. Additionally, the Gaussian model is as conservative as both SVM models, but it is more accurate. All three misclassified sounds were marine mammals that were classified as manmade. Because this is the most conservative model, the Gaussian Naïve Bayes model is the second best starting point to implement and increase the classes to discern between surface ships and submarines.

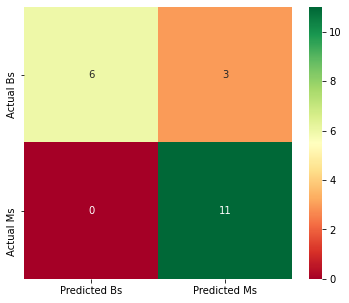
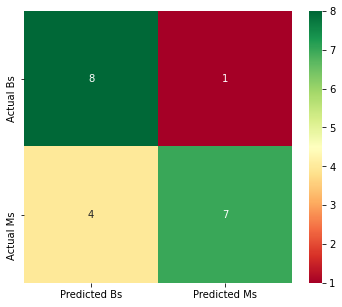


Figure 11. Naive Bayes Confusion Matrices - Multinomial (left) and Gaussian (right)

K-means was the only unsupervised machine learning algorithm used in this work. The method had the lowest accuracy at 50%. Three scaling methods were tested with the algorithm and each was also tested with PCA (Principal Component Analysis). The model performance using standardization alone was best with the Standard Scaler. When used with PCA, the Min Max Scaler was the optimal standardization function. Figure 12compares the different inputs to the K-means algorithm, showing that the Standard Scaler alone was the top performer as it correctly classified more ships and submarines.The K-means algorithm should be tested with larger datasets and more classes to see if it is able to compete with supervised learning when trying to distinguish between classes of ships and submarines.

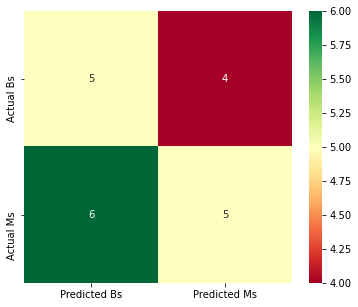
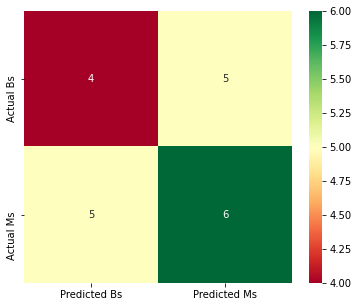


Figure 12. K-Means Confusion Matrices - Standard Scaler (left) and Min Max Scaler with PCA (right)

The K-Nearest Neighbors model is created using the 11 nearest data points based on the results having the best test score and best combined test and training score. This KNN model is 90% accurate on this test data set. The model misclassified one manmade and one biologic noise.

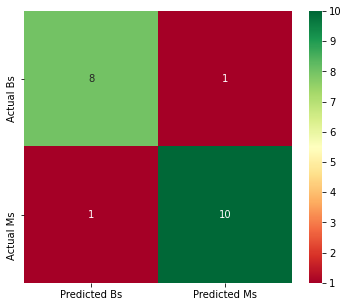


Figure 13. K-Nearest Neighbors Confusion Matrix

The decision tree model in this work has an accuracy of 80%, with three of the misclassified sound clips being manmade. This makes the decision tree a non-conservative model.

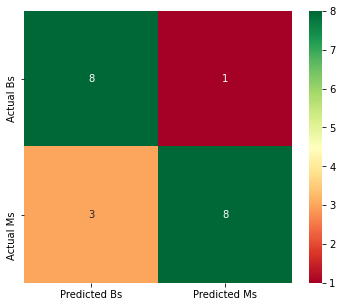


Figure 14. Decision Tree Confusion Matrix

**Conclusion**

In all, open source python AI/ML algorithms can quickly and easily be built to classify ocean noises. While the majority of the algorithms tested are supervised machine learning, which creates some bias, the results of the K-means algorithm shows that supervised learning outperformed by 25% compared to all other algorithms tests. Additionally, for this problem statement the amount of available, labeled data makes it such that a supervised machine learning algorithm can be implemented without additional complications of human classification prior to building the models.

The algorithms and the models created in this work are ranked based on the accuracy at identifying ships and submarines and overall accuracy. The rank weights the accuracy of identifying ships and submarines higher as it will prevent collisions. The overall accuracy is used as a tie breaker.

Of the supervised machine learning algorithms built in this work, the logistics regression model is the most accurate, but both SVM models and the Gaussian Naïve Bayes model correctly classified all of the ships and submarines. Therefore, the Gaussian Naïve Bayes and SVM models were ranked one and two respectively. Table 1 shows a complete rank and score breakout. The top four are the recommended starting point for subsequent work as they have the highest accuracy for determining manmade sounds in the ocean, making them the safest for UUVs and submarines to use. All models should be tested when evaluating the ability to classify types of ships and submarines in future work as the increased number of classes and data for training may change the results of the testing.

Table 1. Summary of Results



**Acknowledgements**

Thank you to maritime.org and Watkins Marine Mammal Sound Database for the free sound clips as they were critical to the success of this project.

**Appendix A**

from google.colab import drive

drive.mount('/content/drive')

*Load necessary libraries*

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import sklearn

import seaborn as sns

import librosa

import librosa.display

import wave, os, glob

import tensorflow as tf

*Load the data into two arrays.*

# This code cell loads the biologic data

# Create empty arrays to pull in the data. x will be the wave filea for each pull, y will be the size of the wave file, and bio will be a list of the wave files

x=[]

y=[]

sr=[]

bio=[]

# Use a for loop to load the floating point time series and amplitude in dB

for filename in glob.glob(os.path.join('/content/drive/My Drive/Colab Notebooks/Project/Biologic','\*wav')):

x,sr=librosa.load(filename,sr=48000) #This line creates the floating point time series from the wav file

y.append(x.shape[0]) #Save the length of the wave file

x=[x,librosa.amplitude\_to\_db(abs(librosa.stft(x)))] #Load the wave file data and also convert it to dB for use in spectrograms

bio=np.append(bio,x,axis=0)

bio=np.array(bio)

# This code cell loads the manmade data

# Create empty arrays to pull in the data. x will be the wave filea for each pull, y will be the size of the wave file, and bio will be a list of the wave files

x=[]

y=[]

sr=[]

man=[]

# Use a for loop to load the floating point time series and amplitude in dB

for filename in glob.glob(os.path.join('/content/drive/My Drive/Colab Notebooks/Project/Manmade','\*wav')):

x,sr=librosa.load(filename,sr=48000) #This line creates the floating point time series from the wav file

y.append(x.shape[0]) #Save the length of the wave file

x=[x,librosa.amplitude\_to\_db(abs(librosa.stft(x)))] #Load the wave file data and also convert it to dB for use in spectrograms

man=np.append(man,x,axis=0)

man=np.array(man)

# Check the shape of bio to make sure all wav files loaded (should be 50 files with two rows for each ==> 100 rows)

print(bio.shape)

*Plot Waveforms for biologic noise:*

*\*\*Note:\*\* The indices for the data for waveplots are even (0, 2, 4,...). Also, all the animals are split: 12 dolphins, 13 seals, 12 whales, and 13 walruses.*

# This code cell plots the waveforms of a Atlantic Spotted Dolphins (first 12 bio wav files)

for i in range(12):

plt.figure(figsize=(14, 5))

librosa.display.waveplot(bio[2\*i], sr=sr)

# This code cell plots the waveforms of Bearded Seals (13-25 wav files in bio)

for i in range(13):

plt.figure(figsize=(14, 5))

librosa.display.waveplot(bio[2\*i+24], sr=sr)

# This code cell plots the waveforms of Beluga Whales (26-37 wav files in bio)

for i in range(12):

plt.figure(figsize=(14, 5))

librosa.display.waveplot(bio[2\*i+50], sr=sr)

# This code cell plots the waveforms of Walruses (38-50 wav files in bio)

for i in range(13):

plt.figure(figsize=(14, 5))

librosa.display.waveplot(bio[2\*i+74], sr=sr)

*Plot Waveforms for manmade sounds:*

*\*\*Note:\*\* The indices for the data for waveplots are even (0, 2, 4,...). Also, all the first 25 are ships and the second 25 are submarines.*

# This code cell plots the waveforms of ships (1-25 in man)

for i in range(25):

plt.figure(figsize=(14, 5))

librosa.display.waveplot(man[2\*i], sr=sr)

# This code cell plots the waveforms of submarines (26-50 in man)

for i in range(25):

plt.figure(figsize=(14, 5))

librosa.display.waveplot(man[2\*i+50], sr=sr)

*Plot the Spectrograms:*

*Note, the data for this are the odd indices (1, 3, 5,...). Breakouts of the animals and ships/submarines remain the same.*

# Spectrograms for Atlantic Spotted Dolphins (1-12 in bio)

for i in range(12):

plt.figure(figsize=(14,5))

librosa.display.specshow(bio[i\*2+1], sr=sr, x\_axis='time', y\_axis='hz')

plt.colorbar()

# Spectrograms for Bearded Seals (13-25 in bio)

for i in range(12):

plt.figure(figsize=(14,5))

librosa.display.specshow(bio[i\*2+25], sr=sr, x\_axis='time', y\_axis='hz')

plt.colorbar()

# Spectrograms for Beluga Whales (26-37 in bio)

for i in range(12):

plt.figure(figsize=(14,5))

librosa.display.specshow(bio[i\*2+51], sr=sr, x\_axis='time', y\_axis='hz')

plt.colorbar()

# Spectrograms for Walruses (28-50 in bio)

for i in range(12):

plt.figure(figsize=(14,5))

librosa.display.specshow(bio[i\*2+75], sr=sr, x\_axis='time', y\_axis='hz')

plt.colorbar()

# Spectrograms for ships (1-25 in man)

for i in range(25):

plt.figure(figsize=(14,5))

librosa.display.specshow(man[i\*2+1], sr=sr, x\_axis='time', y\_axis='hz')

plt.colorbar()

# Spectrograms for submarines (26-50 in man)

for i in range(25):

plt.figure(figsize=(14,5))

librosa.display.specshow(man[i\*2+51], sr=sr, x\_axis='time', y\_axis='hz')

plt.colorbar()

*Feature extraction:*

# Create the header that will be used when writing the csv files. Filenames took 4 columns so it was put in 4 times to align the data portions properly

header = 'filename filename2 filename3 filename4 Class spectral\_centroid spectral\_rolloff Chromagram'

for i in range(1, 21):

header += f' mfcc{i}'

header = header.split()

# Create csv of biologic noise data

# Inport csv to write/read/open csvs

import csv

# Create an empty array for the features. This will be a copy of the data we save to csv

biofeatures=[]

# Set the hop length and the variable B. B will denote that the data is a biologic noise.

hop\_length=512

B='B'

# Create a csv and write the header to it

file = open('data.csv','w',newline='')

with file:

writer=csv.writer(file)

writer.writerow(header)

# Use a for loop to create make a row vector for each sound clip with: file name, class, spectral centroid, spectral rolloff, and chromagram.

for i in range(50):

spectral\_centroids = librosa.feature.spectral\_centroid(bio[2\*i], sr=sr) #Use librosa to calculate spectral centroid

spectral\_rolloff = librosa.feature.spectral\_rolloff(bio[2\*i]+0.01, sr=sr) #Use librosa to calculate spectral rolloff

mfccs = librosa.feature.mfcc(bio[2\*i], sr=sr) #Use librosa to calculate the mfccs

chromagram = librosa.feature.chroma\_stft(bio[2\*i], sr=sr, hop\_length=hop\_length) #Use librosa to calculate the chromagram

# F=['M',spectral\_centroids,spectral\_rolloff,mfccs,chromagram] #This was a previous iteration, but did not work well

F2=f'{filename} {B} {np.mean(spectral\_centroids)} {np.mean(spectral\_rolloff)} {np.mean(chromagram)}' #Create a row vector with all the specified values

# This for loop appends the variable F2 (the array of values for the sound clip) with the mfccs.

for e in mfccs:

F2 += f' {np.mean(e)}'

file = open('data.csv', 'a', newline='')

# This while loop writes the row to the csv file

with file:

writer = csv.writer(file)

writer.writerow(F2.split()) #Using .split() breaks the list into separate items to write into the csv

biofeatures.append(F2) #Save the values to the end of biofeatures

biofeatures=np.array(biofeatures) #Make sure biofeatures is a numpy array for future work

# Create csv of manmade noise data

# Create an empty array for the features. This will be a copy of the data we save to csv

manfeatures=[]

# Set the variable M. M will denote that the data is manmade noise.

M='M'

# Use a for loop to create make a row vector for each sound clip with: file name, class, spectral centroid, spectral rolloff, and chromagram. Write to the same file as the biologic data

for i in range(50):

spectral\_centroids = librosa.feature.spectral\_centroid(man[2\*i], sr=sr) #Use librosa to calculate spectral centroid

spectral\_rolloff = librosa.feature.spectral\_rolloff(man[2\*i]+0.01, sr=sr) #Use librosa to calculate spectral rolloff

mfccs = librosa.feature.mfcc(man[2\*i], sr=sr) #Use librosa to calculate the mfccs

chromagram = librosa.feature.chroma\_stft(man[2\*i], sr=sr, hop\_length=hop\_length) #Use librosa to calculate the chromagram

# F=['M',spectral\_centroids,spectral\_rolloff,mfccs,chromagram] #This was a previous iteration, but did not work well

F2=f'{filename} {M} {np.mean(spectral\_centroids)} {np.mean(spectral\_rolloff)} {np.mean(chromagram)}'#Create a row vector with all the specified values

# This for loop appends the variable F2 (the array of values for the sound clip) with the mfccs.

for e in mfccs:

F2 += f' {np.mean(e)}'

file = open('data.csv', 'a', newline='')

# This while loop writes the row to the csv file

with file:

writer = csv.writer(file)

writer.writerow(F2.split()) #Using .split() breaks the list into separate items to write into the csv

manfeatures.append(F2) #Save the values to the end of manfeatures

manfeatures=np.array(manfeatures) #Make sure manfeatures is a numpy array for future work

*Clean the data - remove filenames*

# Read the data back in from the new csv

data=pd.read\_csv('/content/data.csv',header=0)

#data.head()

# Drop the columns with the file names

data=data.drop('filename', axis=1)

data=data.drop('filename2',axis=1)

data=data.drop('filename3',axis=1)

data=data.drop('filename4',axis=1)

# Separate the inputs and outputs (with filenames gone, the output is the first column with index of [:,0])

X=data.iloc[:,1:-1]

Y=data.iloc[:,0]

# Scale X

from sklearn import preprocessing

# Using scale

Xscaled=preprocessing.scale(X)

# Using StandardScaler

min\_max\_scaler=preprocessing.StandardScaler()

Xscaled2=min\_max\_scaler.fit\_transform(X)

Xscaled2=pd.DataFrame(Xscaled2)

# Using MinMaxScaler

min\_max\_scaler=preprocessing.MinMaxScaler()

Xscaled3=pd.DataFrame(min\_max\_scaler.fit\_transform(X))

print(data.head())

*Split the arrays 80/20 and then combine the training and test portions. Use random state 15 for reproducibility*

# Split data

from sklearn.model\_selection import train\_test\_split

Xtrain,Xtest,Ytrain,Ytest=train\_test\_split(X,Y,test\_size=0.2,random\_state=15)

# Create binary Y data

Y2=[]

for i in range(len(Y)):

if Y[i]=='M':

Y2.append(1)

else:

Y2.append(0)

# Split scaled data - Chose one at random to designate Y2 since they will be split in the same order due to designating random\_state=15

Xtrainscaled,Xtestscaled,Ytrain2,Ytest2=train\_test\_split(Xscaled,Y2,test\_size=0.2,random\_state=15)

Xtrainscaled2,Xtestscaled2,Ytrain,Ytest=train\_test\_split(Xscaled2,Y,test\_size=0.2,random\_state=15)

Xtrainscaled3,Xtestscaled3,Ytrain,Ytest=train\_test\_split(Xscaled3,Y,test\_size=0.2,random\_state=15)

*SVD*

#Support vector machines - Polynomial Kernel

#evaluate model accuracy.

#build a confusion matrix to better understand the result. Visualize the confusion matrix as a heat map using the Seaborn library.

from sklearn import svm

model = svm.SVC(kernel='poly') #define an instance of SVC class using default cubic degree

model.fit(Xtrain,Ytrain) #fit the model on train data and label

#Find results of test data

resultsPoly=model.predict(Xtest)

#accuracy

from sklearn.metrics import accuracy\_score

scorePoly=accuracy\_score(Ytest,resultsPoly)

print(scorePoly)

"""

#Plot confusion matrix

from sklearn.metrics import confusion\_matrix

mat = confusion\_matrix(Ytest, resultsPoly)

sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False,

)

plt.ylabel('True Class')

plt.xlabel('Predicted Class')

"""

# Confusion matrix using heatmap

from sklearn.metrics import confusion\_matrix

mat = confusion\_matrix(Ytest, resultsPoly)

plt.figure(figsize=(6,5)) # set the size of the figure

p=sns.heatmap(mat, annot=True,cmap ='RdYlGn',xticklabels=['Predicted Bs','Predicted Ms'],yticklabels=['Actual Bs','Actual Ms']) # we use the seaborn library to plot the heatmap.

plt.ylabel('True Class')

plt.xlabel('Predicted Class')

#Support vector machines - RBF Kernel

#evaluate model accuracy.

#build a confusion matrix to better understand the result. Visualize the confusion matrix as a heat map using the Seaborn library.

model = svm.SVC(kernel='rbf') #define an instance of SVC class using rbf

model.fit(Xtrain,Ytrain) #fit the model on train data and label

#Find results of test data

resultsRBF=model.predict(Xtest)

#accuracy

from sklearn.metrics import accuracy\_score

scoreRBF=accuracy\_score(Ytest,resultsRBF)

print(scoreRBF)

"""

#Plot confusion matrix

from sklearn.metrics import confusion\_matrix

sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False,

)

plt.ylabel('True Class')

plt.xlabel('Predicted Class')

"""

# Confusion matrix using heatmap

mat = confusion\_matrix(Ytest, resultsRBF)

plt.figure(figsize=(6,5)) # set the size of the figure

p=sns.heatmap(mat, annot=True,cmap ='RdYlGn',xticklabels=['Predicted Bs','Predicted Ms'],yticklabels=['Actual Bs','Actual Ms']) # we use the seaborn library to plot the heatmap.

*Logistics Regression Model*

from sklearn.linear\_model import LogisticRegression

#define an instance of the LogisticRegression class

model = LogisticRegression(penalty="l2",C=1.0,fit\_intercept=True,max\_iter=10000)

# fit the model on the training set

model.fit(Xtrain, Ytrain)

# predict results of the Test data

results = model.predict(Xtest) # Prediction using the test data

#print out the model parameters beta\_i

print(model.intercept\_)

print(model.coef\_)

# evaluate model accuracy.

score = accuracy\_score(Ytest, results)

print(score)

#build a confusion matrix to better understand the result. Visualize the confusion matrix as a heat map using the Seaborn library.

from sklearn.metrics import confusion\_matrix

conf\_mat = confusion\_matrix(Ytest, results) #use the confusion\_matrix() function with arguments-true output and predicted output

conf\_mat

# Confusion matrix using heatmap

plt.figure(figsize=(6,5)) # set the size of the figure

p=sns.heatmap(conf\_mat, annot=True,cmap ='RdYlGn',xticklabels=['Predicted Bs','Predicted Ms'],yticklabels=['Actual Bs','Actual Ms']) # we use the seaborn library to plot the heatmap.

*Keras*

print(Xtrain.head())

print(range(len(Ytrain)))

print(Ytrain)

from keras import models

from keras import layers

model=models.Sequential()

model.add(layers.Dense(12, activation='relu', input\_shape=(Xtrain.shape[1],)))

model.add(layers.Dense(8, activation='relu'))

model.add(layers.Dense(1, activation='sigmoid'))

model.compile(optimizer='adam',

loss='binary\_crossentropy',

metrics=['accuracy'])

history = model.fit(np.array(Xtrain),np.array(Ytrain2),epochs=150,batch\_size=10,verbose=0)

Ykeras=model.predict\_classes(np.array(Xtest))

# Confusion matrix using heatmap

conf\_mat = confusion\_matrix(Ytest2, Ykeras)

plt.figure(figsize=(6,5)) # set the size of the figure

p=sns.heatmap(conf\_mat, annot=True,cmap ='RdYlGn',xticklabels=['Predicted Bs','Predicted Ms'],yticklabels=['Actual Bs','Actual Ms']) # we use the seaborn library to plot the heatmap.

*Build a Naive Bayes classifier*

#import MultinomialNB from sklearn library.

from sklearn.naive\_bayes import MultinomialNB

#create an instance of MultinomialNB class

model=MultinomialNB()

#fit the training data

model.fit(Xtrainscaled3, Ytrain) #Use Xtrainscaled3 because it doesn't have negative values (min\_max\_scaler)

#get the predicted labels for test data

YNB = model.predict(Xtestscaled)

#Print the accuracy of the classifier on test data.

#predict the accuracy of the model on test data

from sklearn.metrics import accuracy\_score

score = accuracy\_score(Ytest, YNB)

print(score)

#Print the confusion matrix as a heat map using Seaborn library.

#build a confusion matrix to visualize the performance of the model using seaborn

from sklearn.metrics import confusion\_matrix

conf\_mat = confusion\_matrix(Ytest, YNB)

# Confusion matrix using heatmap

plt.figure(figsize=(6,5)) # set the size of the figure

p=sns.heatmap(conf\_mat, annot=True,cmap ='RdYlGn',xticklabels=['Predicted Bs','Predicted Ms'],yticklabels=['Actual Bs','Actual Ms']) # we use the seaborn library to plot the heatmap.

*LDA/GNB*

from sklearn.decomposition import LatentDirichletAllocation as LDA

lda=LDA(n\_components=2)

Xtrain2=lda.fit\_transform(Xtrainscaled3,Ytrain)

Xtest2=lda.transform(Xtestscaled3)

#visualize the data. We will plot the scatter plot for those two LDA components.

plt.figure(figsize=(5,5))

plt.xlabel('LD1')

plt.ylabel('LD2')

plt.scatter(Xtest2[:,0],Xtest2[:,1],c=Ytest)

#plt.legend(loc="upper left")

plt.legend()

plt.show()

from sklearn.naive\_bayes import GaussianNB

import time

classifier=GaussianNB()

t=time.time()

classifier.fit(Xtrain,Ytrain)

YGNB=classifier.predict(Xtest)

# evaluate model accuracy.

score = accuracy\_score(Ytest, YGNB)

print(score)

#build a confusion matrix to better understand the result. Visualize the confusion matrix as a heat map using the Seaborn library.

conf\_mat = confusion\_matrix(Ytest, YGNB) #use the confusion\_matrix() function with arguments-true output and predicted output

conf\_mat

#Heat map

plt.figure(figsize=(6,5)) # set the size of the figure

p=sns.heatmap(conf\_mat, annot=True,cmap ='RdYlGn',xticklabels=['Predicted Bs','Predicted Ms'],yticklabels=['Actual Bs','Actual Ms']) # we use the seaborn library to plot the heatmap.

*PCA*

from sklearn.decomposition import PCA

pca=PCA(2)

train\_data=pca.fit\_transform(Xtrainscaled3)

YPCA=pca.transform(Xtestscaled3)

#visualize the data. We will plot the scatter plot for those two LDA components.

plt.figure(figsize=(5,5))

plt.xlabel('LD1')

plt.ylabel('LD2')

plt.scatter(YPCA[:,0],YPCA[:,1],c=Ytest)

#plt.legend(loc="upper left")

plt.show()

*Clustering (Unsupervised)*

from sklearn.cluster import KMeans

kmeans=KMeans(n\_clusters=2,random\_state=0).fit(pca.transform(Xtrainscaled3)) #Chose Xtrainscaled2 because it had the highest accuracy (StandardScaler) - with PCA all three scaling methods had 50% accuracy

YKM=kmeans.predict(pca.transform(Xtestscaled3))

print(YKM)

# evaluate model accuracy.

score = accuracy\_score(Ytest2, YKM)

print(score)

#build a confusion matrix to better understand the result. Visualize the confusion matrix as a heat map using the Seaborn library.

conf\_mat = confusion\_matrix(Ytest2, YKM) #use the confusion\_matrix() function with arguments-true output and predicted output

conf\_mat

#Heat map

plt.figure(figsize=(6,5)) # set the size of the figure

p=sns.heatmap(conf\_mat, annot=True,cmap ='RdYlGn',xticklabels=['Predicted Bs','Predicted Ms'],yticklabels=['Actual Bs','Actual Ms']) # we use the seaborn library to plot the heatmap.

*KNN*

from sklearn.neighbors import KNeighborsClassifier

test\_scores=[]

train\_scores=[]

for i in range(1,15):

knn=KNeighborsClassifier(i)

knn.fit(Xtrain,Ytrain)

YKNNtrain=knn.predict(Xtrain)

train\_accuracy=accuracy\_score(YKNNtrain,Ytrain)

train\_scores.append(train\_accuracy)

YKNNtest=knn.predict(Xtest)

test\_accuracy=accuracy\_score(YKNNtest,Ytest)

test\_scores.append(test\_accuracy)

plt.figure(figsize=(12,5))

p = sns.lineplot(range(1,15),train\_scores,marker='\*',label='Train Score')

p = sns.lineplot(range(1,15),test\_scores,marker='o',label='Test Score')

plt.xlabel("K")

plt.ylabel("Accuracy")

#Setup a knn classifier with k neighbors

knn = KNeighborsClassifier(11)

knn.fit(Xtrain,Ytrain)

knn.score(Xtest,Ytest)

YKNN=knn.predict(Xtest)

#build a confusion matrix to better understand the result. Visualize the confusion matrix as a heat map using the Seaborn library.

conf\_mat = confusion\_matrix(Ytest, YKNN) #use the confusion\_matrix() function with arguments-true output and predicted output

conf\_mat

#Heat map

plt.figure(figsize=(6,5)) # set the size of the figure

p=sns.heatmap(conf\_mat, annot=True,cmap ='RdYlGn',xticklabels=['Predicted Bs','Predicted Ms'],yticklabels=['Actual Bs','Actual Ms']) # we use the seaborn library to plot the heatmap.

*Decision Tree*

from sklearn.tree import DecisionTreeClassifier

from sklearn.model\_selection import cross\_val\_score

clf=DecisionTreeClassifier(random\_state=0)

clf=clf.fit(Xtrain,Ytrain)

YDT=clf.predict(Xtest)

cross\_val\_score(clf, Xtrain, Ytrain, cv=10)

import graphviz

dot\_data = sklearn.tree.export\_graphviz(clf, out\_file=None)

graph = graphviz.Source(dot\_data)

graph.render("Tree")

# evaluate model accuracy.

score = accuracy\_score(Ytest, YDT)

print(score)

#build a confusion matrix to better understand the result. Visualize the confusion matrix as a heat map using the Seaborn library.

conf\_mat = confusion\_matrix(Ytest, YDT) #use the confusion\_matrix() function with arguments-true output and predicted output

conf\_mat

#Heat map

plt.figure(figsize=(6,5)) # set the size of the figure

p=sns.heatmap(conf\_mat, annot=True,cmap ='RdYlGn',xticklabels=['Predicted Bs','Predicted Ms'],yticklabels=['Actual Bs','Actual Ms']) # we use the seaborn library to plot the heatmap.

**References**

[1]        F. Heerde, I. Vatolkin, and G. Rudolph, “Comparing Fuzzy Rule Based Approaches for Music Genre Classification,” *Artificial Intelligence in Music, Sound, Art and Design*, vol. 12103, no. 9th, pp. 35–48, 2020, doi: 10.1007/978-3-030-43859-3.

[2]        R. Palaniappan, K. Sundaraj, and S. Sundaraj, “Artificial intelligence techniques used in respiratory sound analysis - A systematic review,” *Biomedizinische Technik*, vol. 59, no. 1. pp. 7–18, Feb. 2014, doi: 10.1515/bmt-2013-0074.

[3]        N. Singh, “Classification of Animal Sound Using Convolutional Neural Classification of Animal Sound Using Convolutional Neural Network Network,” doi: 10.21427/7pb8-9409.

[4]        D. Kucharski, M. Kajor, D. Grochala, M. Iwaniec, and J. Iwaniec, “Combining Spectral Analysis with Artificial Intelligence in Heart Sound Study,” *Advances in Science and Technology Research Journal*, vol. 13, no. 2, pp. 112–118, Jun. 2019, doi: 10.12913/22998624/108447.

[5]        D. Bardou, K. Zhang, and S. M. Ahmad, “Lung sounds classification using convolutional neural networks,” *Artificial Intelligence in Medicine*, vol. 88, pp. 58–69, Jun. 2018, doi: 10.1016/j.artmed.2018.04.008.

[6]        H. Coskun and T. Yigit, “Artificial intelligence applications on classification of heart sounds,” in *Nature-Inspired Intelligent Techniques for Solving Biomedical Engineering Problems*, IGI Global, 2018, pp. 184–202.

[7]        San Francisco Maritime National Park Association, “Historic Naval Sound and Video.” https://maritime.org/sound/#sealion (accessed Nov. 17, 2020).

[8]        Woods Hole Oceanographic Institution, “Watkins Marine Mammal Sound Database.” https://cis.whoi.edu/science/B/whalesounds/index.cfm (accessed Nov. 17, 2020).

[9]        F. Pedregosa and et al., “Scikit-learn: Machine Learning in Python,” *JMLR*, vol. 12, no. 85, pp. 2825–2830, 2011.

[10]       P. Pandy, “Music Genre Classification with Python,” *Towards Data Science*, Dec. 13, 2018. https://towardsdatascience.com/music-genre-classification-with-python-c714d032f0d8 (accessed Nov. 17, 2020).

[11]       B. McFeee and et al., “Librosa.” Librosa, 2020.

[12]       “Wikipedia: Chroma Feaure.” https://en.wikipedia.org/wiki/Chroma\_feature (accessed Nov. 17, 2020).

[13]       “Wikipedia: Mel-frequency Cepstrum.” https://en.wikipedia.org/wiki/Mel-frequency\_cepstrum (accessed Nov. 17, 2020).

[14]       “Wikipedia: Keras.” https://en.wikipedia.org/wiki/Keras (accessed Nov. 30, 2020).