Deliverable II

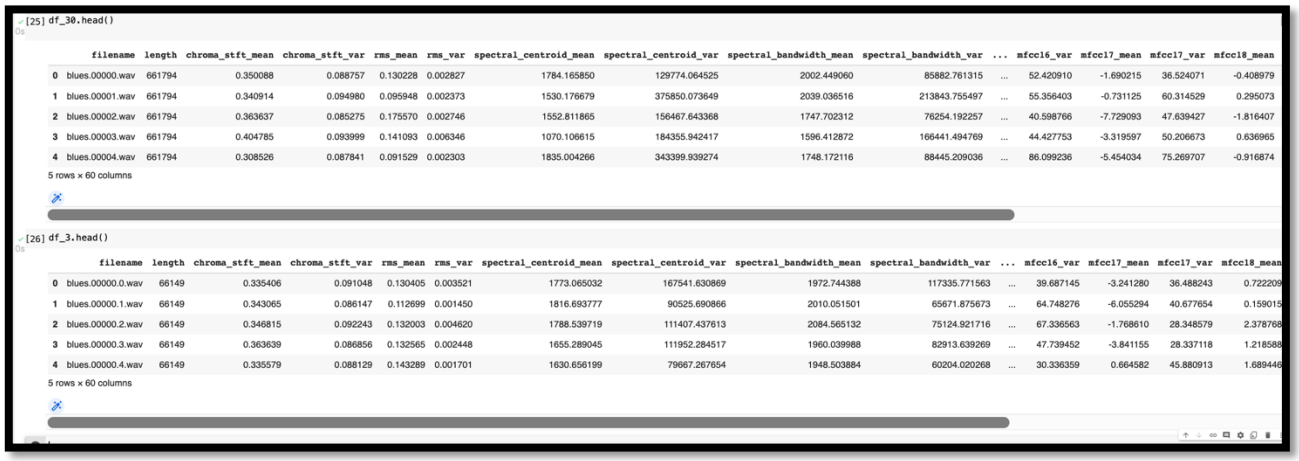
By: Karim Jabbour
ID: 260931803
Date: Mar 13, 2022
Ref:

- https://blog.clairvoyantsoft.com/music-genre-classification-using-cnn-ef9461553726
- https://www.analyticsvidhya.com/blog/2021/06/music-genres-classification-using-deep-learning-techniques/

Code: https://colab.research.google.com/drive/1srbDlZmcAscGTV6HIokShUzz9k6DJjk5?usp=sharing

1. Problem: The goal of this project is to use machine learning to classify music audio files into their respective genres.

2. Data Preprocessing:
   - The data is the GTZAN Genre classification dataset that contains 1000 audio samples split equally into 10 genres with 100 tracks per genre.
   - Each track is 30 seconds long.
   - Files provides include:
     - Genres Original: the folder with the 1000 audio tracks
     - Images Original: a folder with images for each audio file
     - 2 CSVs: Represent audio features for each audio file such that there is a mean and variance over multiple features . One is for the 30 sec clips and the other involves 3 sec clips.
   - CSV Files sample screenshot (features_30_sec.csv & features_3_sec.csv):



```
[25] df_30.head()
```

	filename	length	chroma_stft_mean	chroma_stft_var	rms_mean	rms_var	spectral_centroid_mean	spectral_centroid_var	spectral_bandwidth_mean	spectral_bandwidth_var	...	mfcc16_var	mfcc17_mean	mfcc17_var	mfcc18_mean
0	blues.00000.wav	661794	0.350088	0.088757	0.130228	0.002827	1784.165850	129774.064525	2002.449060	85882.761315	...	52.420910	-1.690215	36.524071	-0.408979
1	blues.00001.wav	661794	0.340914	0.094980	0.095948	0.002373	1530.176679	375850.073649	2039.036516	213843.755497	...	55.356403	-0.731125	60.314529	0.295073
2	blues.00002.wav	661794	0.363637	0.085275	0.175570	0.002746	1552.811865	156467.643368	1747.702312	76254.192257	...	40.598766	-7.729093	47.639427	-1.816407
3	blues.00003.wav	661794	0.404785	0.093999	0.141093	0.006346	1070.106615	184355.942417	1596.412872	166441.494769	...	44.427753	-3.319597	50.206673	0.636965
4	blues.00004.wav	661794	0.308526	0.087841	0.091529	0.002303	1835.004266	343399.939274	1748.172116	88445.209036	...	86.099236	-5.454034	75.269707	-0.916874

5 rows × 60 columns

```
[26] df_3.head()
```

	filename	length	chroma_stft_mean	chroma_stft_var	rms_mean	rms_var	spectral_centroid_mean	spectral_centroid_var	spectral_bandwidth_mean	spectral_bandwidth_var	...	mfcc16_var	mfcc17_mean	mfcc17_var	mfcc18_mean
0	blues.00000.0.wav	66149	0.335406	0.091048	0.130405	0.003521	1773.065032	167541.630869	1972.744388	117335.771563	...	39.687145	-3.241280	36.488243	0.722209
1	blues.00000.1.wav	66149	0.343065	0.086147	0.112699	0.001450	1816.693777	90525.690866	2010.051501	65671.875673	...	64.748276	-6.055294	40.677654	0.159015
2	blues.00000.2.wav	66149	0.346815	0.092243	0.132003	0.004620	1788.539719	111407.437613	2084.565132	75124.921716	...	67.336563	-1.768610	28.348579	2.378768
3	blues.00000.3.wav	66149	0.363639	0.086856	0.132565	0.002448	1655.289045	111952.284517	1960.039988	82913.639269	...	47.739452	-3.841155	28.337118	1.218588
4	blues.00000.4.wav	66149	0.335579	0.088129	0.143289	0.001701	1630.656199	79667.267654	1948.503884	60204.020268	...	30.336359	0.664582	45.880913	1.689446

5 rows × 60 columns

- Shape of data reveals that the 3 second clips contains around 10x the rows (samples) as the 30 sec clips:

```
[28] df_30.shape
(1000, 60)

df_3.shape
(9990, 60)
```

- Provided Features:

```
df_3.dtypes

filename                  object
length                     int64
chroma_stft_mean         float64
chroma_stft_var          float64
rms_mean                 float64
rms_var                  float64
spectral_centroid_mean   float64
spectral_centroid_var    float64
spectral_bandwidth_mean  float64
spectral_bandwidth_var   float64
rolloff_mean             float64
rolloff_var              float64
zero_crossing_rate_mean  float64
zero_crossing_rate_var   float64
```

- Loading audio files as time series (librosa.load):
  - I will choose sampling rate of 44.1 kHZ (common choice)

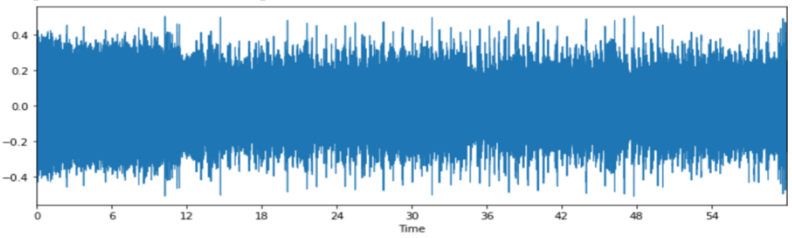
```
librosa.load(audio1,sr=44100)
```

```
(array([0.11283495, 0.13125767, 0.12303283, ..., 0.05828203, 0.03469915,
        0.01158567], dtype=float32), 44100)
```

- Wave form (raw audio form):

```
plt.figure(figsize=(12,4))
librosa.display.waveplot(librosa.load(audio1,sr=44100)[0])
```

```
<matplotlib.collections.PolyCollection at 0x7fcba77e05d0>
```



- Feature Extraction (LabelEncoder):
  - Using LabelEncoder we give each label (ex: blues, rock,etc.) a specific numerical code

```
y=LabelEncoder().fit_transform(label_column)
print(y)
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2]
```

- StandardScaler:
  - Needed to drop the FileName column
  - Important to do for machine learning models

```
X=fit.fit_transform(array1)
print(X)
```

- ```
  [[-0.13282213 -0.35013678  0.31258717 ... -0.30059734  0.60406407
    -0.51298758]
   [-0.13282213 -0.46248155  1.11757233 ... -0.40708699  0.42412706
    -0.53842129]
   [-0.13282213 -0.18422456 -0.13770124 ... -0.52729705 -0.29618888
    -0.8749539 ]
   ...
   [-0.13282213  0.65463736 -1.43198917 ... -0.63865065 -0.26361549
    -0.89060474]
   [-0.13282213 -0.19833855  0.66814351 ... -0.5114848  -0.65064889
    -0.63768256]
   [-0.13282213 -0.2483391  -0.05894495 ...  0.16033426  0.5868411
    -0.4526752 ]]
  ```
  ```

- Splitting into Training and Testing (3:1) :

```
[66] X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.33)

print(len(X_train))
print(len(X_test))
print(len(y_train))
print(len(y_test))
```

```
670
330
670
330
```

3. Model:
    - Referencing https://blog.clairvoyantsoft.com/music-genre-classification-using-cnn-ef9461553726 , to show why I am using CNN neural network model:

	With data processing			Without data processing		
	Train	CV	Test	Train	CV	Test
Support Vector Machine	.97	.60	.60	.75	.32	.28
K-Nearest Neighbors	1.00	.52	.54	1.00	.21	.21
Feed-forward Neural Network	.96	.55	.54	.64	.26	.25
Convolution Neural Network	.95	.84	**.82**	.85	.59	.53

    - Properties of CNN Model
        - Optimizer Used: ADAM
        - Hidden Layers activation function: RELU
        - Output layer function: SOFTMAX
        - Loss calculation: Sparse Categorical Cross-entropy
        - Dropout layers : prevent over-fitting
        - Epochs: 1000 (30 sec) & 600 (3 sec)

```
[25] def trainModel(model,epochs,optimizer):
       batch_size=128
       model.compile(optimizer=optimizer,loss='sparse_categorical_crossentropy',metrics='accuracy')
       return model.fit(X_train,y_train,validation_data=(X_test,y_test),epochs=epochs,batch_size=batch_size)
```

```
[26] def plotValidate(history):
       print("Validation Accuracy: ", max(history.history["val_accuracy"]))
       pd.DataFrame(history.history).plot(figsize=(12,6))
       plt.show()
```

```
model=k.models.Sequential(
    [

      k.layers.Dense(512,activation='relu',input_shape=(X_train.shape[1],)),
      k.layers.Dropout(0.2),

      k.layers.Dense(256,activation='relu'),
      k.layers.Dropout(0.2),

      k.layers.Dense(128,activation='relu'),
      k.layers.Dropout(0.2),

      k.layers.Dense(64,activation='relu'),
      k.layers.Dropout(0.2),


      k.layers.Dense(32,activation='relu'),
      k.layers.Dropout(0.2),

      k.layers.Dense(10,activation='softmax'),




    ])
print(model.summary())
model_history=trainModel(model=model,epochs=1000,optimizer='adam')
```

4. Results:
    a. Results show that data of 3-sec clips produced much higher accuracy in CNN model even with almost half number of epochs (600 vs 1000)
    b. 91.9% vs 75.15%

```
[28] test_loss, test_acc= model.evaluate(X_test,y_test,batch_size=128)
     print("The test Loss is :", test_loss)
     print("\nThe best test accuracy is: ", test_acc*100)

     3/3 [==============================] - 0s 5ms/step - loss: 2.7657 - accuracy: 0.7515
     The test Loss is : 2.7656843662261963

     The best test accuracy is:  75.15151500701904
```

```
test_loss, test_acc= model.evaluate(X_test,y_test,batch_size=128)
print("The test Loss is :", test_loss)
print("\nThe best test accuracy is: ", test_acc*100)

26/26 [==============================] - 0s 3ms/step - loss: 0.7018 - accuracy: 0.9193
The test Loss is : 0.7017708420753479

The best test accuracy is:  91.93205833435059
```

    c. Results are good for the 3_sec dataset

5. Next Steps:
    a. I would like to see how I may extract my own features from the audio files
    b. I would like to see how neural networks can be applied to to ther files such as the images
    c. I would like to allow for a predictor application that can take one audio file as inpu and classify it