Music Genre classification

zz

November 2023

1 GTZAN dataset

Data set description : Dataset description

The dataset used is GTZAN (the famous GTZAN dataset, the MNIST of sounds)

The GTZAN dataset contains 1000 audio files. Contains a total of 10 genres, each genre contains 100 audio files

1. Blues
2. Classical
3. Country
4. Disco
5. Hip-hop
6. Jazz
7. Metal
8. Pop
9. Reggae
10. Rock

Genres original

¿ A compilation of ten genres, each with 100 audio recordings, each lasting 30 seconds (the famous GTZAN dataset, the MNIST of sounds)

Images original

¿ Each audio file has a visual representation. Neural networks are one technique to classify data because they usually take in some form of picture representation.

CSV files

¿ The audio files' features are contained within. Each song lasts for 30 seconds long has a mean and variance computed across several features taken from an audio file in one file. The songs are separated into 3 second audio files in the other file, which has the same format.

Dive deep with me into the world of data! While the dataset has graciously done some of the heavy lifting by extracting certain features for us, we're about to embark on a thrilling journey to uncover the mysteries behind these features. Let's truly understand their essence and the magic of how to obtain them. So, why not take a bold step? Let's delete those pre-existing folders and craft our own path in this adventure!

2 properties of sounds

2.1 Wave Form

2.1.1 Definition

Sound is type of energy made by vibrations. When any object vibrates,it causes movement in the air =¿ these particles bump into the one close to them. This movement call sound waves.
A waveform represents the shape and form of a signal (like a sound) in terms of its amplitude over time
A waveform plot is a visual representation of an audio signal. It displays how the amplitude of the sound signal varies with time
Display Waveform :

```
import matplotlib.pyplot as plt

# Wave form of the audio
plt.figure(figsize=(12,6))
librosa.display.waveshow(data, color="#2B4F72", alpha
    = 0.5)
plt.show()
```

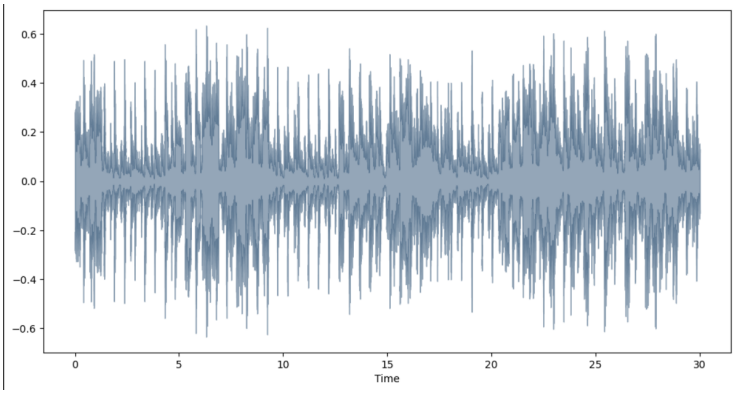

Figure 1: Here is the result

2.2 Sampling Rate

2.2.1 Definition

Sound, in its natural form, is a continuous analog signal. To enable digital processing of these signals, they must be converted into a discrete digital format through a process called 'sampling'. The 'sampling rate', quantified in Hertz

(Hz), denotes the number of samples taken per second from the continuous signal.

2.2.2 Example Code

```
data, sr = librosa.load(data_path)

plt.figure(figsize=(12,6))
librosa.display.waveshow(data, color="#2B4F72", alpha
    = 0.5, sr=22050)
plt.show()
```

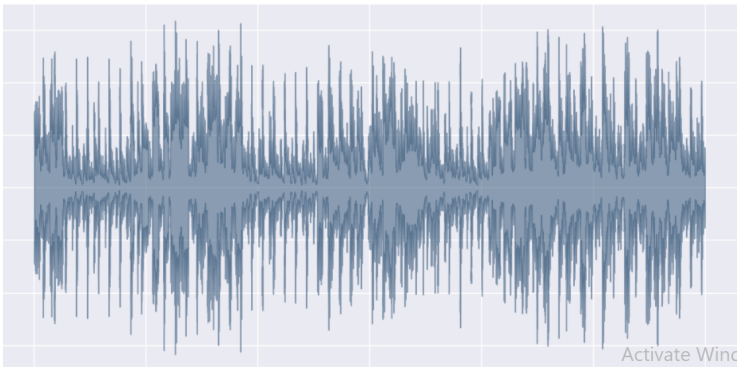

Figure 2: Here is the result

2.3 Root Mean Square Enenrgy

2.3.1 Definition

RMS-E used to measure the magnitude of "Volume" of an audio signal
Why is it important ?

RMS is a crucial metric in audio because our perception of loudness is more closely related to the RMS level of a sound rather than its peak level. Two sounds with the same peak level can have different perceived loudness if one has a higher RMS level. This is why RMS is often used in audio normalization and compression algorithms to ensure consistent perceived loudness across different audio tracks or segments.

```
# Let load .wav file with default sampling rate of
    22,050Hz
data, sr = librosa.load(data_path)

# Compute RMS
```

```
rms = librosa.feature.rms(y=data)

# Plot RMS
plt.figure(figsize=(10, 4))
librosa.display.waveshow(data, sr=sr, alpha=0.4, label
    ='Audio Waveform')
plt.plot(librosa.times_like(rms[0], sr=sr), rms[0],
    color='r', label='RMS')
plt.legend(loc='upper right')
plt.title('RMS over Time')
plt.tight_layout()
plt.show()
```

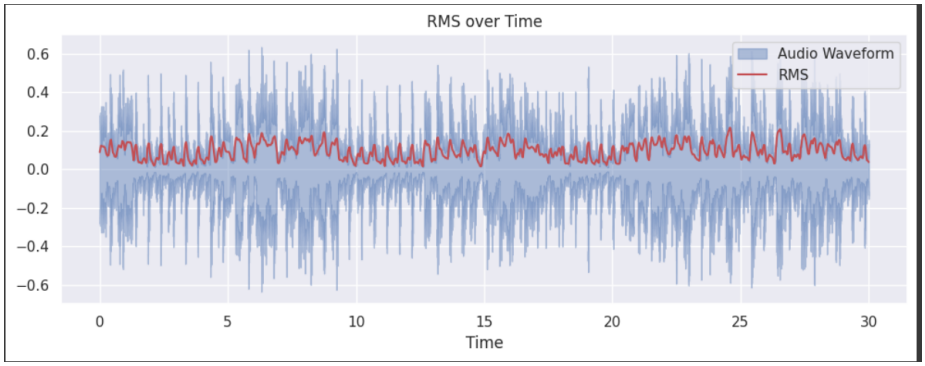

Figure 3: Here is the result

2.4 Zero-Crossing Rate

2.4.1 Definition

The Zero Crossing Rate (ZCR) is a fundamental feature in audio signal processing and analysis. It represents the rate at which a signal changes its sign, i.e., goes from positive to negative or vice versa.
This tell us :

- Frequency Content: A higher zero crossing rate often indicates the presence of higher frequencies in the signal, while a lower rate suggests lower frequencies. This makes sense because higher frequency components will cause the signal to oscillate (cross zero) more frequently.
- Noise Detection: Noise, especially of the high-frequency variety, can result in increased zero crossings. Thus, an unexpectedly high ZCR can sometimes be an indicator of noise.

- Speech Analysis: In speech processing, the ZCR can help distinguish between voiced and unvoiced speech segments. Voiced segments, like vowels, tend to have a lower ZCR, while unvoiced segments, like consonants such as "s" or "f", tend to have a higher ZCR.

```
## Example of ZCR

# Let load .wav file with default sampling rate of
    22,050Hz
data, sr = librosa.load(data_path)

# Calculate ZRC of the first 1000 data point of our
    song
n0 = 0
n1 = 1000
plt.figure(figsize=(14, 5))
plt.plot(data[n0:n1])
plt.grid()

zero_crossings = librosa.zero_crossings(data[n0:n1],
    pad=False)
print(f'ZRC = {sum(zero_crossings)}')
```

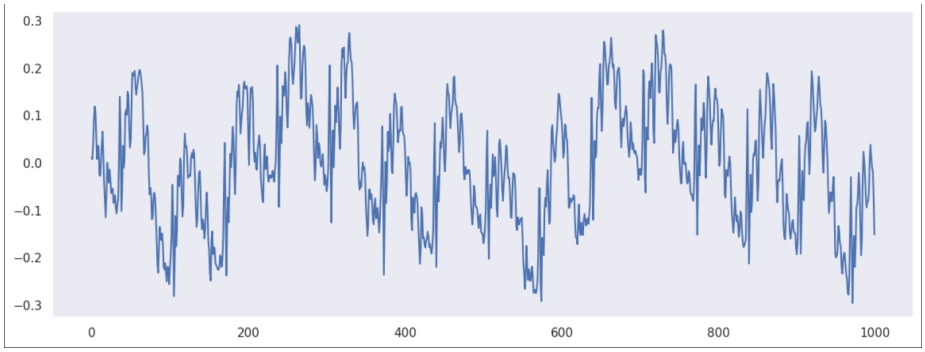

Figure 4: Here is the result

2.5 Special roll-off

2.5.1 Definition

Spectral roll-off is a measure used in audio signal processing and analysis to characterize the spectral shape of an audio signal. It represents a threshold below which a specified percentage of the total spectral energy is contained. The spectral roll-off point is typically defined as the frequency below which a

certain percentage (often 85 to 95 percent) of the total spectral energy lies. In other words, if you were to sum the magnitudes of the spectral components from the lowest frequency up to the roll-off frequency, it would account for, say, 85 percent of the total spectral energy of the signal.

```
import sklearn.preprocessing

# Function to normalize an array
def normalize(x, axis=0):
    return sklearn.preprocessing.minmax_scale(x, axis=
        axis)

# Load the audio file with default sampling rate of
    22,050Hz
data, sr = librosa.load(data_path)

# Compute and norm the spectral roll-off
rolloff_85 = librosa.feature.spectral_rolloff(y=data,
    sr=sr, roll_percent=0.85)[0]
rolloff_85_norm = normalize(rolloff_85)

# Plot the waveform and normalized spectral roll-offs
plt.figure(figsize=(12, 6))
librosa.display.waveshow(data, sr=sr, alpha=0.4, label
    ='Waveform')
times = librosa.times_like(rolloff_85)
plt.plot(times, rolloff_85_norm, color='r', label='85%
    roll-off (normalized)')

plt.title('Waveform with Normalized Spectral Roll-offs
    ')
plt.legend(loc='upper right')
plt.tight_layout()
plt.show()
```

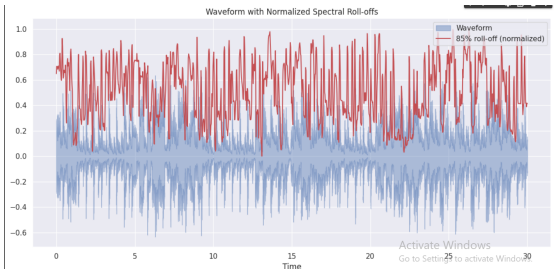

Figure 5: Here is the result

2.6 Spectral Centroid

2.6.1 Definition

The spectral centroid is a measure used in digital signal processing to characterize the "brightness" or "sharpness" of a sound. It provides a center of gravity or a "balance point" for the spectrum of a sound signal, indicating where the "center of mass" of the spectrum is located.

3 Music classification project

3.1 Initialize data set into feature$_{3s}ec.csv$

```
from glob import glob
import pandas as pd

num_segment=10
num_mfcc=20
sample_rate=22050
n_fft=2048
hop_length=512

my_csv={"filename":[], "chroma_stft_mean": [], "
    chroma_stft_var": [], "rms_mean": [], "rms_var":
    [], "spectral_centroid_mean": [],
        "spectral_centroid_var": [], "
            spectral_bandwidth_mean": [], "
            spectral_bandwidth_var": [], "rolloff_mean"
            : [], "rolloff_var": [],
        "zero_crossing_rate_mean": [], "
            zero_crossing_rate_var": [], "harmony_mean"
            : [], "harmony_var": [], "perceptr_mean":
            [],
        "perceptr_var": [], "tempo": [], "mfcc1_mean":
             [], "mfcc1_var" : [], "mfcc2_mean" : [], "
            mfcc2_var" : [],
        "mfcc3_mean" : [], "mfcc3_var" : [], "
            mfcc4_mean" : [], "mfcc4_var" : [], "
            mfcc5_mean" : [],
        "mfcc5_var" : [], "mfcc6_mean" : [], "
            mfcc6_var" : [], "mfcc7_mean" : [], "
            mfcc7_var" : [],
        "mfcc8_mean" : [], "mfcc8_var" : [], "
            mfcc9_mean" : [], "mfcc9_var" : [], "
            mfcc10_mean" : [],
```

```
17          "mfcc10_var" : [], "mfcc11_mean" : [], "
                mfcc11_var" : [], "mfcc12_mean" : [], "
                mfcc12_var" : [],
18          "mfcc13_mean" : [], "mfcc13_var" : [], "
                mfcc14_mean" : [], "mfcc14_var" : [], "
                mfcc15_mean" : [],
19          "mfcc15_var" : [], "mfcc16_mean" : [], "
                mfcc16_var" : [], "mfcc17_mean" : [], "
                mfcc17_var" : [],
20          "mfcc18_mean" : [], "mfcc18_var" : [], "
                mfcc19_mean" : [], "mfcc19_var" : [], "
                mfcc20_mean" : [],
21          "mfcc20_var":[], "label":[]}
```

With :

- num segment is number of subdivisions of original data
- num mfcc is the MFCC number system number to be calculated, in the GTZAN set is 20

3.2 find the features and Y

```
1 dataset_path="/content/GTZAN/genres_original"
2 audio_files = glob(dataset_path + "/*/*")
3 genre = glob(dataset_path + "/*")
4 n_genres=len(genre)
5 genre=[genre[x].split('/')[-1] for x in range(n_genres
      )]
```

3.3 Calculate mean and var of each features and insert into feature 3 csv file

```
1  samples_per_segment = int(sample_rate*30/num_segment)
2
3  genre=""
4  for f in sorted(audio_files):
5      if genre!=f.split('/')[-2]:
6          genre=f.split('/')[-2]
7          print("Processsing " + genre + "...")
8      fname=f.split('/')[-1]
9      try:
10         y, sr = librosa.load(f, sr=sample_rate)
11     except:
12         continue
```

```
13
14      for n in range(num_segment):
15          y_seg = y[samples_per_segment*n:
                samples_per_segment*(n+1)]
16          #Chromagram
17          chroma_hop_length = 512
18          chromagram = librosa.feature.chroma_stft(y=
                y_seg, sr=sample_rate, hop_length=
                chroma_hop_length)
19          my_csv["chroma_stft_mean"].append(chromagram.
                mean())
20          my_csv["chroma_stft_var"].append(chromagram.
                var())
21
22          #Root Mean Square Energy
23          RMSEn= librosa.feature.rms(y=y_seg)
24          my_csv["rms_mean"].append(RMSEn.mean())
25          my_csv["rms_var"].append(RMSEn.var())
26
27          #Spectral Centroid
28          spec_cent=librosa.feature.spectral_centroid(y=
                y_seg)
29          my_csv["spectral_centroid_mean"].append(
                spec_cent.mean())
30          my_csv["spectral_centroid_var"].append(
                spec_cent.var())
31
32          #Spectral Bandwith
33          spec_band=librosa.feature.spectral_bandwidth(y
                =y_seg,sr=sample_rate)
34          my_csv["spectral_bandwidth_mean"].append(
                spec_band.mean())
35          my_csv["spectral_bandwidth_var"].append(
                spec_band.var())
36
37          #Rolloff
38          spec_roll=librosa.feature.spectral_rolloff(y=
                y_seg,sr=sample_rate)
39          my_csv["rolloff_mean"].append(spec_roll.mean()
                )
40          my_csv["rolloff_var"].append(spec_roll.var())
41
42          #Zero Crossing Rate
43          zero_crossing=librosa.feature.
                zero_crossing_rate(y=y_seg)
```

```
        my_csv["zero_crossing_rate_mean"].append(
            zero_crossing.mean())
        my_csv["zero_crossing_rate_var"].append(
            zero_crossing.var())

        #Harmonics and Perceptrual
        harmony, perceptr = librosa.effects.hpss(y=
            y_seg)
        my_csv["harmony_mean"].append(harmony.mean())
        my_csv["harmony_var"].append(harmony.var())
        my_csv["perceptr_mean"].append(perceptr.mean()
            )
        my_csv["perceptr_var"].append(perceptr.var())

        #Tempo
        tempo, _ = librosa.beat.beat_track(y=y_seg, sr
            =sample_rate)
        my_csv["tempo"].append(tempo)

        #MFCC
        mfcc=librosa.feature.mfcc(y=y_seg,sr=
            sample_rate, n_mfcc=num_mfcc, n_fft=n_fft,
            hop_length=hop_length)
        mfcc=mfcc.T


        fseg_name='.'.join(fname.split('.')[:2])+f'.{n
            }.wav'
        my_csv["filename"].append(fseg_name)
        my_csv["label"].append(genre)
        for x in range(20):
            feat1 = "mfcc" + str(x+1) + "_mean"
            feat2 = "mfcc" + str(x+1) + "_var"
            my_csv[feat1].append(mfcc[:,x].mean())
            my_csv[feat2].append(mfcc[:,x].var())
    print(fname)

df = pd.DataFrame(my_csv)
df.to_csv('/content/GTZAN/features_3_sec.csv', index=
    False)
```

and then after that we get a table :

```
df = pd.read_csv("/content/GTZAN/features_3_sec.csv")
df.head()
```

	filename	chroma_stft_mean	chroma_stft_var	rms_mean	rms_var	spectral_centroid_mean	spectral_centroid_var	spectral_bandwidth_mean	spect
0	blues.00000.0.wav	0.335555	0.090997	0.130189	0.003559	1773.358004	169450.829520	1972.334258	
1	blues.00000.1.wav	0.343523	0.086782	0.112119	0.001491	1817.244034	90766.297254	2010.751494	
2	blues.00000.2.wav	0.347746	0.092495	0.130895	0.004552	1790.722357	110071.206973	2088.184750	
3	blues.00000.3.wav	0.363863	0.087207	0.131349	0.002338	1660.545231	109496.936296	1967.920582	
4	blues.00000.4.wav	0.335481	0.088482	0.142370	0.001734	1634.465077	77425.419232	1954.633566	

5 rows × 59 columns

Figure 6: Shape = 9990x59

3.4 Encoding y and X

3.4.1 Encoding y

```python
from sklearn.preprocessing import LabelEncoder

# Label Encoding - encod the categorical classes with
    numerical integer values for training

# Blues - 0
# Classical - 1
# Country - 2
# Disco - 3
# Hip-hop - 4
# Jazz - 5
# Metal - 6
# Pop - 7
# Reggae - 8
# Rock - 9

encoder=LabelEncoder()
y=encoder.fit_transform(y)
y
```

3.4.2 Encoding x

```python
#scaling
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
X=scaler.fit_transform(X)
```

it use (Z-score formular) to convert mean = 0, stardard deviation = 1

3.5 MLP model

library :

```
import os
import numpy as np

import torch
from torch import nn, optim
from torch.functional import F
from torch.utils.data import DataLoader, TensorDataset
```

Simple MLP model :

```
class MLP(nn.Module):
    def __init__(self, input_size):
        super(MLP, self).__init__()
        self.flatten = nn.Flatten()
        self.fc1 = nn.Linear(input_size, 512)
        self.fc2 = nn.Linear(512, 256)
        self.fc3 = nn.Linear(256, 128)
        self.fc4 = nn.Linear(128, 64)
        self.fc5 = nn.Linear(64, 32)
        self.fc6 = nn.Linear(32, 10)
        self.dropout = nn.Dropout(0.2)

    def forward(self, x):
        x = self.flatten(x)
        x = F.relu(self.fc1(x))
        x = self.dropout(x)
        x = F.relu(self.fc2(x))
        x = self.dropout(x)
        x = F.relu(self.fc3(x))
        x = self.dropout(x)
        x = F.relu(self.fc4(x))
        x = self.dropout(x)
        x = F.relu(self.fc5(x))
        x = self.dropout(x)
        x = self.fc6(x)
        return x
```

Initialize Hyperparameter :

```
input_size = X_train.shape[1]
model = MLP(input_size)

criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr
   =0.000146)
num_epochs = 300
batch_size = 256
```

```
8
9  train_dataset = TensorDataset(torch.tensor(X_train),
       torch.tensor(y_train))
10 train_loader = DataLoader(train_dataset, batch_size=
       batch_size, shuffle=True)
11
12 val_dataset = TensorDataset(torch.tensor(X_test),
       torch.tensor(y_test))
13 val_loader = DataLoader(val_dataset, batch_size=
       batch_size, shuffle=False)
14
15 step = 0
```

Training :

```
for epoch in range(num_epochs):
    model.train()
    for inputs, labels in train_loader:
        optimizer.zero_grad()
        outputs = model(inputs.float())
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        if step % 100 ==0:
          print(f"Step {step}, Train Loss: {loss.item
              ():.4f}")
        step += 1

    # Validation
    model.eval()
    val_loss = 0.0
    correct = 0
    total = 0
    with torch.no_grad():
        for inputs, labels in val_loader:
            outputs = model(inputs.float())
            loss = criterion(outputs, labels)
            val_loss += loss.item()
            _, predicted = outputs.max(1)
            total += labels.size(0)
            correct += predicted.eq(labels).sum().item
                ()

    val_loss /= len(val_loader)
    val_accuracy = 100 * correct / total
```

```
30      print(f"Epoch {epoch+1}/{num_epochs}, Validation
            Loss: {val_loss:.4f}, Validation Accuracy: {
            val_accuracy:.2f}%")
```