Music Classification With AWS Machine Learning

CIS 4130 – Section CMWA – Fall 2022 –Professor Richard Holowczak

By Brian Rodriguez

(brian.rodriguez3@baruchmail.cuny.edu)

Description of the project being created:

The dataset that I have chosen is titled “Music Classification WAV” where I have downloaded the dataset from Kaggle and is 41.6 GB. This dataset contains sound files (.wav) that are recorded full songs and cropped 16 seconds of a song. There are sound files contained within the dataset that can be used to train a machine-learning model to help recognize different genres of music. Within the included CSV file, I am able to see how many songs of each genre there are and for my project, I hope to build a model that will be able to distinguish the music genre of the sample sound files.

Included in the dataset is a CSV file that contains the genre of each sound file in the training folder, this may be used to help train the model to understand the different genres so that after the model is trained I will be able to use the test sound files to see if the model is properly functioning. There are 18 different genres each having a genre_id, I am hoping that my model will be able to identify which genre_id correlates with the sound file.

This project was constructed using Python and the Keras library to read the sound files through code and I hope to become familiar with the code and how to implement it within my project. My goal for this project is to have a model that will be able to read sound files (.wav) and identify what genre of music is the sample file, by first training the model to learn what each genre will sound like by using the csv file containing which genre of music each sample file is, my model will then be able to identify future samples outside of the training data.

https://www.kaggle.com/datasets/asisheriberto/music-classification-wav

Connecting To Data Source:

My data is coming directly from the Kaggle website and is being uploaded directly to a bucket within my EC2 instance in AWS. I connected to kaggle through an API generated through the kaggle website and then followed the code below to create the bucket that is used to store the kaggle dataset.

The complete code used to download and load the dataset in this milestone can be found in Appendix A.

Downloading dataset to s3 bucket

> kaggle datasets download --quiet asisheriberto/music-classification-wav -p - |aws s3 cp - s3://music-data-br/music.zip

Checking if the bucket was created with stored dataset

> aws s3 ls s3://music-data-br

```
[ec2-user@ip-172-31-0-238 ~]$ aws s3 ls s3://music-data-br
2022-10-18 17:43:11 44682980240 music.zip
```

Testing Boto3 module in Python

```
>>> import boto3
>>> s3 = boto3.resource('s3')
>>> for bucket in s3.buckets.all():
...     print(bucket.name)
...
music-data-br
```

Unzipping File

Due to the size of the zip file, it could not be unzipped into my s3 bucket as the program within python3 would be killed due to the limitations of the RAM of my current EC2 instance. I instead increased my instance storage and unzipped the file in my EC2 instance for the moment and will move all files into an s3 bucket in the future. All unzipped files in my EC2 instance will be accessible through python3 to produce descriptive statistics about the dataset.

$ unzip music-classification-wav.zip

```
[ec2-user@ip-172-31-0-238 ~]$ ll
total 43635740
-rw-rw-r-- 1 ec2-user ec2-user         114 Oct 19 15:36 connections.py
-rw-rw-r-- 1 ec2-user ec2-user         114 Oct 19 15:26 connnections.py
drwxrwxr-x 3 ec2-user ec2-user          23 Oct 18 21:11 data_test
drwxrwxr-x 3 ec2-user ec2-user          57 Oct 18 21:04 IA
-rw-rw-r-- 1 ec2-user ec2-user 44682980240 Oct 18 16:39 music-classification-wav.zip
-rw-rw-r-- 1 ec2-user ec2-user         523 Oct 20 22:17 opener.py
drwxrwxr-x 2 ec2-user ec2-user          40 Oct 19 15:36 __pycache__
drwxrwxr-x 3 ec2-user ec2-user          24 Oct 18 21:04 TRAIN_V2
```

After all, the files have been unzipped in the EC2 instance, I used the copy command to move them into my S3 bucket so that I would now have the unzipped file contents accessible from my S3 bucket.

Descriptive Statistics:

My dataset consists of 3 different folders, one being ‘data-test’, a folder containing 5076 .wav files to be used in my machine learning algorithm to test if the algorithm is properly working. The next folder, ‘IA’, contains 17 folders that hold .wav files of different genres, each folder holds a different number of .wav files with folder 17 (‘Blues’) having the least sound files, 58 .wav files, and folder 0 (‘Electronic’) having the most sound files, 3071 .wav files. Within the IA folder are also 2 CSV files, genres.csv which identify the genre of each folder, and train.csv, containing an organized list of all .wav files and their respectable genres which will be used in the training algorithm. The third folder in my dataset is ‘TRAIN_V2’ which is a copy of the training files that can be found in the previous folder, ‘IA’. As I move forward with the project, I believe it will be a challenge to process the 17 different genres as I might have to focus on only a few genres. Seeing how each folder holds a different number of sound files, I will need to avoid using folders that have too few files as it might not work for training the model properly.

All code used in this Milestone to extract statistics on the data can be found in Appendix B.

```
>>> import pandas as pd
```

Number of files in data_out (Training Files) and in data_test (Test Files)

Training Files

./IA/data_out/0 : 3071
./IA/data_out/1 : 3095

./IA/data_out/10 : 796
./IA/data_out/11 : 495
./IA/data_out/12 : 408
./IA/data_out/13 : 306
./IA/data_out/14 : 142
./IA/data_out/15 : 94
./IA/data_out/16 : 94
./IA/data_out/17 : 58
./IA/data_out/2 : 2582
./IA/data_out/3 : 1800
./IA/data_out/4 : 1757
./IA/data_out/5 : 1214
./IA/data_out/6 : 1181
./IA/data_out/7 : 1044
./IA/data_out/8 : 945
./IA/data_out/9 : 814

Test Files

>>> count_files(dir_path)
./data_test/data_test : 5076

List of music genres and count of genres
>>> print(genre_list)
['Instrumental', 'Punk', 'Folk', 'Old-Time / Historic', 'Rock', 'International', 'Chiptune / Glitch', 'Experimental', 'Pop', 'Electronic', 'Hip-Hop', 'Jazz', 'Classical', 'Ambient Electronic', 'Blues', 'Easy Listening', 'Soul-RnB', 'Country', 'Spoken']
>>> len(genre_list)
19

Genres and their respective folder numbers

{'genre': 'genre_id', 'Electronic': '0', 'Rock': '1', 'Punk': '2', 'Experimental': '3', 'Hip-Hop': '4', 'Folk': '5', 'Chiptune / Glitch': '6', 'Instrumental': '7', 'Pop': '8', 'International': '9', 'Ambient Electronic': '10', 'Classical': '11', 'Old-Time / Historic': '12', 'Jazz': '13', 'Country': '14', 'Soul-RnB': '15', 'Spoken': '16', 'Blues': '17', 'Easy Listening': '18'}

Count of .wav files in each genre

```
>>> print(genre_count)
{'Instrumental': 1045, 'Punk': 2584, 'Folk': 1215, 'Old-Time / Historic': 408, 'Rock': 3097,
'International': 814, 'Chiptune / Glitch': 1181, 'Experimental': 1801, 'Pop': 945,
'Electronic': 3073, 'Hip-Hop': 1761, 'Jazz': 306, 'Classical': 495, 'Ambient Electronic':
796, 'Blues': 58, 'Easy Listening': 13, 'Soul-RnB': 94, 'Country': 142, 'Spoken': 94}
```

Coding And Modeling:

For the next milestone of my project, I will be building a pipeline that uses RandomForestClassifiers to create predictions of what genre of music the '.wav' file will be. I begin by creating a function that will extract 2 features from my dataset, one being mfcc and the other being the mean of the array of those original mfcc features. Mfcc (Mel-frequency cepstral coefficients) helps us take snippets of music signals from an audio file and compare those signal waves to that of other audio files. The librosa package also helps us read the '.wav' files into Python3, as mfcc grabs the frequency points several times over the audio file we are left with an array of many data points from the features. This gave me trouble as I was unsure of how I would be able to use this array within spark but was able to transform it into a Spark vector which was then usable in Spark. In my code, there is also a portion that reads the csv file that is within the dataset containing a list of all files and their genres. This code connects each .wav file with the filename in the csv to determine the genre of that '.wav' file. Before using the spark data frame we need to drop the 'genre' column as it will give errors for being a string and not a numeric type of data. In the end, I build the pipeline using RandomForestClassifier, this classification method takes a collection of trees (each tree trained on a random subset of data) and compares them to make predictions. I first tried using LogisticRegression however because my data contains many different target variables (17 genres) this method was much more effective and although it was not very accurate it had 3 out of 20 perfect agreements. For the code below, I relied on 'Implementing Audio Classification Project Using Deep Learning (analyticsvidhya.com)' and 'DataTechNotes: MLlib Random Forest Classification Example with PySpark' as a resource for the following code.

The complete code used to build the following pipeline can be found in Appendix C.

```
#Read in the data as a pandas dataframe

metadata = pd.read_csv('C:/Users/Brianrod/Downloads/music1/IA/train.csv')

#metadata.head(10)
```

```
#Defining a function that will extract features from each wave file of each folder.
```

```
file_name='C:/Users/Brianrod/Downloads/music1/IA/data_out'
#file_name='s3://music-data-br/data_out'

def features_extractor(file):
    #load the file (audio)
    #We are able to load and read these files using the Librosa package
    audio, sample_rate = librosa.load(file_name, res_type='kaiser_fast')
    #mfcc is one of the 2 features we extarct as we slow down the rate of the audio to capture
#more information from each frame of the audio file
    mfccs_features = librosa.feature.mfcc(y=audio, sr=sample_rate, n_mfcc=40)
    #for our second set of features we take the mean of the array using the previous feature.
    mfccs_scaled_features = np.mean(mfccs_features.T,axis=0)
    return mfccs_scaled_features
```

```
### converting extracted_features to Pandas data frame
extracted_features_df=pd.DataFrame(extracted_features,columns=['feature','genre'])
extracted_features_df
```

In [21]:
```
### converting extracted_features to Pandas dataframe
extracted_features_df=pd.DataFrame(extracted_features,columns=['feature','genre'])
extracted_features_df
```

Out[21]:

	feature	genre
0	[-220.76562, 90.60733, 49.78978, 26.039112, 5....	Instrumental
1	[18.86783, 80.36447, 0.5729405, 23.20239, -0.7...	Punk
2	[-290.6577, 83.498055, -42.749413, 21.640993, ...	Folk
3	[-262.08475, 144.38524, -108.97106, -26.358757...	Old-Time / Historic
4	[15.768597, 72.60664, 4.0164847, 32.805832, 2....	Rock
...	...	...
19891	[-64.83869, 46.36941, 8.45531, 45.98196, -0.35...	Electronic
19892	[-170.57887, 106.53604, 42.701595, 28.565674, ...	Hip-Hop
19893	[-12.067158, 46.283924, 0.009649797, 29.335272...	Hip-Hop
19894	[-129.28217, 82.20686, 18.308973, 53.467686, 1...	Punk
19895	[-289.23605, 147.76265, 30.262993, 26.133278, ...	Rock

19896 rows × 2 columns

```
#Now we will build the pipeline to generate predictions of what the genre of music it will be.
#For this pipeline I have used RandomForestClassifier which works better with many target
#variable, in this case 'genres'
forestclassifier = RandomForestClassifier(
    featuresCol='vector',
```

```
    labelCol='genre_label'
)

forest_pipeline = Pipeline(
    stages= [assembler, forestclassifier]
)
```

```
# fit the pipeline for the trained data
model = forest_pipeline.fit(genre_labelled_df)
# transform the data
sample_data_train = model.transform(genre_labelled_df)

# view some of the columns generated
sample_data_train.select('vector', 'genre_label', 'rawPrediction', 'probability', 'prediction').show()
```

```
+--------------------+-----------+--------------------+--------------------+----------+
|              vector|genre_label|       rawPrediction|         probability|prediction|
+--------------------+-----------+--------------------+--------------------+----------+
|[-220.765625,90.6...|          3|[0.0,0.8219636575...|[0.0,0.0410981828...|      15.0|
|[-308.45782470703...|          3|[0.0,0.7811105867...|[0.0,0.0390555293...|      11.0|
|[-323.08123779296...|          3|[0.0,0.5351607894...|[0.0,0.0267580394...|       3.0|
|[-251.44256591796...|          3|[0.0,0.9394591462...|[0.0,0.0469729573...|      11.0|
|[-383.65261840820...|          3|[0.0,0.6563661210...|[0.0,0.0328183060...|      15.0|
|[-285.96405029296...|          3|[0.0,0.5798971386...|[0.0,0.0289948569...|      15.0|
|[-143.62411499023...|          3|[0.0,0.7904316344...|[0.0,0.0395215817...|      15.0|
|[-85.137138366699...|          3|[0.0,0.8070518119...|[0.0,0.0403525905...|       4.0|
|[-196.28053283691...|          3|[0.0,0.6371877569...|[0.0,0.0318593878...|      15.0|
|[-180.95129394531...|          3|[0.0,1.0049414621...|[0.0,0.0502470731...|       7.0|
|[-537.60827636718...|          3|[0.0,0.5673089948...|[0.0,0.0283654497...|       3.0|
|[-129.29640197753...|          3|[0.0,0.7544424120...|[0.0,0.0377221206...|      11.0|
|[-167.74966430664...|          3|[0.0,0.9588656328...|[0.0,0.0479432816...|      15.0|
|[-256.83239746093...|          3|[0.0,1.1684175150...|[0.0,0.0584208757...|      11.0|
|[-138.42581176757...|          3|[0.0,1.3848731400...|[0.0,0.0692436570...|       4.0|
|[-176.87876892089...|          3|[0.0,0.7277292965...|[0.0,0.0363864648...|       7.0|
|[-326.57540893554...|          3|[0.0,1.2718760393...|[0.0,0.0635938019...|       7.0|
|[-347.40597534179...|          3|[0.0,0.9088282901...|[0.0,0.0454414145...|      15.0|
|[-64.209693908691...|          3|[0.0,0.7962447940...|[0.0,0.0398122397...|      15.0|
|[-282.24951171875...|          3|[0.0,0.7579142697...|[0.0,0.0378957134...|       3.0|
+--------------------+-----------+--------------------+--------------------+----------+
only showing top 20 rows
```

```
#Determining the accuracy of the pipeline
```

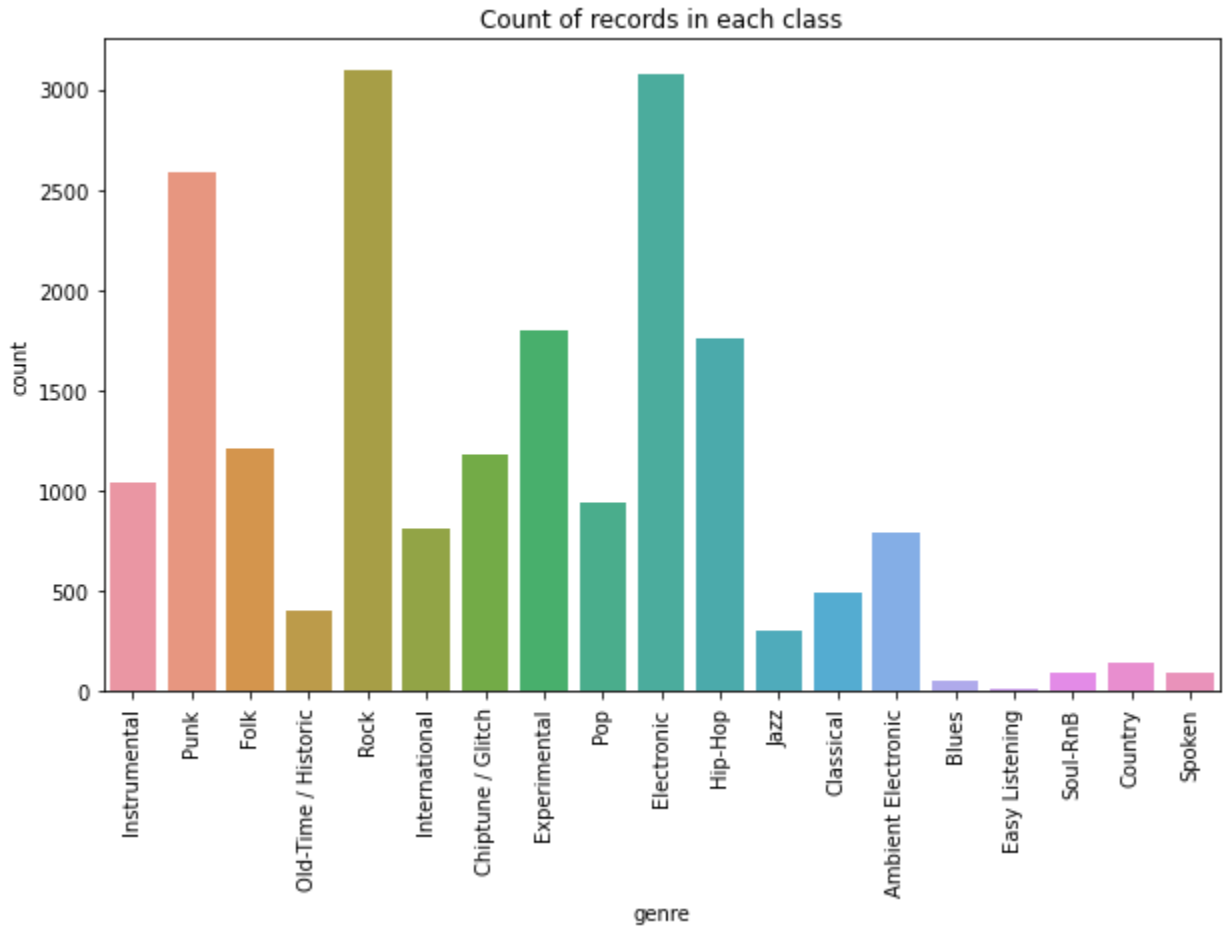
Prediction Accuracy: 0.25088877160396833

Visualizing Results:

All code used to create the visualization below can be found in Appendix D.

Visual 1:

This is a bar graph created in Matplot lib and was created to give users an idea of how many files are in each genre of the dataset. From this bar graph, we see that genres such as Punk, Rock, and Electronic dominate the dataset and should therefore expect to see more of those genres when making predictions through the pipeline. This also shows us that genres such as Blues do not have as many files and therefore might not be enough training data for the model to predict that specific genre.

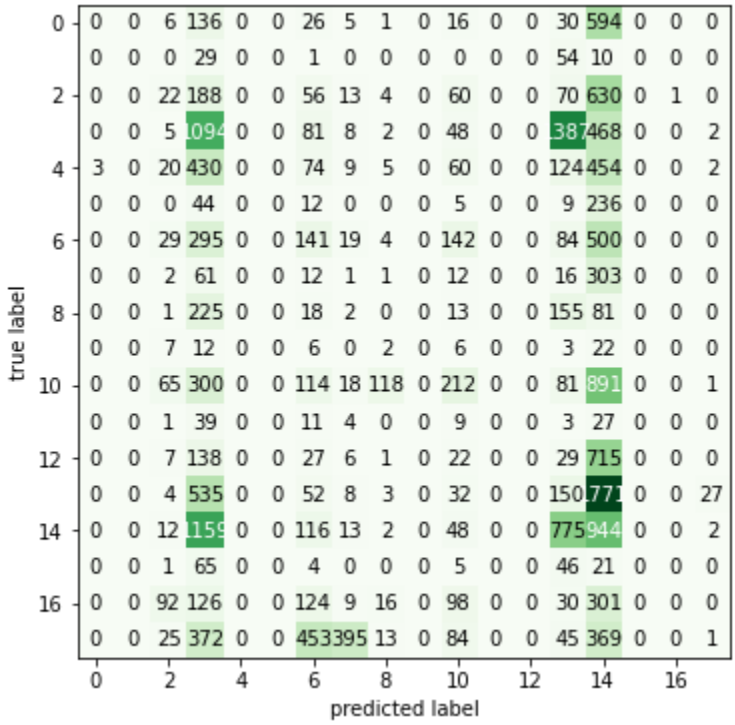


Visual 2:

The visual below is a confusion matric that was created using the data from the final spark data frame depicting the predictions made after the RandomForestClassifier. Although the confusion matrix is very large we can see the original genre's folder number on the y-axis and the predicted

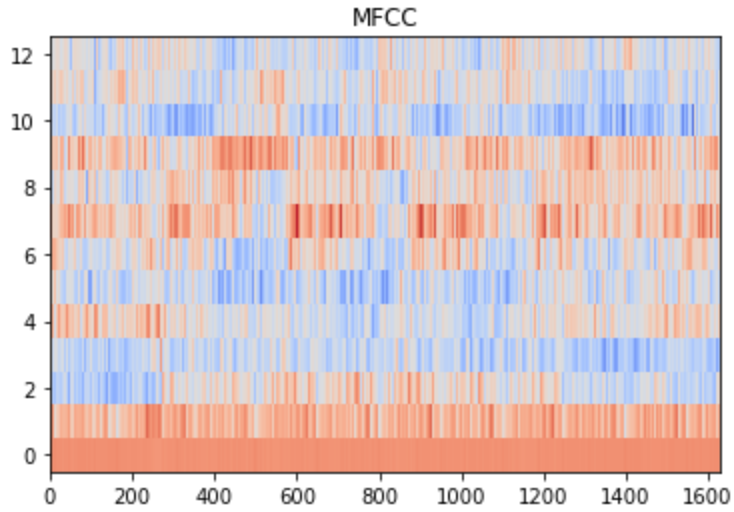
genre folder number on the x-axis. This matrix show's us how many genre labels were accurately predicted for genre 2 ('Punk'), there were 22 correct predictions.

Confusion Matrix:



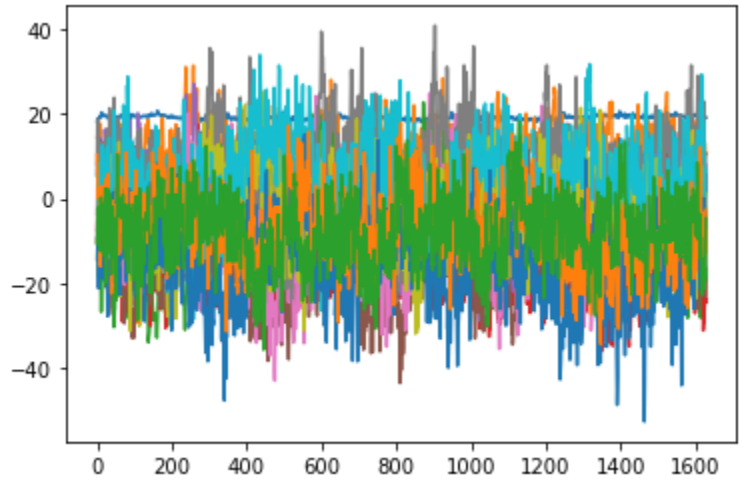
Visual 3:

For the next two plots, I will be visualizing the MFCC features (Mel-frequency cepstral coefficients), which is the main data that is used to make predictions in the pipeline. These features are extracted from the .wav files and for the following 2 visuals I will be using a single '.wav' file from the 'Experimental' genre. The 2 plots are useful when it comes to visualizing spectral characteristics of audio data and when analyzing the audio content of a '.wav' file. The visual below uses the imshow() function to create a plot of a two-dimensional matrix of data.



Visual 4:

This second plot depicts MFCC features as a line plot through the plot() function. The plot below show's the overall trend and shape of MFCC features over the length of the audio file. When extracting the features in the code, I slice up the features into small bits so that the machine is not reading the whole graph at once but instead breaks up the graph below into portions creating an array of values of different frequency points as shown in the pandas data frame when coding and modeling.



Summary and Conclusion:

When beginning my project I attempted to complete it through Amazon EC2 instance however it was very difficult to download such a large file to my amazon s3 bucket as the boto3 module had difficulties unzipping the file due to the size and would kill after a few minutes. Due to this difficulty and trouble connecting to my s3 bucket, I had to complete the pipeline and visualization through Jupyter notebooks using a locally download dataset. When working with

my data I learned that much of the dataset was not needed with only about half of the dataset being used in the final pipeline. Removing all copies and unneeded data could have made processing the data less complicated and more efficient.

When working on the pipeline I learned that using audio files (.wav) files did not work very well with Pyspark. As there are not many particular functions to work with this data type in Pyspark I needed to convert all data to a numeric data type. This made things very difficult as I was unsure of how to approach the pipeline and resulted in a pipeline that was not very effective in predicting the correct genre of music. I was however able to build an effective pipeline in Python3 as there are many packages that are able to read these audio files.

For my final pipeline, I was able to read each .wav file into Jupyter notebook and extract features from these files using the mfcc package which helped read the audio files. With RandomForestClassifiers, my pipeline made predictions based on the data from the extracted mfcc features and the final result showed that this classification method was not very effective with this type of data with only 3 out of 20 predictions being correct. Throughout this project, I learned that although it could be very interesting to work with different types of data, it can also be very difficult to work with them, especially in Pyspark, which works best with numeric data or string data. However, I enjoyed working on this project and had the opportunity to learn about getting past what might seem like a roadblock and keep going forward through new ways.

Below are links to the resources used as references and to help build the code for my project:

- https://www.analyticsvidhya.com/blog/2022/03/implementing-audio-classification-project-using-deep-learning/
- https://www.datatechnotes.com/2021/12/mllib-random-forest-classification.html
- Building Machine Learning Pipelines using Pyspark (analyticsvidhya.com)
- Music Genre Classification Project Using Machine Learning Techniques (analyticsvidhya.com)
- Python - How to Draw Confusion Matrix using Matplotlib - Data Analytics (vitalflux.com)
- matplotlib - How to plot MFCC in Python? - Stack Overflow
- DataCamp (BigData With Pyspark)

Appendix A:

Command to install functions to connect to kaggle website where data is held
> pip3 install kaggle

Bucket Command
> aws s3api create-bucket --bucket music-data-br --region us-east-2
--create-bucket-configuration LocationConstraint=us-east-2

Connecting to Kaggle
> mkdir .kaggle

> nano .kaggle/kaggle.json
Insert API into nano
{"username":"brod940","key":"eb*f***f****a**ec***f****fde*e**"}
Save and exit out of nano
> chmod 600 .kaggle/kaggle.json

Editing Kaggle API to write file to standard output
> nano ~/.local/lib/python3.7/site-packages/kaggle/api/kaggle_api_extended.py
Edit line 1582 to code below:

> if not os.path.exists(outpath) and outpath != "-":

Edit line 1594 to code below:

> with open(outfile, 'wb') if outpath != "-" else os.fdopen(sys.stdout.fileno(), 'wb',
closefd=False) as out:

Save and exit out of nano

Downloading dataset to s3 bucket
> kaggle datasets download --quiet asisheriberto/music-classification-wav -p - |aws s3
cp - s3://music-data-br/music.zip

Checking if bucket was created with stored dataset
> aws s3 ls s3://music-data-br

Unzipping the file in the EC2 instance
$ unzip music-classification-wav.zip
Moving unzipped files to s3 bucket
> aws s3 cp --recursive data_test s3://music-data-br

```
> aws s3 cp --recursive IA s3://music-data-br
> aws s3 cp --recursive TRAIN_V2 s3://music-data-br
```

Appendix B:

Number of files in data_out (Training Files) and in data_test (Test Files)

Training Files

```
>>> import os
>>> dir_path = './IA/data_out/'
>>> def count_files(dir_path):
...     for r, d, f in os.walk(dir_path):
...             for folder in d:
...                     one_folder = os.path.join(r, folder)
...                     print(one_folder, ':', len([name for name in os.listdir(one_folder) if
os.path.isfile(one_folder + "/" + name)]))
...
>>> count_files(dir_path)

./IA/data_out/0 : 3071
./IA/data_out/1 : 3095
./IA/data_out/10 : 796
./IA/data_out/11 : 495
./IA/data_out/12 : 408
./IA/data_out/13 : 306
./IA/data_out/14 : 142
./IA/data_out/15 : 94
./IA/data_out/16 : 94
./IA/data_out/17 : 58
./IA/data_out/2 : 2582
./IA/data_out/3 : 1800
./IA/data_out/4 : 1757
./IA/data_out/5 : 1214
./IA/data_out/6 : 1181
./IA/data_out/7 : 1044
./IA/data_out/8 : 945
./IA/data_out/9 : 814
```

Test Files

```
>>> dir_path = './data_test/'
```

```
>>> def count_files(dir_path):
...     for r, d, f in os.walk(dir_path):
...             for folder in d:
...                     one_folder = os.path.join(r, folder)
...                     print(one_folder, ':', len([name for name in os.listdir(one_folder) if
os.path.isfile(one_folder + "/" + name)]))
...
>>> count_files(dir_path)
./data_test/data_test : 5076
```

List of music genres and count of genres

```
>>> data = pd.read_csv("./IA/train.csv")
>>> genre_list = []
>>> for d in data['genre']:
...     if d not in genre_list:
...             genre_list.append(d)
...
>>> print(genre_list)
['Instrumental', 'Punk', 'Folk', 'Old-Time / Historic', 'Rock', 'International', 'Chiptune /
Glitch', 'Experimental', 'Pop', 'Electronic', 'Hip-Hop', 'Jazz', 'Classical', 'Ambient
Electronic', 'Blues', 'Easy Listening', 'Soul-RnB', 'Country', 'Spoken']
>>> len(genre_list)
19
```

Genres and their respective folder numbers

```
>>> reader = csv.reader(open("./IA/genres.csv", 'r'))
>>> d = {}
>>> for row in reader:
...     k, v = row
...     d[k] = v
...
>>> d
{'genre': 'genre_id', 'Electronic': '0', 'Rock': '1', 'Punk': '2', 'Experimental': '3', 'Hip-Hop': '4',
'Folk': '5', 'Chiptune / Glitch': '6', 'Instrumental': '7', 'Pop': '8', 'International': '9', 'Ambient
Electronic': '10', 'Classical': '11', 'Old-Time / Historic': '12', 'Jazz': '13', 'Country': '14',
'Soul-RnB': '15', 'Spoken': '16', 'Blues': '17', 'Easy Listening': '18'}
```

Count of .wav files in each genre

```
>>> genre_count = {}
>>> for d in data['genre']:
...     if d not in genre_count:
...             genre_count[d] = 1
...     else:
...             genre_count[d] = genre_count[d]+1
...
>>> print(genre_count)
{'Instrumental': 1045, 'Punk': 2584, 'Folk': 1215, 'Old-Time / Historic': 408, 'Rock': 3097,
'International': 814, 'Chiptune / Glitch': 1181, 'Experimental': 1801, 'Pop': 945,
'Electronic': 3073, 'Hip-Hop': 1761, 'Jazz': 306, 'Classical': 495, 'Ambient Electronic':
796, 'Blues': 58, 'Easy Listening': 13, 'Soul-RnB': 94, 'Country': 142, 'Spoken': 94}
```

Appendix C:

Packages for coding and modeling

```
!pip install librosa
!pip install tensorflow
!pip install mlxtend
import IPython.display as ipd
import librosa
import librosa.display
import pandas as pd
from tqdm import tqdm
import numpy as np
import os
import matplotlib as plt

#Starting up Pyspark in Jupyter Notebook
import findspark
findspark.init('C:\spark-3.3.1-bin-hadoop2')
import pyspark
from pyspark.sql import SparkSession
from pyspark import SparkContext
from pyspark.sql import SQLContext
from pyspark.ml.classification import RandomForestClassifier
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from sklearn.metrics import confusion_matrix
```

```
from pyspark.ml.feature import VectorAssembler
from pyspark.ml import Pipeline
```

```
#Read in the data as a pandas dataframe

metadata = pd.read_csv('C:/Users/Brianrod/Downloads/music1/IA/train.csv')

#metadata.head(10)
```

```
#Defining a function that will extract features from each wave file of each folder.
file_name='C:/Users/Brianrod/Downloads/music1/IA/data_out'

def features_extractor(file):
    #load the file (audio)
    #We are able to load and read these files using the Librosa package
    audio, sample_rate = librosa.load(file_name, res_type='kaiser_fast')
    #mfcc is one of the 2 features we extarct as we slow down the rate of the audio to capture
#more information from each frame of the audio file
    mfccs_features = librosa.feature.mfcc(y=audio, sr=sample_rate, n_mfcc=40)
    #for our second set of features we take the mean of the array using the previous feature.
    mfccs_scaled_features = np.mean(mfccs_features.T,axis=0)
    return mfccs_scaled_features
```

```
#Now we use the features_extractor function and append that value into an array
extracted_features=[]
#tqdm will help us see the progress the loop has made and it can take up to 10 minutes.
for index_num,row in tqdm(metadata.iterrows()):
    try:
        #Through the file_name variable we are able to create a path between the .wav files and the
#information contained about the .wav file in the csv file. This helps us know what genre each
#.wav file will be.
        file_name =
os.path.join(os.path.abspath('C:/Users/Brianrod/Downloads/music1/IA/data_out'),str(row["genre
_id"])+'/',str(row["filename"]))
        final_class_labels=row["genre"]
```

```
        data=features_extractor(file_name)
        extracted_features.append([data,final_class_labels])
    #I have placed the below line so that the loop does not end when it faces a file that does not match as a 'wav' file.
    except OSError:
        pass
```

```
### converting extracted_features to Pandas dataframe
extracted_features_df=pd.DataFrame(extracted_features,columns=['feature','genre'])
extracted_features_df
```

```
#We now need to convert the pandas data frame into a spark data frame to be using pyspark.
spark = SparkSession.builder.appName("pandas to spark").getOrCreate()
col = [f'Value{i}' for i in range(len(extracted_features_df['feature'][0]))]
df3 = extracted_features_df.feature.apply(pd.Series)
df3.columns = col
df3['genre'] = extracted_features_df['genre']
df_sp = spark.createDataFrame(df3)
#type(df_sp)
#df_sp.show()
#df_sp.printSchema()
```

```
#The features column stores the data as an array and we will need to transform it into a vector to
#be able to use it.
extracted_features_df[0:1].to_dict()
```

```
#Specifying input columns and the output column where they will be combined which is 'vector'
assembler = VectorAssembler(inputCols=['Value0',
'Value1','Value2','Value3','Value4','Value5','Value6','Value7','Value8',
'Value9','Value10','Value11','Value12','Value13','Value14','Value15','Value16','Value17','Value18','
Value19','Value20',
'Value21','Value22','Value23','Value24',
'Value25','Value26','Value27','Value28','Value29','Value30','Value31','Value32',
'Value33','Value34','Value35','Value36','Value37','Value38','Value39'],
                        outputCol='vector')

#Transforming the data into a vector
final_data = assembler.transform(df_sp)
```

```
#Viewing the vector
final_data.select('vector').show()
```

```
#We now need to remove the 'genre' as it is a string and will give errors when building the
#pipeline as it will only take numeric data.
genre_df = df_sp.select('genre').distinct().toPandas()
genre_df["genre_label"] = genre_df.index + 1
genere_spark_df = spark.createDataFrame(genre_df)
genre_labelled_df = df_sp.join(
    genere_spark_df,
    "genre"
).drop("genre")
```

```
#Now we will build the pipeline to generate predictions of what the genre of music it will be.
#For this pipeline I have used RandomForestClassifier which works better with many target
#variable, in this case 'genres'
forestclassifier = RandomForestClassifier(
    featuresCol='vector',
    labelCol='genre_label'
)

forest_pipeline = Pipeline(
    stages= [assembler, forestclassifier]
)
```

```
# fit the pipeline for the trained data
model = forest_pipeline.fit(genre_labelled_df)
# transform the data
sample_data_train = model.transform(genre_labelled_df)

# view some of the columns generated
sample_data_train.select('vector', 'genre_label', 'rawPrediction', 'probability', 'prediction').show()
```

For accuracy of the model:

```
evaluator=MulticlassClassificationEvaluator(predictionCol="prediction",labelCol="genre_label"
)
acc = evaluator.evaluate(sample_data_train)
print("Prediction Accuracy: ", acc)
```

Appendix D:

Visual 1:

```
import seaborn as sns
import matplotlib.pyplot as plt
plt.figure(figsize=(10, 6))
sns.countplot(metadata['genre'])
plt.title("Count of records in each class")
plt.xticks(rotation="vertical")
plt.show()
```

Visual 2:

For Confusion Matrix:

```
y_pred=sample_data_train.select("prediction").collect()
y_orig=sample_data_train.select("genre_label").collect()
cm = confusion_matrix(y_orig, y_pred)
print("Confusion Matrix:")

import matplotlib as plt
!pip install mlxtend
from mlxtend.plotting import plot_confusion_matrix
fig, ax = plot_confusion_matrix(conf_mat=cm, figsize=(6, 6), cmap=plt.cm.Greens)
```

Visual 3 & 4:

```
import matplotlib.pyplot as plt
import numpy as np
from matplotlib import cm

(rate,sig) = wav.read('C:/Users/Brianrod/Downloads/music1/IA/data_out/3/126.wav')
mfcc_feat = mfcc(sig,rate)

#mfcc_feat = var

ig, ax = plt.subplots()
mfcc_data= np.swapaxes(mfcc_feat, 0 ,1)
cax = ax.imshow(mfcc_data, interpolation='nearest', cmap=cm.coolwarm, origin='lower',
aspect='auto')

ax.set_title('MFCC')
```

```
#Showing mfcc_data
plt.show()
#Showing mfcc_feat
plt.plot(mfcc_feat)
plt.show()
```