notebook

December 2, 2021

1 Music Genre Classification

Wan Yee Ki 1155143499, AIST2010 Project

Remark:

1. librosa.load is dfferent from scipy.io.wavfile.read
   - librosa.load() normalizes the data (betwenn 1 and -1)
   - scipy.io.wavfile.read() does not normalize the data
2. This notebook has been tested on Windows 10 64-bit only

PC Environment:

1. Intel i7-9750H
2. NVIDIA GTX 1660 TI

Python Environment (Python 3.9.6):

- audiomentation==0.19.1
- tqdm==4.62.2
- tensorflow==2.5.0
- spotipy==2.19.0
- seaborn==0.11.2
- scipy==1.7.1
- scikit-learn==1.0
- pandas==1.3.3
- pedalboard==0.3.8
- numpy==1.20.3
- multiprocess==0.70.12.2
- matplotlib==3.4.2
- librosa==0.8.1
- youtube_dl==2021.6.6

Reference:

1. https://klyshko.github.io/teaching/2019-02-22-teaching
2. Building Machine Learning Systems
3. Tensorflow Quick Start Guide

[1]:
```
import soundfile as sf
import matplotlib.pyplot as plt
import tensorflow as tf
```

```python
import numpy as np
import pandas as pd
import multiprocess as mp
import urllib.request
import librosa.display
import librosa
import tarfile
import os
import sys
import gc
import scipy
import scipy.io.wavfile
from tqdm import tqdm
import multiprocess as mp
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix
from sklearn.decomposition import PCA
from sklearn import preprocessing


plt.style.use('seaborn')
```

1.1 Constant

[2]:
```python
DATA_DIR = "sound_data" # folder for storing data
DATA_URL = 'http://opihi.cs.uvic.ca/sound/genres.tar.gz' # url for downloading␣
↪data
SOX_PATH = r'C:\Program Files (x86)\sox-14-4-2' # path to sox (download from␣
↪https://sourceforge.net/projects/sox/)
SOX = SOX_PATH + '\sox.exe'
FFT_TOTAL = 2000 # number of frequencies to be extracted while performing FFT
```

1.2 Data Preparation

1.2.1 Downloading data

[3]:
```python
# create data folder if not exist
if not os.path.exists(DATA_DIR):
    os.makedirs(DATA_DIR)
```

[4]:
```python
# download GTZAN dataset if not exist
# It takes around 45 minutes to download
# Total size: 1.2GB
if not os.path.exists(f"{DATA_DIR}/genres.tar.gz"):
```

```
    urllib.request.urlretrieve(DATA_URL, f"{DATA_DIR}/genres.tar.gz")
```

1.2.2 Extracting files

[5]:
```
# extract the downloaded file
if not os.path.exists(f"{DATA_DIR}/genres"):
    with tarfile.open(f"{DATA_DIR}/genres.tar.gz") as f:
        f.extractall(f"{DATA_DIR}")
```

[6]:
```
# list of all genres available for classification
GENRES = list(os.walk(f"{DATA_DIR}/genres"))[0][1]
print(GENRES)
```

```
['blues', 'classical', 'country', 'disco', 'hiphop', 'jazz', 'metal', 'pop',
'reggae', 'rock']
```

1.3 Utils

At the very begining, I think it would be possible to retrieve song metadata using Spotify API.

However, I soon found that most of the songs in Spotify are missing the field "genre"

The code is kept here for reference

Reference:

1. https://github.com/christianlomboy/MIR-Genre-Predictor

1.3.1 Fetch song metadata from Spotify

You may generate your own client id and client secret in the following website:

https://developer.spotify.com/dashboard/login

[ ]:
```
import spotipy
SPOTIFY_CLIENT_ID =
SPOTIFY_CLIENT_SECRET =
```

[ ]:
```
# Create Spotify Search Engine object
def get_spotify_engine(client_id, client_secret):
    return spotipy.Spotify(auth_manager=spotipy.oauth2.
 ↪SpotifyClientCredentials(client_id=client_id, client_secret=client_secret))

# fetch song information from spotify
def fetch_song_info(sp, song_name):
    results = sp.search(q=song_name, type='track')
    items = results['tracks']['items']

    album_genre = sp.
 ↪album(items[0]["album"]["external_urls"]["spotify"])['genres']
```

```
    artist_genre = sp.
↪artist(items[0]["artists"][0]["external_urls"]["spotify"])['genres']
    print(f"Song Name: {items[0]['name']}")
    print(f"Song Artist: {items[0]['artists'][0]['name']}")
    print(f"Song Album: {items[0]['album']['name']}")
    print(f"Album Genre: {album_genre}")
    print(f"Artist Genre: {artist_genre}")
    print(f"Track id: {items[0]['id']}")
    print("-- -- -- -- -- -- --")

    features = sp.audio_features(tracks = items[0]['id'])
    print(f"Danceability: {features[0]['danceability']}")
    print(f"Energy: {features[0]['energy']}")
    print(f"Key: {features[0]['key']}")
    print(f"Lounness: {features[0]['loudness']}")
    print(f"Mode: {features[0]['mode']}")
    print(f"Speechiness: {features[0]['speechiness']}")
    print(f"Acousticness: {features[0]['acousticness']}")
    print(f"Instrumentalness: {features[0]['instrumentalness']}")
    print(f"Liveness: {features[0]['liveness']}")
    print(f"Valence: {features[0]['valence']}")
    print(f"Tempo: {features[0]['tempo']}")
    print(f"Duration: {features[0]['duration_ms']//1000}s")
    print(f"Time signature: {features[0]['time_signature']}")
```

[ ]:
```
sp = get_spotify_engine(SPOTIFY_CLIENT_ID, SPOTIFY_CLIENT_SECRET)
```

[ ]:
```
fetch_song_info(sp, 'River flows to you')
```

```
Song Name: River Flows In You
Song Artist: Yiruma
Song Album: Yiruma 2nd Album 'First Love' (The Original & the Very First
Recording)
Album Genre: []
Artist Genre: ['korean instrumental', 'neo-classical', 'new age piano']
Track id: 2agBDIr9MYDUducQPC1sFU
-- -- -- -- -- -- --
Danceability: 0.315
Energy: 0.22
Key: 9
Lounness: -21.343
Mode: 1
Speechiness: 0.0514
Acousticness: 0.987
Instrumentalness: 0.943
Liveness: 0.0802
Valence: 0.116
Tempo: 145.195
```

```
Duration: 188s
Time signature: 4
```

1.3.2 Downloading Music from Youtube

[ ]:
```python
# search music from youtube, download it and conert to wav
import youtube_dl
def download_youtube_audio(link, output_dir):
    ytdl_format_options = {
    'format': 'bestaudio/best',
    'outtmpl': f'{output_dir}/%(title)s.%(ext)s',
    'noplaylist': True,
    'nocheckcertificate': True,
    'ignoreerrors': False,
    'logtostderr': False,
    'quiet': False,
    'no_warnings': False,
    'default_search': 'auto',
    'source_address': '0.0.0.0',
    'postprocessors': [{
        'key': 'FFmpegExtractAudio',
        'preferredcodec': 'wav'
    }]
   }

    with youtube_dl.YoutubeDL(ytdl_format_options) as ydl:
        info = ydl.extract_info(link, download=True)
```

[ ]:
```python
download_youtube_audio('River Flows in You', f'{DATA_DIR}/youtube_audio')
```

```
[download] Downloading playlist: River Flows in You
[youtube:search] query "River Flows in You": Downloading page 1
[youtube:search] playlist River Flows in You: Downloading 1 videos
[download] Downloading video 1 of 1
[youtube] 7maJOI3QMu0: Downloading webpage
[youtube] Downloading just video 7maJOI3QMu0 because of --no-playlist
[youtube] 7maJOI3QMu0: Downloading player ad2aeb77
[download] Destination: sound_data\youtube_audio\Yiruma, (  ) - River Flows in
You.webm
[download] 100% of 3.84MiB in 00:48
[ffmpeg] Destination: sound_data\youtube_audio\Yiruma, (  ) - River Flows in
You.wav
Deleting original file sound_data\youtube_audio\Yiruma, (  ) - River Flows in
You.webm (pass -k to keep)
[download] Finished downloading playlist: River Flows in You
```

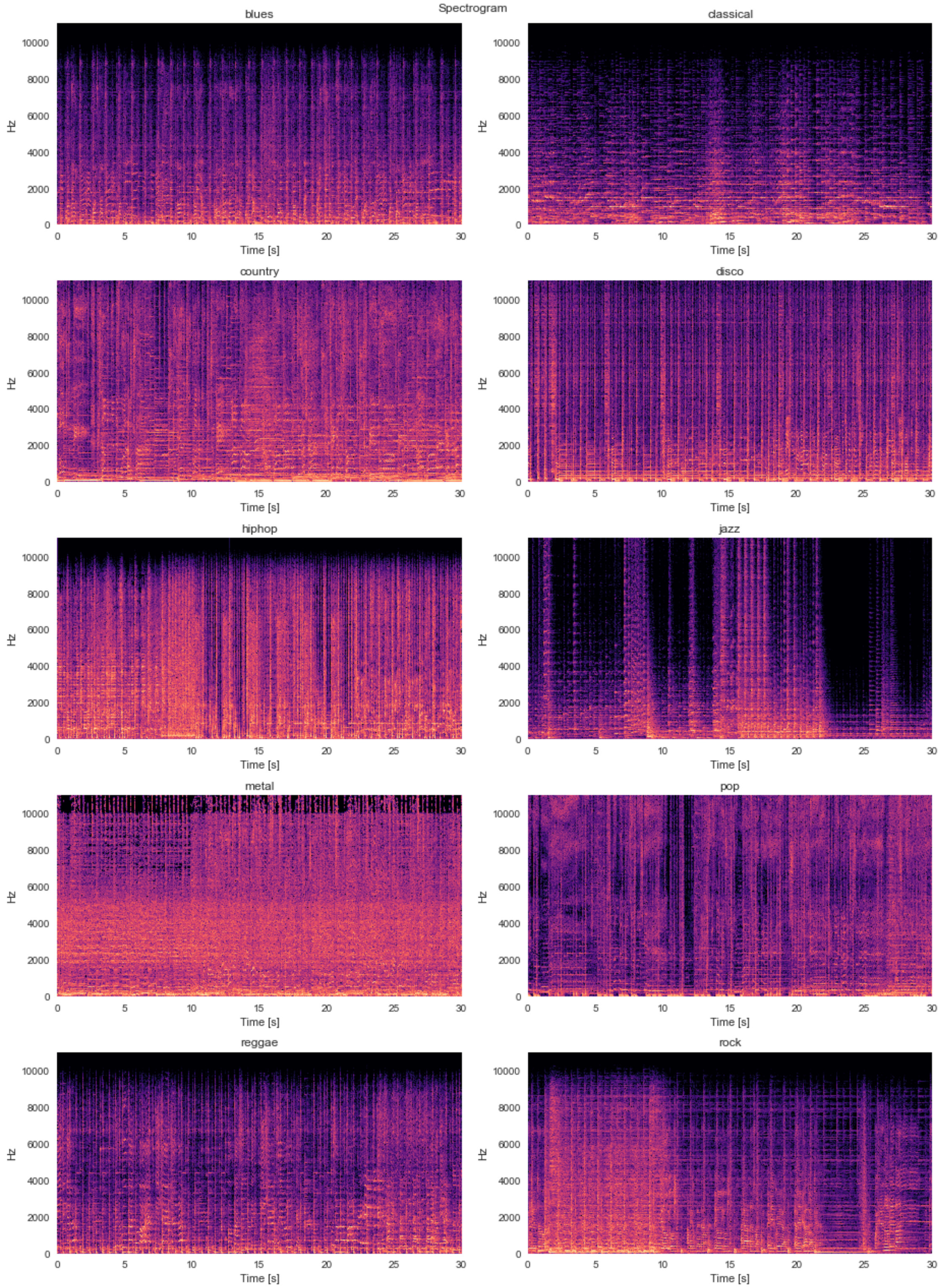
1.4 Visulaization

1.4.1 1. Creating plots: Waveform, Spectrum, Spectrogram

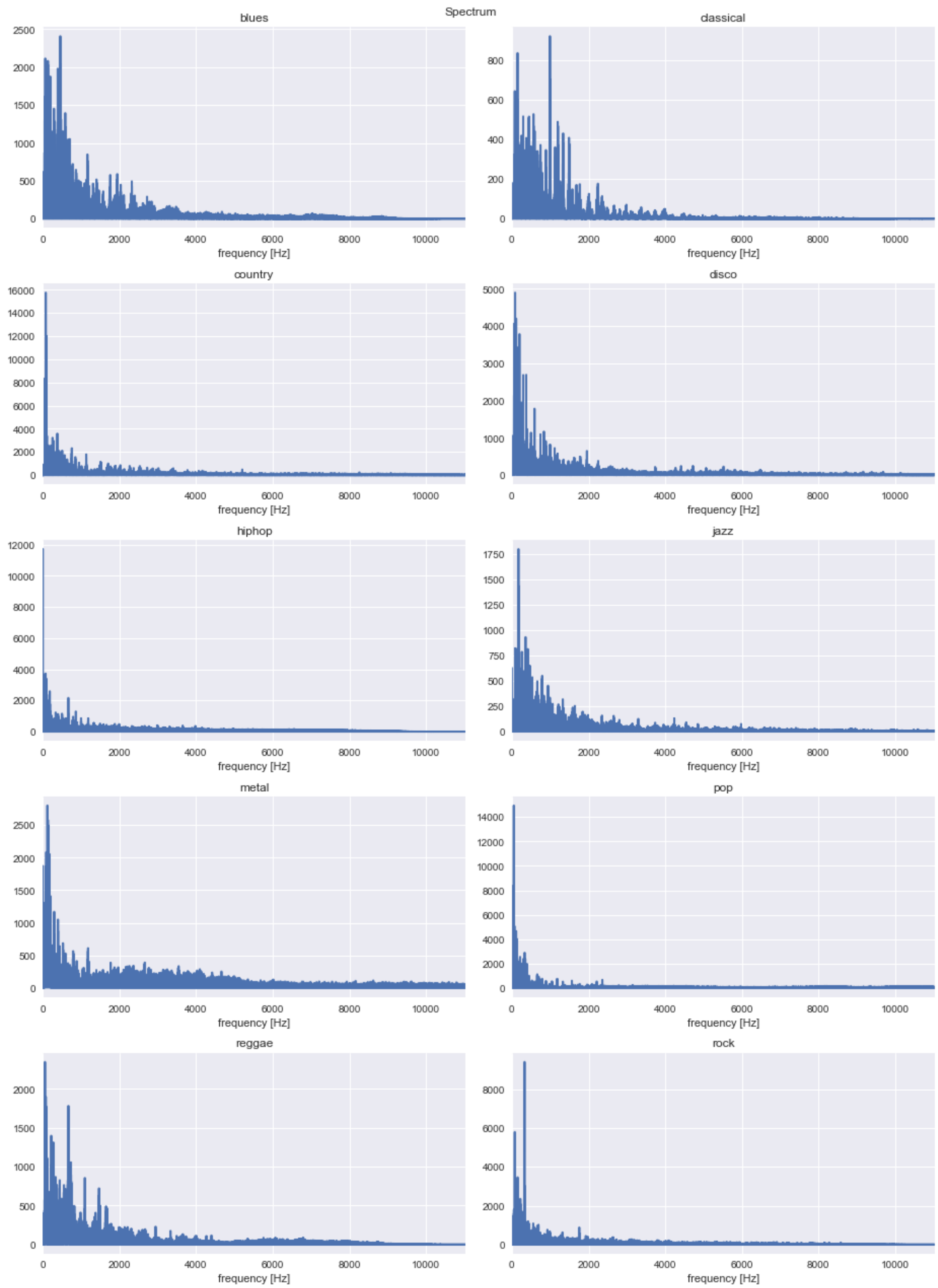
[ ]:
```python
def plot_spectrum(file_name, ax=None):
    if ax is None:
        fig, ax = plt.subplots(1)

    X, sample_rate = librosa.load(file_name, sr=22050, mono=True)
    spectrum = np.fft.fft(X)
    freq = np.fft.fftfreq(len(X), 1.0 / sample_rate)

    ax.plot(freq, abs(spectrum), linewidth=2)
    ax.set(title='Spectrum', xlabel='frequency [Hz]', xlim=(0,11025))

    return ax

def plot_spectrogram(file_name, ax=None):
    if ax is None:
        fig, ax = plt.subplots(1)
    X, sample_rate = librosa.load(file_name, sr=22050, mono=True)
    D = librosa.stft(X)
    S_db = librosa.amplitude_to_db(np.abs(D), ref=np.max)
    librosa.display.specshow(S_db, ax=ax, x_axis='time', y_axis='linear')
    ax.set(title='Spectrogram', xlabel='Time [s]')

def plot_waveform(file_name, ax=None):
    if ax is None:
        fig, ax = plt.subplots(1)
    x, sr = librosa.load(file_name, sr=22050, mono=True)
    librosa.display.waveplot(x, ax=ax)
    ax.set(title='Waveform', xlabel='Time [s]')
```

[ ]:
```python
# Creating a sine wave audio for testing purpose
!"{SOX}" --null -r 22050 sine_a.wav synth 1 sine 1000
```

[ ]:
```python
# Creating spectrogram among all genres
fig, ax = plt.subplots(5, 2, figsize=(13,18))
fig.suptitle('Spectrogram')
fig.tight_layout(h_pad=5)
for i, g in enumerate(GENRES):
    plot_spectrogram(f'{DATA_DIR}/genres/{g}/{g}.00000.wav', ax[i//2, i%2])
    ax[i//2, i%2].set_title(g)
fig.tight_layout()

plt.show()
```


Spectrogram
blues
classical
country
disco
hiphop
jazz
metal
pop
reggae
rock
Hz
Time [s]
0
2000
4000
6000
8000
10000
0
5
10
15
20
25
30

[ ]:
```
# Creating spectrum among all genres
fig, ax = plt.subplots(5, 2, figsize=(13,18))
fig.suptitle('Spectrum')
fig.tight_layout(h_pad=5)
for i, g in enumerate(GENRES):
    plot_spectrum(f'{DATA_DIR}/genres/{g}/{g}.00000.wav', ax[i//2, i%2])
    ax[i//2, i%2].set_title(g)
fig.tight_layout()

plt.show()
```

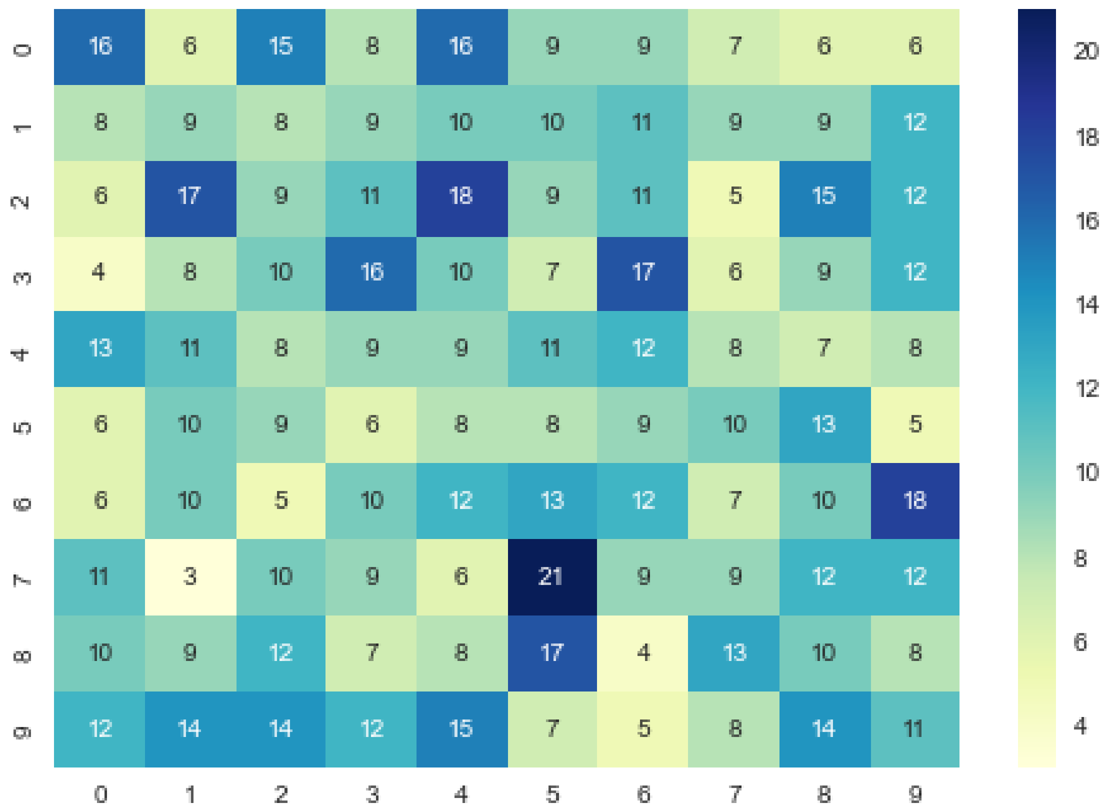

Spectrum
blues
classical
country
disco
hiphop
jazz
metal
pop
reggae
rock
frequency [Hz]

1.4.2 2. Confusion matrix

```
[ ]: def result_visualise(y_true, y_pred, ytick=None):
         plt.clf()
         mat = confusion_matrix(y_true, y_pred)
         sns.heatmap(mat, annot=True, cmap='YlGnBu', yticklabels=ytick,␣
      ↪xticklabels=ytick)
         plt.show()
```

```
[ ]: # Generating data for testing purpose
     y_true = np.random.randint(0, 10, 1000)
     y_pred = np.random.randint(0, 10, 1000)
```

```
[ ]: result_visualise(y_true, y_pred, [str(i) for i in range(10)])
```



1.5 First Attempt

1.5.1 Feature extraction

In the first attempt, I will extract the frequencies with the FFT_TOTAL highest magnitude

```
[ ]: # Extract frequencies with highest amplitude
# Notice that the following code will use all of the CPU resources
if not os.path.exists(f'{DATA_DIR}/original_fft_max{FFT_TOTAL}.csv'):
    def produce_df(path, label, num):
        def convert_fft_csv(df, filename, path, label, num):
            sampFreq, sound = scipy.io.wavfile.read(path)
            signal = sound
            fft_spectrum = np.fft.rfft(signal)
            freq = np.fft.rfftfreq(signal.size, d=1./sampFreq)
            fft_spectrum_abs = np.abs(fft_spectrum)

            x = np.column_stack((np.round(freq, 1), np.round(fft_spectrum_abs)))
            data = np.array(sorted(x, key=lambda x: x[1], reverse=True)[:num])[:
 ↪, 0]

            tmpdf = pd.DataFrame(data.reshape(1,-1), columns=[f"x_{i}" for i in␣
 ↪range(num)]).assign(filename=filename).assign(label=label)
            return df.append(tmpdf, ignore_index=True)

        import pandas as pd, numpy as np, scipy.io.wavfile, os
        df = pd.DataFrame(columns=['filename', 'label']+[f"x_{i}" for i in␣
 ↪range(num)], index=None)
        for file in os.listdir(path):
            exact_path = f"{path}/{file}"
            df = convert_fft_csv(df, file, exact_path, label, num)
        return df

    para = []
    for g in GENRES:
        para.append((f"{DATA_DIR}/genres/{g}", g, FFT_TOTAL))

    pool = mp.Pool(mp.cpu_count())
    data = pool.starmap(produce_df, para)
```

```
[ ]: # store the data as csv format under DATA_DIR
if not os.path.exists(f'{DATA_DIR}/original_fft_max{FFT_TOTAL}.csv'):
    df = pd.DataFrame(columns=['filename', 'label']+[f"x_{i}" for i in␣
 ↪range(FFT_TOTAL)], index=None)
    for i in range(10):
        df = df.append(data[i], ignore_index=True)
        df.to_csv(f'{DATA_DIR}/original_fft_max{FFT_TOTAL}.csv')
    pool.terminate()
```

1.5.2 Loading back data

[ ]:
```
def read_fft_csv(path):
    df = pd.read_csv(path, index_col=0)
    X = df.loc[:, 'x_0':]
    y = df.loc[:, 'label']
    return X, y
X, y = read_fft_csv(f'{DATA_DIR}/original_fft_max{FFT_TOTAL}.csv')
```

[ ]:
```
# replace genre name to number labelling
for idx, label in enumerate(GENRES):
    y = y.replace(label, idx)
```

1.5.3 Classification

Here we start with simple model first

Seperate data into training set and testing set

[ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
```

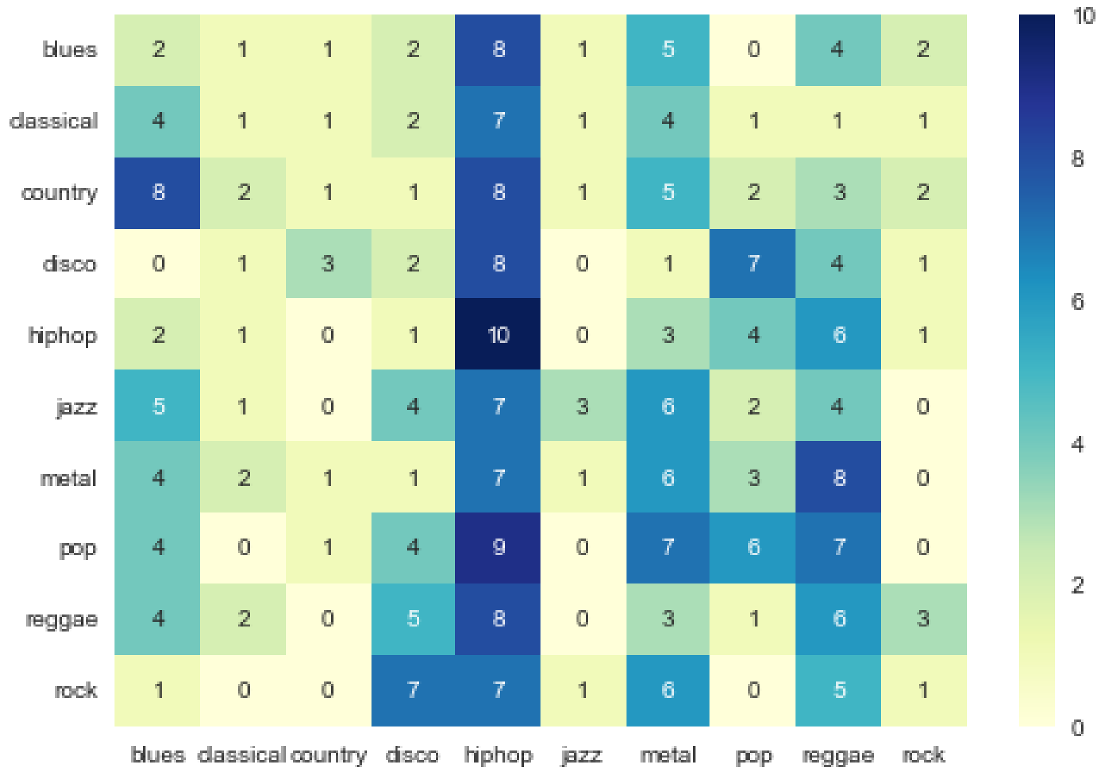
1. Logistic Regression

[ ]:
```
clf = LogisticRegression(max_iter=100000).fit(X_train, y_train)
```

[ ]:
```
print(f"Accuracy on training data: {clf.score(X_train, y_train)}")
print(f"Accuracy on testing data: {clf.score(X_test, y_test)}")
```

```
Accuracy on training data: 1.0
Accuracy on testing data: 0.12666666666666668
```

[ ]:
```
result_visualise(y_test, clf.predict(X_test), GENRES)
```



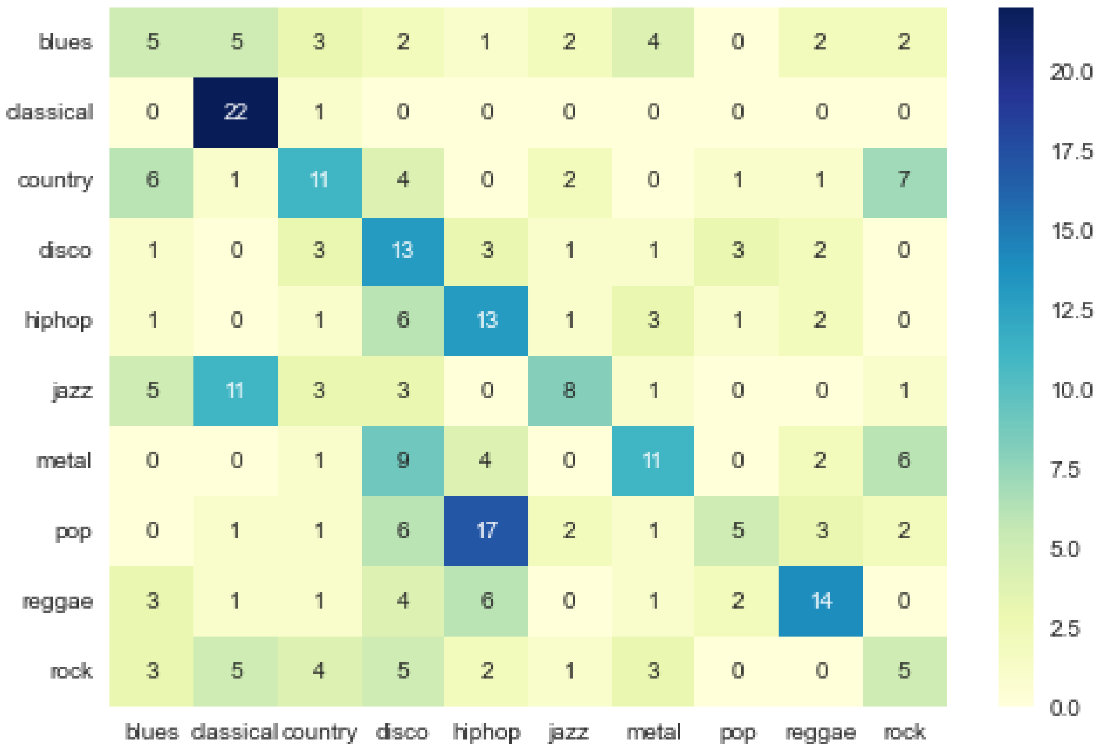
2. Random Forest

```
[ ]: clf = RandomForestClassifier(1000).fit(X_train, y_train)
```

```
[ ]: print(f"Accuracy on training data: {clf.score(X_train, y_train)}")
     print(f"Accuracy on training data: {clf.score(X_test, y_test)}")
```

```
Accuracy on training data: 1.0
Accuracy on training data: 0.3566666666666667
```

```
[ ]: result_visualise(y_test, clf.predict(X_test), GENRES)
```



3. Simple Neural Network

[ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
X_train = tf.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))
X_test = tf.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))
```

[ ]:
```
batch_size = 16
epochs = 50

model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Flatten(input_shape=(2000,1)))
model.add(tf.keras.layers.Dense(2048, activation='relu'))
model.add(tf.keras.layers.Dense(2048, activation='relu'))
model.add(tf.keras.layers.Dense(2048, activation='relu'))
model.add(tf.keras.layers.Dense(2048, activation='relu'))
model.add(tf.keras.layers.Dense(2048, activation='relu'))
model.add(tf.keras.layers.Dense(512, activation='relu'))
model.add(tf.keras.layers.Dense(256, activation='relu'))
model.add(tf.keras.layers.Dense(128, activation='relu'))
model.add(tf.keras.layers.Dense(10, activation=tf.nn.softmax))

optimiser = tf.keras.optimizers.Adam()
```

```
model.compile(optimizer=optimiser, loss='sparse_categorical_crossentropy',␣
↪metrics=['accuracy'])
```

[ ]:
```
model.fit(X_train, y_train, batch_size=batch_size, epochs=epochs)
```

```
Epoch 1/50
44/44 [==============================] - 3s 12ms/step - loss: 85.0187 -
accuracy: 0.1229
Epoch 2/50
44/44 [==============================] - 0s 9ms/step - loss: 2.3964 - accuracy:
0.1100
Epoch 3/50
44/44 [==============================] - 0s 8ms/step - loss: 2.3116 - accuracy:
0.1157
Epoch 4/50
44/44 [==============================] - 0s 8ms/step - loss: 2.3278 - accuracy:
0.1543
Epoch 5/50
44/44 [==============================] - 0s 8ms/step - loss: 2.3181 - accuracy:
0.1386
Epoch 6/50
44/44 [==============================] - 0s 8ms/step - loss: 2.1982 - accuracy:
0.1800
Epoch 7/50
44/44 [==============================] - 0s 8ms/step - loss: 2.2149 - accuracy:
0.1771
Epoch 8/50
44/44 [==============================] - 0s 8ms/step - loss: 2.1896 - accuracy:
0.1814
Epoch 9/50
44/44 [==============================] - 0s 8ms/step - loss: 2.2085 - accuracy:
0.1986
Epoch 10/50
44/44 [==============================] - 0s 8ms/step - loss: 2.4389 - accuracy:
0.2043
Epoch 11/50
44/44 [==============================] - 0s 8ms/step - loss: 2.9624 - accuracy:
0.1429
Epoch 12/50
44/44 [==============================] - 0s 8ms/step - loss: 2.4358 - accuracy:
0.1586
Epoch 13/50
44/44 [==============================] - 0s 7ms/step - loss: 2.2372 - accuracy:
0.1771
Epoch 14/50
44/44 [==============================] - 0s 8ms/step - loss: 2.2561 - accuracy:
0.1743
Epoch 15/50
```

```
44/44 [==============================] - 0s 8ms/step - loss: 2.1798 - accuracy:
0.1800
Epoch 16/50
44/44 [==============================] - 0s 8ms/step - loss: 2.3921 - accuracy:
0.1786
Epoch 17/50
44/44 [==============================] - 0s 8ms/step - loss: 2.3685 - accuracy:
0.1429
Epoch 18/50
44/44 [==============================] - 0s 8ms/step - loss: 2.2421 - accuracy:
0.1657
Epoch 19/50
44/44 [==============================] - 0s 8ms/step - loss: 2.2773 - accuracy:
0.1586
Epoch 20/50
44/44 [==============================] - 0s 8ms/step - loss: 2.4304 - accuracy:
0.1400
Epoch 21/50
44/44 [==============================] - 0s 8ms/step - loss: 2.2513 - accuracy:
0.1671
Epoch 22/50
44/44 [==============================] - 0s 8ms/step - loss: 2.2821 - accuracy:
0.1586
Epoch 23/50
44/44 [==============================] - 0s 8ms/step - loss: 2.2301 - accuracy:
0.1743
Epoch 24/50
44/44 [==============================] - 0s 8ms/step - loss: 2.2462 - accuracy:
0.1543
Epoch 25/50
44/44 [==============================] - 0s 8ms/step - loss: 2.3552 - accuracy:
0.1543
Epoch 26/50
44/44 [==============================] - 0s 7ms/step - loss: 2.2964 - accuracy:
0.1671
Epoch 27/50
44/44 [==============================] - 0s 8ms/step - loss: 2.2733 - accuracy:
0.1643
Epoch 28/50
44/44 [==============================] - 0s 8ms/step - loss: 2.2321 - accuracy:
0.1814
Epoch 29/50
44/44 [==============================] - 0s 9ms/step - loss: 2.1218 - accuracy:
0.1971
Epoch 30/50
44/44 [==============================] - 0s 8ms/step - loss: 2.0189 - accuracy:
0.2329
Epoch 31/50
```

```
44/44 [==============================] - 0s 8ms/step - loss: 2.0393 - accuracy:
0.2114
Epoch 32/50
44/44 [==============================] - 0s 9ms/step - loss: 2.0114 - accuracy:
0.2429
Epoch 33/50
44/44 [==============================] - 0s 9ms/step - loss: 2.0307 - accuracy:
0.2014
Epoch 34/50
44/44 [==============================] - 0s 9ms/step - loss: 2.0061 - accuracy:
0.2214
Epoch 35/50
44/44 [==============================] - 0s 9ms/step - loss: 2.1134 - accuracy:
0.2257
Epoch 36/50
44/44 [==============================] - 0s 8ms/step - loss: 2.4493 - accuracy:
0.1243
Epoch 37/50
44/44 [==============================] - 0s 8ms/step - loss: 2.2060 - accuracy:
0.1786
Epoch 38/50
44/44 [==============================] - 0s 8ms/step - loss: 2.0797 - accuracy:
0.1943
Epoch 39/50
44/44 [==============================] - 0s 8ms/step - loss: 2.0895 - accuracy:
0.2043
Epoch 40/50
44/44 [==============================] - 0s 8ms/step - loss: 1.9741 - accuracy:
0.2286
Epoch 41/50
44/44 [==============================] - 0s 8ms/step - loss: 1.9692 - accuracy:
0.2300
Epoch 42/50
44/44 [==============================] - 0s 9ms/step - loss: 1.8900 - accuracy:
0.2386
Epoch 43/50
44/44 [==============================] - 0s 7ms/step - loss: 1.9786 - accuracy:
0.2457
Epoch 44/50
44/44 [==============================] - 0s 8ms/step - loss: 1.9895 - accuracy:
0.2114
Epoch 45/50
44/44 [==============================] - 0s 8ms/step - loss: 1.7746 - accuracy:
0.2757
Epoch 46/50
44/44 [==============================] - 0s 8ms/step - loss: 1.8169 - accuracy:
0.2186
Epoch 47/50
```

```
44/44 [==============================] - 0s 7ms/step - loss: 1.8616 - accuracy:
0.2543
Epoch 48/50
44/44 [==============================] - 0s 8ms/step - loss: 2.0220 - accuracy:
0.2343
Epoch 49/50
44/44 [==============================] - 0s 7ms/step - loss: 1.7299 - accuracy:
0.2714
Epoch 50/50
44/44 [==============================] - 0s 8ms/step - loss: 1.6624 - accuracy:
0.2957
```

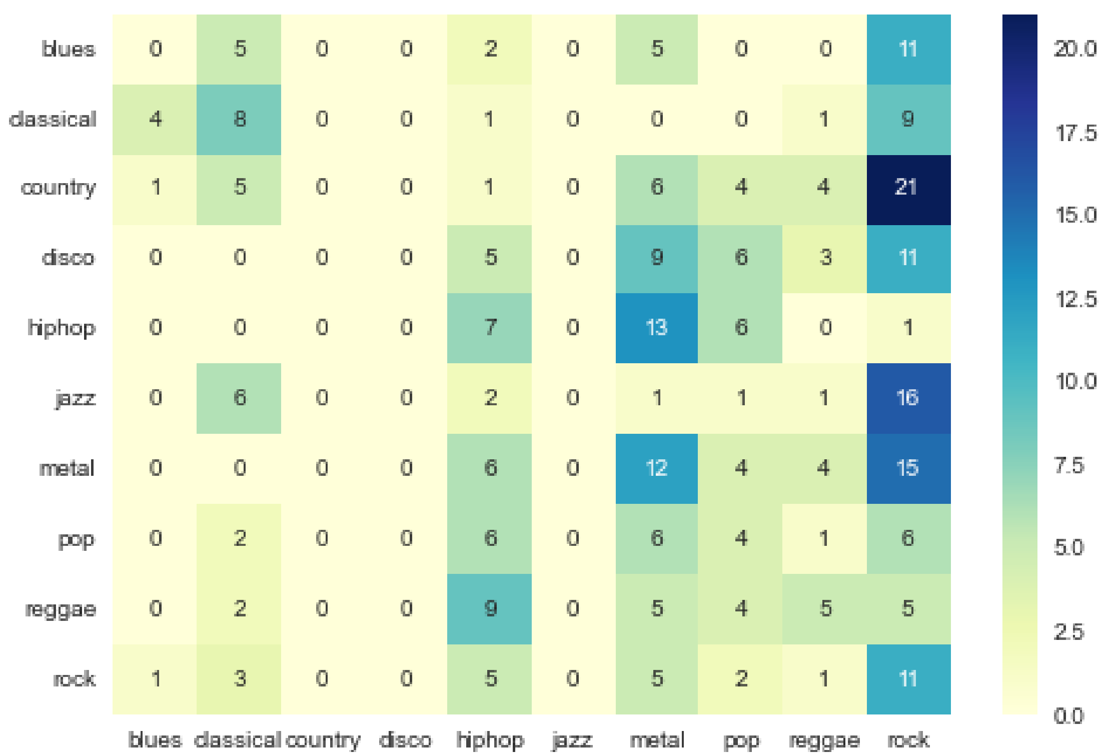
[ ]: `<tensorflow.python.keras.callbacks.History at 0x1bd1c4aa6d0>`

[ ]:
```
model.evaluate(X_test, y_test)
```

```
10/10 [==============================] - 0s 13ms/step - loss: 2.4748 - accuracy:
0.1567
```

[ ]: `[2.47478985786438, 0.15666666626930237]`

[ ]:
```
result_visualise(y_test, np.argmax(model.predict(X_test), axis=-1), GENRES)
```



4. CNN

[ ]:
```
batch_size = 16
epochs = 50

model2 = tf.keras.models.Sequential()
model2.add(tf.keras.layers.Conv1D(64, 2, activation='relu',␣
 ↪input_shape=(2000,1))) # edited
model2.add(tf.keras.layers.MaxPooling1D(pool_size = 2))
model2.add(tf.keras.layers.Conv1D(128, 2, activation = 'relu'))
model2.add(tf.keras.layers.MaxPooling1D(pool_size = 2))
model2.add(tf.keras.layers.Conv1D(256, 2, activation = 'relu'))
model2.add(tf.keras.layers.MaxPooling1D(pool_size = 4))
model2.add(tf.keras.layers.Conv1D(512, 2, activation = 'relu'))
model2.add(tf.keras.layers.MaxPooling1D(pool_size = 4))
model2.add(tf.keras.layers.Flatten())
model2.add(tf.keras.layers.Dense(2048, activation = 'relu'))
model2.add(tf.keras.layers.Dense(1024, activation = 'relu'))
model2.add(tf.keras.layers.Dense(512, activation = 'relu'))
model2.add(tf.keras.layers.Dense(10, activation='softmax'))

optimiser = tf.keras.optimizers.Adam()
model2.compile(optimizer=optimiser, loss='sparse_categorical_crossentropy',␣
 ↪metrics=['accuracy'])
#model2.summary()
```

[ ]:
```
model2.fit(X_train, y_train,epochs=epochs,batch_size=batch_size)
```

```
Epoch 1/50
44/44 [==============================] - 6s 20ms/step - loss: 59.4738 -
accuracy: 0.1257
Epoch 2/50
44/44 [==============================] - 1s 17ms/step - loss: 2.2588 - accuracy:
0.1657
Epoch 3/50
44/44 [==============================] - 1s 17ms/step - loss: 2.2316 - accuracy:
0.1800
Epoch 4/50
44/44 [==============================] - 1s 17ms/step - loss: 2.2064 - accuracy:
0.1771
Epoch 5/50
44/44 [==============================] - 1s 18ms/step - loss: 2.1126 - accuracy:
0.2029
Epoch 6/50
44/44 [==============================] - 1s 17ms/step - loss: 2.0963 - accuracy:
0.2514
Epoch 7/50
44/44 [==============================] - 1s 17ms/step - loss: 2.0326 - accuracy:
0.2943
Epoch 8/50
```

```
44/44 [==============================] - 1s 17ms/step - loss: 1.8603 - accuracy:
0.3571
Epoch 9/50
44/44 [==============================] - 1s 17ms/step - loss: 1.9290 - accuracy:
0.3357
Epoch 10/50
44/44 [==============================] - 1s 17ms/step - loss: 1.7815 - accuracy:
0.4014
Epoch 11/50
44/44 [==============================] - 1s 17ms/step - loss: 1.4413 - accuracy:
0.4914
Epoch 12/50
44/44 [==============================] - 1s 17ms/step - loss: 1.2382 - accuracy:
0.5657
Epoch 13/50
44/44 [==============================] - 1s 17ms/step - loss: 1.0116 - accuracy:
0.6429
Epoch 14/50
44/44 [==============================] - 1s 17ms/step - loss: 1.3025 - accuracy:
0.6071
Epoch 15/50
44/44 [==============================] - 1s 17ms/step - loss: 1.3466 - accuracy:
0.5786
Epoch 16/50
44/44 [==============================] - 1s 17ms/step - loss: 0.9465 - accuracy:
0.7214
Epoch 17/50
44/44 [==============================] - 1s 17ms/step - loss: 0.8134 - accuracy:
0.7486
Epoch 18/50
44/44 [==============================] - 1s 17ms/step - loss: 0.7286 - accuracy:
0.7714
Epoch 19/50
44/44 [==============================] - 1s 17ms/step - loss: 0.8286 - accuracy:
0.7414
Epoch 20/50
44/44 [==============================] - 1s 18ms/step - loss: 0.8756 - accuracy:
0.7357
Epoch 21/50
44/44 [==============================] - 1s 17ms/step - loss: 0.7528 - accuracy:
0.7771
Epoch 22/50
44/44 [==============================] - 1s 17ms/step - loss: 0.7557 - accuracy:
0.7771
Epoch 23/50
44/44 [==============================] - 1s 18ms/step - loss: 0.7781 - accuracy:
0.7943
Epoch 24/50
```

```
44/44 [==============================] - 1s 18ms/step - loss: 0.5905 - accuracy:
0.8271
Epoch 25/50
44/44 [==============================] - 1s 17ms/step - loss: 2.2815 - accuracy:
0.4271
Epoch 26/50
44/44 [==============================] - 1s 19ms/step - loss: 1.3503 - accuracy:
0.5471
Epoch 27/50
44/44 [==============================] - 1s 19ms/step - loss: 0.7950 - accuracy:
0.7629
Epoch 28/50
44/44 [==============================] - 1s 18ms/step - loss: 0.9965 - accuracy:
0.7429
Epoch 29/50
44/44 [==============================] - 1s 20ms/step - loss: 0.6377 - accuracy:
0.8071
Epoch 30/50
44/44 [==============================] - 1s 19ms/step - loss: 0.4622 - accuracy:
0.8571
Epoch 31/50
44/44 [==============================] - 1s 19ms/step - loss: 0.5439 - accuracy:
0.8314
Epoch 32/50
44/44 [==============================] - 1s 19ms/step - loss: 0.7462 - accuracy:
0.8100
Epoch 33/50
44/44 [==============================] - 1s 18ms/step - loss: 0.5063 - accuracy:
0.8757
Epoch 34/50
44/44 [==============================] - 1s 18ms/step - loss: 0.5031 - accuracy:
0.8871
Epoch 35/50
44/44 [==============================] - 1s 17ms/step - loss: 0.4368 - accuracy:
0.8829
Epoch 36/50
44/44 [==============================] - 1s 17ms/step - loss: 0.4032 - accuracy:
0.8914
Epoch 37/50
44/44 [==============================] - 1s 18ms/step - loss: 0.6129 - accuracy:
0.8486
Epoch 38/50
44/44 [==============================] - 1s 18ms/step - loss: 0.3673 - accuracy:
0.9000
Epoch 39/50
44/44 [==============================] - 1s 17ms/step - loss: 0.1873 - accuracy:
0.9443
Epoch 40/50
```

```
44/44 [==============================] - 1s 19ms/step - loss: 0.1436 - accuracy:
0.9557
Epoch 41/50
44/44 [==============================] - 1s 17ms/step - loss: 0.1131 - accuracy:
0.9643
Epoch 42/50
44/44 [==============================] - 1s 17ms/step - loss: 0.1597 - accuracy:
0.9700
Epoch 43/50
44/44 [==============================] - 1s 18ms/step - loss: 0.1264 - accuracy:
0.9657
Epoch 44/50
44/44 [==============================] - 1s 20ms/step - loss: 0.3253 - accuracy:
0.9086
Epoch 45/50
44/44 [==============================] - 1s 19ms/step - loss: 0.1309 - accuracy:
0.9714
Epoch 46/50
44/44 [==============================] - 1s 17ms/step - loss: 0.0600 - accuracy:
0.9829
Epoch 47/50
44/44 [==============================] - 1s 18ms/step - loss: 0.0272 - accuracy:
0.9914
Epoch 48/50
44/44 [==============================] - 1s 18ms/step - loss: 0.0618 - accuracy:
0.9771
Epoch 49/50
44/44 [==============================] - 1s 17ms/step - loss: 0.0393 - accuracy:
0.9900
Epoch 50/50
44/44 [==============================] - 1s 17ms/step - loss: 0.0103 - accuracy:
1.0000
```

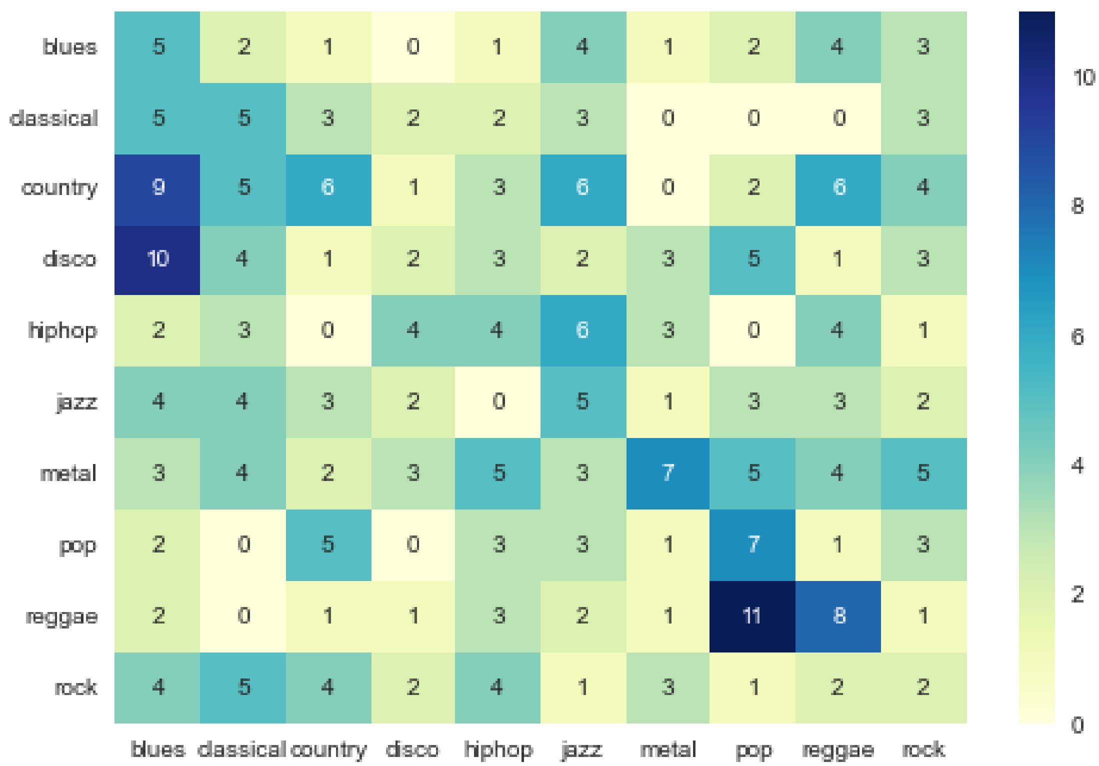
[ ]:
```
<tensorflow.python.keras.callbacks.History at 0x1bd28c26b50>
```

[ ]:
```
model2.evaluate(X_test, y_test)
```

```
10/10 [==============================] - 0s 7ms/step - loss: 12.9256 - accuracy:
0.1700
```

[ ]:
```
[12.925633430480957, 0.17000000178813934]
```

[ ]:
```
result_visualise(y_test, np.argmax(model2.predict(X_test), axis=-1), GENRES)
```



1.6 Second Attempt

1.6.1 Feature extraction

In the second attempt, instead of using frequencies as the data for model training, the following parameters will be extracted:

1. chroma stft mean
2. chroma stft variance
3. rms mean
4. rms variance
5. spectral centroid mean
6. spectral centroid variance
7. spectral bandwidth mean
8. spectral bandwidth variance
9. spectral rolloff mean
10. spectral rolloff variance
11. zero crossing rate mean
12. zero crossing rate variance
13. tempo
14. Mel-frequency cepstral coefficients (20)

```
[ ]: # Notice that the following code will use all of the CPU resource
if not os.path.exists(f'{DATA_DIR}/original_audio_features.csv'):
    def produce_df_2(path, label):
        def fetch_features(df, filename, path, label):
            y, sr = librosa.load(path)
            mfcc = librosa.feature.mfcc(y, sr)
            data = [[filename,
                    label,
                    np.mean(librosa.feature.chroma_stft(y, sr)),
                    np.var(librosa.feature.chroma_stft(y, sr)),
                    np.mean(librosa.feature.rms(y)),
                    np.var(librosa.feature.rms(y)),
                    np.mean(librosa.feature.spectral_centroid(y, sr)),
                    np.var(librosa.feature.spectral_centroid(y, sr)),
                    np.mean(librosa.feature.spectral_bandwidth(y, sr)),
                    np.var(librosa.feature.spectral_bandwidth(y, sr)),
                    np.mean(librosa.feature.spectral_rolloff(y, sr)),
                    np.var(librosa.feature.spectral_rolloff(y, sr)),
                    np.mean(librosa.feature.zero_crossing_rate(y)),
                    np.var(librosa.feature.zero_crossing_rate(y)),
                    librosa.beat.tempo(y)[0]] + [np.mean(mfcc[i]) for i in␣
 ↪range(20)]]

            tmpdf = pd.DataFrame(data, columns=df.columns)
            return df.append(tmpdf, ignore_index=True)

        import numpy as np, pandas as pd, librosa, os
        df = pd.DataFrame(columns=['filename',
                                   'label',
                                   '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',
                                   'tempo'] + [f"mfcc{i}" for i in range(1, 21)],␣
 ↪index=None)
        for file in os.listdir(path):
            exact_path = f"{path}/{file}"
            df = fetch_features(df, file, exact_path, label)
        return df
```

```
    para = []
    for g in GENRES:
        para.append((f"{DATA_DIR}/genres/{g}", g))

    pool = mp.Pool(mp.cpu_count()//3)
    data = pool.starmap(produce_df_2, para)
```

[ ]:
```
# store the data as csv format under DATA_DIR
if not os.path.exists(f'{DATA_DIR}/original_audio_features.csv'):
    df = pd.DataFrame(columns=data[0].columns, index=None)
    for i in range(10):
        df = df.append(data[i], ignore_index=True)
    df.to_csv(f'{DATA_DIR}/original_audio_features.csv')
```

1.6.2 Loading back data

[ ]:
```
def read_fft_csv(path):
    df = pd.read_csv(path, index_col=0)
    X = df.loc[:, 'chroma_stft_mean':]
    y = df.loc[:, 'label']
    return X, y
X, y = read_fft_csv(f'{DATA_DIR}/original_audio_features.csv')
```

[ ]:
```
# replace genre name to number labelling
for idx, label in enumerate(GENRES):
    y = y.replace(label, idx)
```

1.6.3 Classification

Seperate data into training set and testign set

[ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
```

1. Logistic Regression

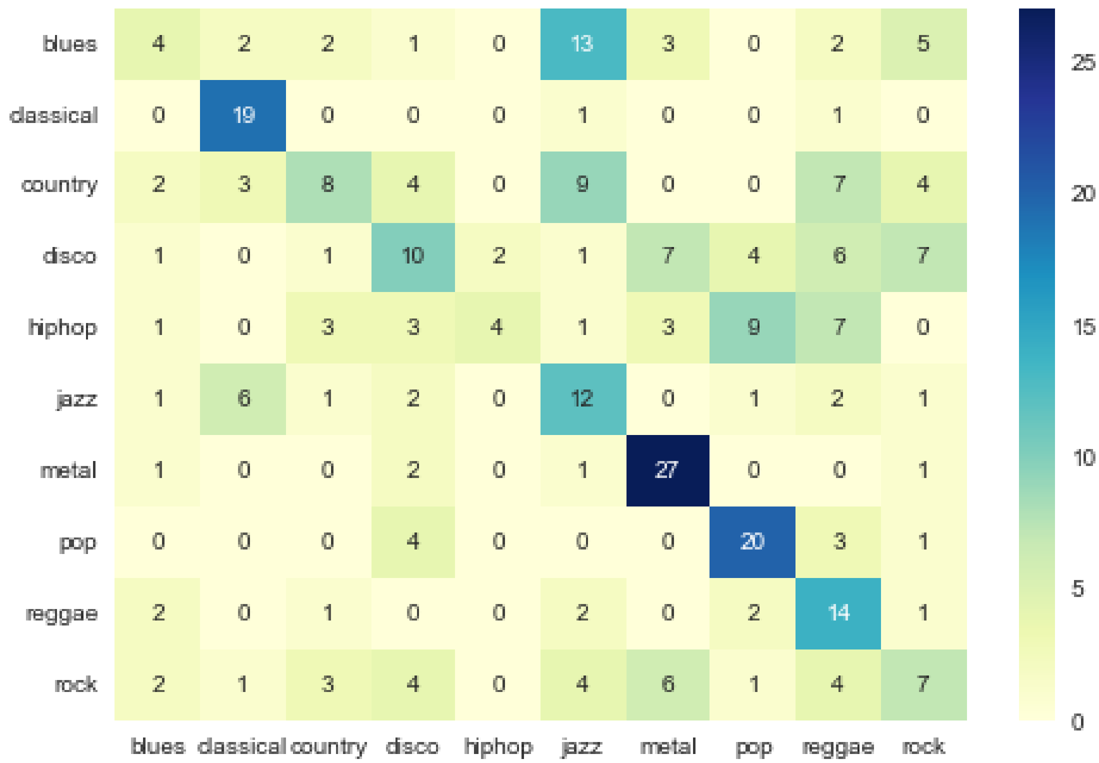
[ ]:
```
clf = LogisticRegression(max_iter=1000000).fit(X_train, y_train)
```

```
C:\ProgramData\Anaconda3\envs\tf-gpu\lib\site-
packages\sklearn\linear_model\_logistic.py:814: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of f AND g EVALUATIONS EXCEEDS LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

```
[ ]: print(f"Accuracy on training data: {clf.score(X_train, y_train)}")
     print(f"Accuracy on testing data: {clf.score(X_test, y_test)}")
```

```
Accuracy on training data: 0.4757142857142857
Accuracy on testing data: 0.44666666666666666
```

```
[ ]: result_visualise(y_test, clf.predict(X_test), GENRES)
```



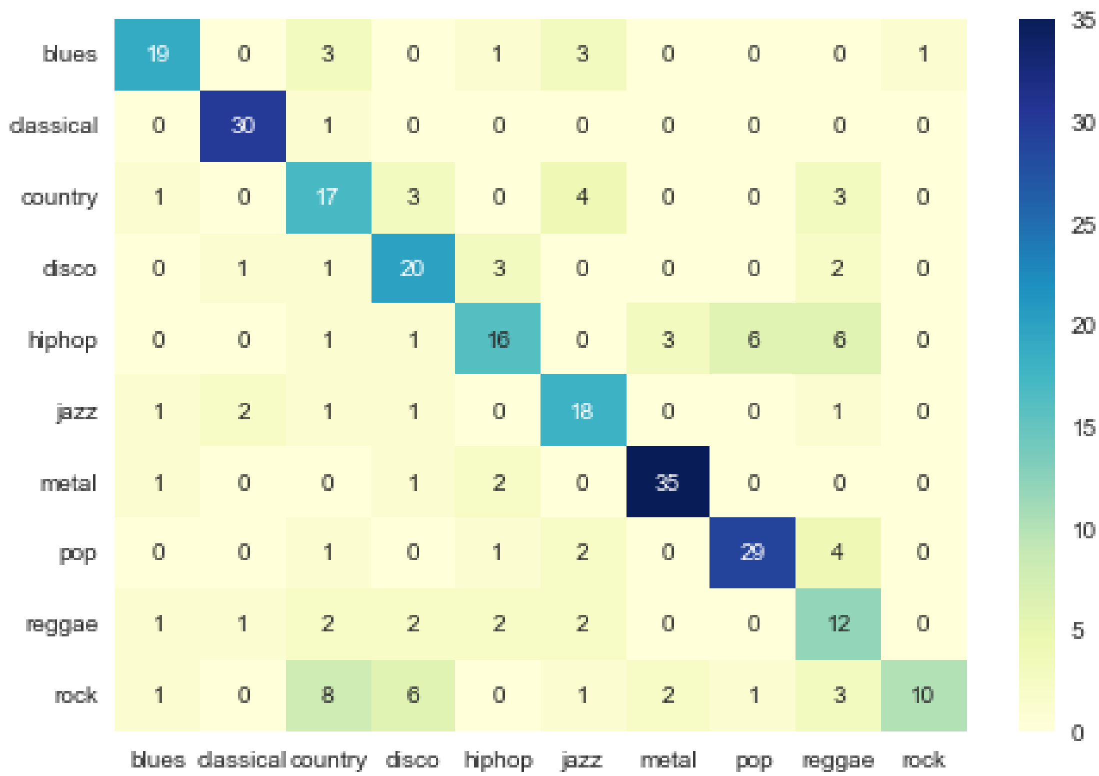
2. Random Forest

```
[ ]: clf = RandomForestClassifier(1000).fit(X_train, y_train)
```

```
[ ]: print(f"Accuracy on training data: {clf.score(X_train, y_train)}")
     print(f"Accuracy on training data: {clf.score(X_test, y_test)}")
```

```
Accuracy on training data: 0.9985714285714286
Accuracy on training data: 0.6866666666666666
```

```
[ ]: result_visualise(y_test, clf.predict(X_test), GENRES)
```



Scaling data to have unit norm

[ ]:
```
scaler = preprocessing.StandardScaler().fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

[ ]:
```
X_train = tf.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))
X_test = tf.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))
```

3. Simple Neural Network

[ ]:
```
batch_size = 16
epochs = 50

model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Flatten(input_shape=(33,1)))

model.add(tf.keras.layers.Dense(256, activation='relu'))
model.add(tf.keras.layers.Dense(128, activation='relu'))
model.add(tf.keras.layers.Dense(64, activation='relu'))
model.add(tf.keras.layers.Dense(10, activation=tf.nn.softmax))

optimiser = tf.keras.optimizers.Adam()
```

```
model.compile(optimizer=optimiser, loss='sparse_categorical_crossentropy',␣
↪metrics=['accuracy'])
```

[ ]:
```
model.fit(X_train, y_train, batch_size=batch_size, epochs=epochs)
```

```
Epoch 1/50
44/44 [==============================] - 1s 4ms/step - loss: 1.8718 - accuracy:
0.3400
Epoch 2/50
44/44 [==============================] - 0s 4ms/step - loss: 1.2076 - accuracy:
0.5929
Epoch 3/50
44/44 [==============================] - 0s 4ms/step - loss: 0.9453 - accuracy:
0.6657
Epoch 4/50
44/44 [==============================] - 0s 4ms/step - loss: 0.7936 - accuracy:
0.7314
Epoch 5/50
44/44 [==============================] - 0s 4ms/step - loss: 0.6777 - accuracy:
0.7714
Epoch 6/50
44/44 [==============================] - 0s 4ms/step - loss: 0.6078 - accuracy:
0.7843
Epoch 7/50
44/44 [==============================] - 0s 4ms/step - loss: 0.5022 - accuracy:
0.8314
Epoch 8/50
44/44 [==============================] - 0s 4ms/step - loss: 0.4533 - accuracy:
0.8443
Epoch 9/50
44/44 [==============================] - 0s 4ms/step - loss: 0.3924 - accuracy:
0.8843
Epoch 10/50
44/44 [==============================] - 0s 3ms/step - loss: 0.3488 - accuracy:
0.8929
Epoch 11/50
44/44 [==============================] - 0s 4ms/step - loss: 0.2923 - accuracy:
0.9129
Epoch 12/50
44/44 [==============================] - 0s 3ms/step - loss: 0.2763 - accuracy:
0.9143
Epoch 13/50
44/44 [==============================] - 0s 3ms/step - loss: 0.2600 - accuracy:
0.9086
Epoch 14/50
44/44 [==============================] - 0s 3ms/step - loss: 0.1921 - accuracy:
0.9471
Epoch 15/50
```

```
44/44 [==============================] - 0s 4ms/step - loss: 0.1661 - accuracy:
0.9529
Epoch 16/50
44/44 [==============================] - 0s 5ms/step - loss: 0.1497 - accuracy:
0.9671
Epoch 17/50
44/44 [==============================] - 0s 3ms/step - loss: 0.1124 - accuracy:
0.9829
Epoch 18/50
44/44 [==============================] - 0s 3ms/step - loss: 0.0991 - accuracy:
0.9771
Epoch 19/50
44/44 [==============================] - 0s 3ms/step - loss: 0.0711 - accuracy:
0.9886
Epoch 20/50
44/44 [==============================] - 0s 4ms/step - loss: 0.0639 - accuracy:
0.9943
Epoch 21/50
44/44 [==============================] - 0s 4ms/step - loss: 0.0556 - accuracy:
0.9957
Epoch 22/50
44/44 [==============================] - 0s 3ms/step - loss: 0.0500 - accuracy:
0.9914
Epoch 23/50
44/44 [==============================] - 0s 4ms/step - loss: 0.0513 - accuracy:
0.9900
Epoch 24/50
44/44 [==============================] - 0s 4ms/step - loss: 0.0349 - accuracy:
0.9971
Epoch 25/50
44/44 [==============================] - 0s 4ms/step - loss: 0.0429 - accuracy:
0.9914
Epoch 26/50
44/44 [==============================] - 0s 3ms/step - loss: 0.0411 - accuracy:
0.9914
Epoch 27/50
44/44 [==============================] - 0s 4ms/step - loss: 0.0252 - accuracy:
0.9971
Epoch 28/50
44/44 [==============================] - 0s 3ms/step - loss: 0.0186 - accuracy:
0.9986
Epoch 29/50
44/44 [==============================] - 0s 4ms/step - loss: 0.0184 - accuracy:
0.9971
Epoch 30/50
44/44 [==============================] - 0s 4ms/step - loss: 0.0312 - accuracy:
0.9914
Epoch 31/50
```

```
44/44 [==============================] - 0s 4ms/step - loss: 0.0568 - accuracy:
0.9857
Epoch 32/50
44/44 [==============================] - 0s 4ms/step - loss: 0.0241 - accuracy:
0.9971
Epoch 33/50
44/44 [==============================] - 0s 4ms/step - loss: 0.0170 - accuracy:
0.9986
Epoch 34/50
44/44 [==============================] - 0s 3ms/step - loss: 0.0151 - accuracy:
0.9971
Epoch 35/50
44/44 [==============================] - 0s 3ms/step - loss: 0.0197 - accuracy:
0.9971
Epoch 36/50
44/44 [==============================] - 0s 3ms/step - loss: 0.0136 - accuracy:
0.9971
Epoch 37/50
44/44 [==============================] - 0s 4ms/step - loss: 0.0111 - accuracy:
0.9986
Epoch 38/50
44/44 [==============================] - 0s 4ms/step - loss: 0.0100 - accuracy:
0.9986
Epoch 39/50
44/44 [==============================] - 0s 2ms/step - loss: 0.0168 - accuracy:
0.9971
Epoch 40/50
44/44 [==============================] - 0s 3ms/step - loss: 0.0085 - accuracy:
0.9986
Epoch 41/50
44/44 [==============================] - 0s 4ms/step - loss: 0.0111 - accuracy:
0.9971
Epoch 42/50
44/44 [==============================] - 0s 3ms/step - loss: 0.0134 - accuracy:
0.9971
Epoch 43/50
44/44 [==============================] - 0s 5ms/step - loss: 0.0102 - accuracy:
0.9957
Epoch 44/50
44/44 [==============================] - 0s 4ms/step - loss: 0.0084 - accuracy:
0.9986
Epoch 45/50
44/44 [==============================] - 0s 4ms/step - loss: 0.0089 - accuracy:
0.9971
Epoch 46/50
44/44 [==============================] - 0s 4ms/step - loss: 0.0068 - accuracy:
0.9986
Epoch 47/50
```

```
44/44 [==============================] - 0s 3ms/step - loss: 0.0090 - accuracy:
0.9971
Epoch 48/50
44/44 [==============================] - 0s 3ms/step - loss: 0.0084 - accuracy:
0.9986
Epoch 49/50
44/44 [==============================] - 0s 3ms/step - loss: 0.0063 - accuracy:
0.9986
Epoch 50/50
44/44 [==============================] - 0s 3ms/step - loss: 0.0125 - accuracy:
0.9957
```

```
[ ]: <tensorflow.python.keras.callbacks.History at 0x1bd4a0b6af0>
```

```
[ ]: model.evaluate(X_test, y_test)
```

```
10/10 [==============================] - 0s 4ms/step - loss: 1.3397 - accuracy:
0.7267
```

```
[ ]: [1.3396912813186646, 0.7266666889190674]
```

1.7 Third Attempt

In the third attempt, I perform FFT on the audio file and extract amplitude among frequency 0 - 11025 Hz

1.7.1 Feature extraction

```
[ ]: if not os.path.exists(f'{DATA_DIR}/original_fft_amp.csv'):
         def produce_df_fft(path, label):
             def get_fft(df, filename, path, label):
                 y, signal = scipy.io.wavfile.read(path)
                 fft_spectrum = np.fft.rfft(signal)
                 freq = np.fft.rfftfreq(signal.size, d=1./y)
                 fft_spectrum_abs = np.abs(fft_spectrum)

                 data = np.column_stack((freq, np.round(fft_spectrum_abs)))
                 tmpdf = pd.DataFrame(data, columns=['freq', 'amp'])
                 tmpdf.loc[:, 'freq'] = np.round(tmpdf['freq'])

                 return df.append(
                     pd.DataFrame(
                         np.array(tmpdf.groupby('freq').max('amp')).reshape(1, -1),
                         columns=[str(i) for i in range(0, 11025+1)],
                     ).assign(filename=filename).assign(label=label),
                     ignore_index=True
                 )

             import pandas as pd, numpy as np, scipy.io.wavfile, os
```

```
        df = pd.DataFrame(columns=['filename', 'label'] + [str(i) for i in␣
 ↪range(0, 11025+1)], index=None)
        for file in os.listdir(path):
            exact_path = f"{path}/{file}"
            df = get_fft(df, file, exact_path, label)
        return df

    para = []
    for g in GENRES:
        para.append((f"{DATA_DIR}/genres/{g}", g))

    pool = mp.Pool(mp.cpu_count()//2)
    data = pool.starmap(produce_df_fft, para)
```

[ ]:
```
if not os.path.exists(f'{DATA_DIR}/original_fft_amp.csv'):
    df = pd.DataFrame(columns=['filename', 'label']+[str(i) for i in range(0,␣
 ↪11025+1)], index=None)
    for i in range(10):
        df = df.append(data[i], ignore_index=True)
    df.to_csv(f'{DATA_DIR}/original_fft_amp.csv')
    pool.terminate()
```

1.7.2 Loading back data

[ ]:
```
def read_fft_csv(path):
    df = pd.read_csv(path, index_col=0)
    X = df.loc[:, '0':]
    y = df.loc[:, 'label']
    return X, y
X, y = read_fft_csv(f'{DATA_DIR}/original_fft_amp.csv')
```

[ ]:
```
# replace genre name to number labelling
for idx, label in enumerate(GENRES):
    y = y.replace(label, idx)
```

1.7.3 Classification

Seperate data into training set and testing set

[ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=42)
```

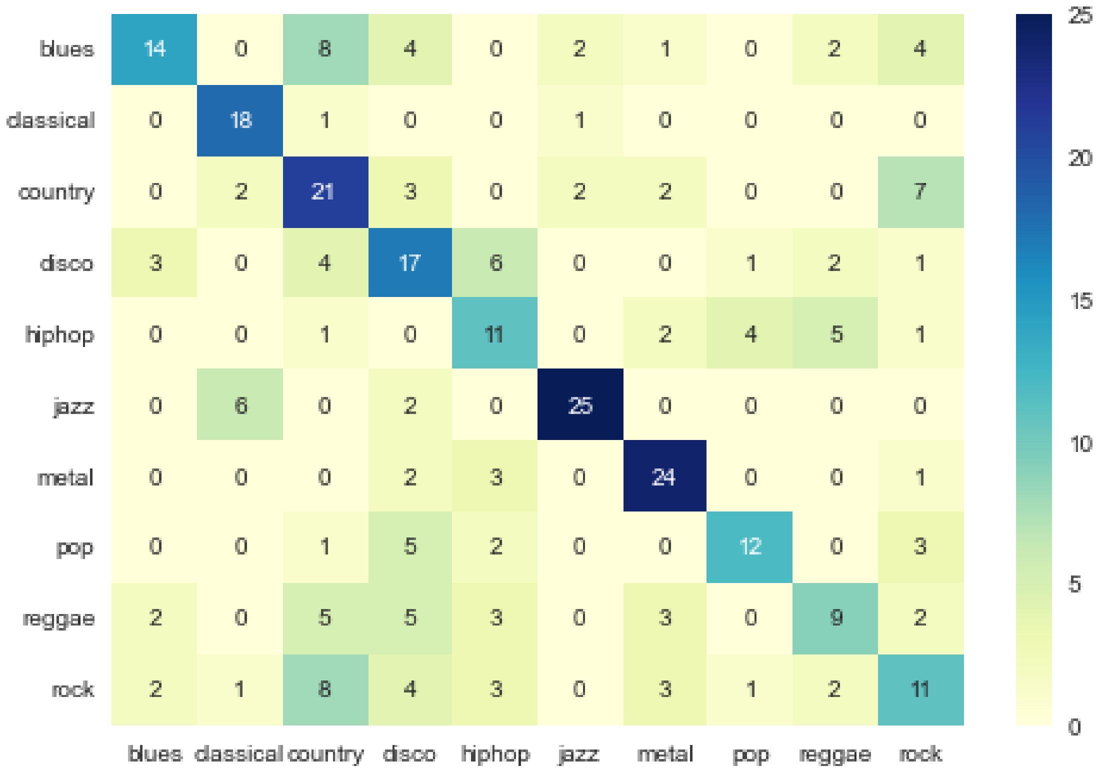
1. Logistic Regression

[ ]:
```
clf = LogisticRegression(max_iter=100000).fit(X_train, y_train)
```

[ ]:
```
print(f"Accuracy on training data: {clf.score(X_train, y_train)}")
print(f"Accuracy on testing data: {clf.score(X_test, y_test)}")
```

```
Accuracy on training data: 1.0
Accuracy on testing data: 0.54
```

[ ]:
```
result_visualise(y_test, clf.predict(X_test), GENRES)
```



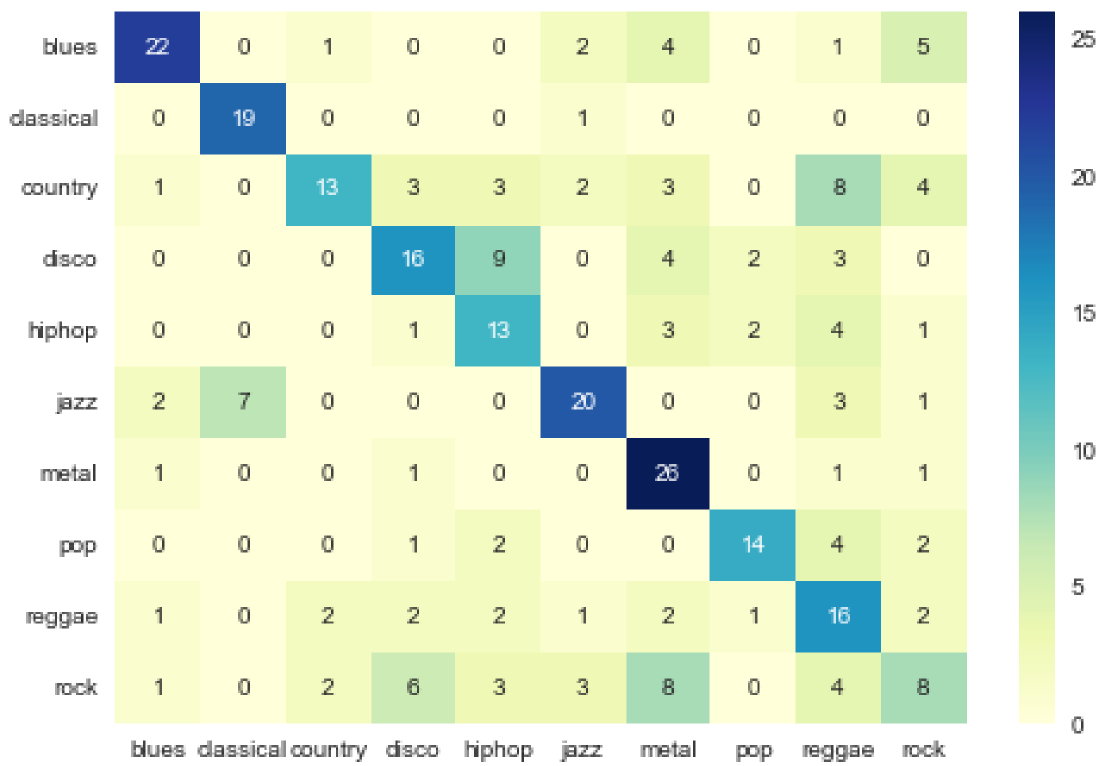
2. Random Forest

[ ]:
```
clf = RandomForestClassifier(1000).fit(X_train, y_train)
```

[ ]:
```
print(f"Accuracy on training data: {clf.score(X_train, y_train)}")
print(f"Accuracy on testing data: {clf.score(X_test, y_test)}")
```

```
Accuracy on training data: 1.0
Accuracy on testing data: 0.5566666666666666
```

[ ]:
```
result_visualise(y_test, clf.predict(X_test), GENRES)
```



3. Simple Neural Network

[ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
↪random_state=42)
scaler = preprocessing.StandardScaler().fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
X_train = tf.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))
X_test = tf.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))
```

[ ]:
```
tf.random.set_seed(42)
```

[ ]:
```
batch_size = 16
epochs = 50

model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Flatten(input_shape=(11026,1)))
model.add(tf.keras.layers.Dense(8192, activation='relu'))
model.add(tf.keras.layers.Dropout(0.05))
model.add(tf.keras.layers.Dense(4096, activation='relu'))
model.add(tf.keras.layers.Dropout(0.05))
model.add(tf.keras.layers.Dense(2048, activation='relu'))
model.add(tf.keras.layers.Dropout(0.05))
```

```
model.add(tf.keras.layers.Dense(1024, activation='relu'))
model.add(tf.keras.layers.Dropout(0.05))
model.add(tf.keras.layers.Dense(512, activation='relu'))
model.add(tf.keras.layers.Dropout(0.05))
model.add(tf.keras.layers.Dense(256, activation='relu'))
model.add(tf.keras.layers.Dropout(0.05))
model.add(tf.keras.layers.Dense(128, activation='relu'))
model.add(tf.keras.layers.Dense(10, activation=tf.nn.softmax))

optimiser = tf.keras.optimizers.Adam()
model.compile(optimizer=optimiser, loss='sparse_categorical_crossentropy',␣
↪metrics=['accuracy'])
```

[ ]:
```
model.fit(X_train, y_train, batch_size=batch_size, epochs=epochs)
```

```
Epoch 1/50
44/44 [==============================] - 3s 34ms/step - loss: 9.0890 - accuracy:
0.1771
Epoch 2/50
44/44 [==============================] - 2s 34ms/step - loss: 2.0578 - accuracy:
0.2657
Epoch 3/50
44/44 [==============================] - 1s 34ms/step - loss: 1.9295 - accuracy:
0.3071
Epoch 4/50
44/44 [==============================] - 1s 34ms/step - loss: 1.8369 - accuracy:
0.3200
Epoch 5/50
44/44 [==============================] - 1s 34ms/step - loss: 1.7502 - accuracy:
0.3414
Epoch 6/50
44/44 [==============================] - 1s 34ms/step - loss: 1.6913 - accuracy:
0.3629
Epoch 7/50
44/44 [==============================] - 1s 34ms/step - loss: 1.6695 - accuracy:
0.3957
Epoch 8/50
44/44 [==============================] - 1s 34ms/step - loss: 1.6175 - accuracy:
0.4514
Epoch 9/50
44/44 [==============================] - 1s 34ms/step - loss: 1.6749 - accuracy:
0.3971
Epoch 10/50
44/44 [==============================] - 2s 34ms/step - loss: 1.5485 - accuracy:
0.4371
Epoch 11/50
44/44 [==============================] - 2s 36ms/step - loss: 1.4416 - accuracy:
0.4771
```

```
Epoch 12/50
44/44 [==============================] - 2s 34ms/step - loss: 1.3010 - accuracy:
0.5000
Epoch 13/50
44/44 [==============================] - 1s 33ms/step - loss: 1.6552 - accuracy:
0.5057
Epoch 14/50
44/44 [==============================] - 1s 34ms/step - loss: 1.4022 - accuracy:
0.5157
Epoch 15/50
44/44 [==============================] - 1s 33ms/step - loss: 1.3934 - accuracy:
0.5243
Epoch 16/50
44/44 [==============================] - 1s 34ms/step - loss: 1.2929 - accuracy:
0.5371
Epoch 17/50
44/44 [==============================] - 1s 34ms/step - loss: 1.1846 - accuracy:
0.5957
Epoch 18/50
44/44 [==============================] - 1s 34ms/step - loss: 1.1916 - accuracy:
0.5986
Epoch 19/50
44/44 [==============================] - 1s 34ms/step - loss: 1.0471 - accuracy:
0.6529
Epoch 20/50
44/44 [==============================] - 1s 34ms/step - loss: 1.2618 - accuracy:
0.5900
Epoch 21/50
44/44 [==============================] - 1s 34ms/step - loss: 1.2833 - accuracy:
0.5700
Epoch 22/50
44/44 [==============================] - 1s 34ms/step - loss: 1.0598 - accuracy:
0.6143
Epoch 23/50
44/44 [==============================] - 1s 34ms/step - loss: 1.0629 - accuracy:
0.6371
Epoch 24/50
44/44 [==============================] - 1s 34ms/step - loss: 1.2792 - accuracy:
0.6143
Epoch 25/50
44/44 [==============================] - 1s 33ms/step - loss: 1.4477 - accuracy:
0.5443
Epoch 26/50
44/44 [==============================] - 1s 34ms/step - loss: 1.3086 - accuracy:
0.6129
Epoch 27/50
44/44 [==============================] - 1s 34ms/step - loss: 1.2875 - accuracy:
0.6300
```

```
Epoch 28/50
44/44 [==============================] - 1s 34ms/step - loss: 1.0647 - accuracy:
0.6557
Epoch 29/50
44/44 [==============================] - 1s 33ms/step - loss: 1.1423 - accuracy:
0.6629
Epoch 30/50
44/44 [==============================] - 1s 34ms/step - loss: 1.2524 - accuracy:
0.5657
Epoch 31/50
44/44 [==============================] - 1s 33ms/step - loss: 1.1243 - accuracy:
0.6400
Epoch 32/50
44/44 [==============================] - 1s 34ms/step - loss: 1.1991 - accuracy:
0.6329
Epoch 33/50
44/44 [==============================] - 1s 33ms/step - loss: 1.0263 - accuracy:
0.6514
Epoch 34/50
44/44 [==============================] - 1s 34ms/step - loss: 0.9360 - accuracy:
0.6986
Epoch 35/50
44/44 [==============================] - 1s 33ms/step - loss: 0.8558 - accuracy:
0.7300
Epoch 36/50
44/44 [==============================] - 1s 34ms/step - loss: 0.8621 - accuracy:
0.7071
Epoch 37/50
44/44 [==============================] - 1s 33ms/step - loss: 0.7700 - accuracy:
0.7157
Epoch 38/50
44/44 [==============================] - 1s 33ms/step - loss: 0.7080 - accuracy:
0.7657
Epoch 39/50
44/44 [==============================] - 1s 34ms/step - loss: 0.8243 - accuracy:
0.7386
Epoch 40/50
44/44 [==============================] - 1s 34ms/step - loss: 1.4626 - accuracy:
0.5514
Epoch 41/50
44/44 [==============================] - 1s 34ms/step - loss: 1.3695 - accuracy:
0.5271
Epoch 42/50
44/44 [==============================] - 1s 33ms/step - loss: 1.0680 - accuracy:
0.6314
Epoch 43/50
44/44 [==============================] - 1s 33ms/step - loss: 1.1802 - accuracy:
0.6143
```

```
Epoch 44/50
44/44 [==============================] - 1s 33ms/step - loss: 0.9230 - accuracy:
0.6771
Epoch 45/50
44/44 [==============================] - 1s 33ms/step - loss: 0.8356 - accuracy:
0.7229
Epoch 46/50
44/44 [==============================] - 1s 34ms/step - loss: 0.8313 - accuracy:
0.7057
Epoch 47/50
44/44 [==============================] - 1s 33ms/step - loss: 0.7413 - accuracy:
0.7471
Epoch 48/50
44/44 [==============================] - 1s 34ms/step - loss: 0.6890 - accuracy:
0.7443
Epoch 49/50
44/44 [==============================] - 1s 34ms/step - loss: 0.7804 - accuracy:
0.7500
Epoch 50/50
44/44 [==============================] - 1s 34ms/step - loss: 0.8178 - accuracy:
0.7243
```

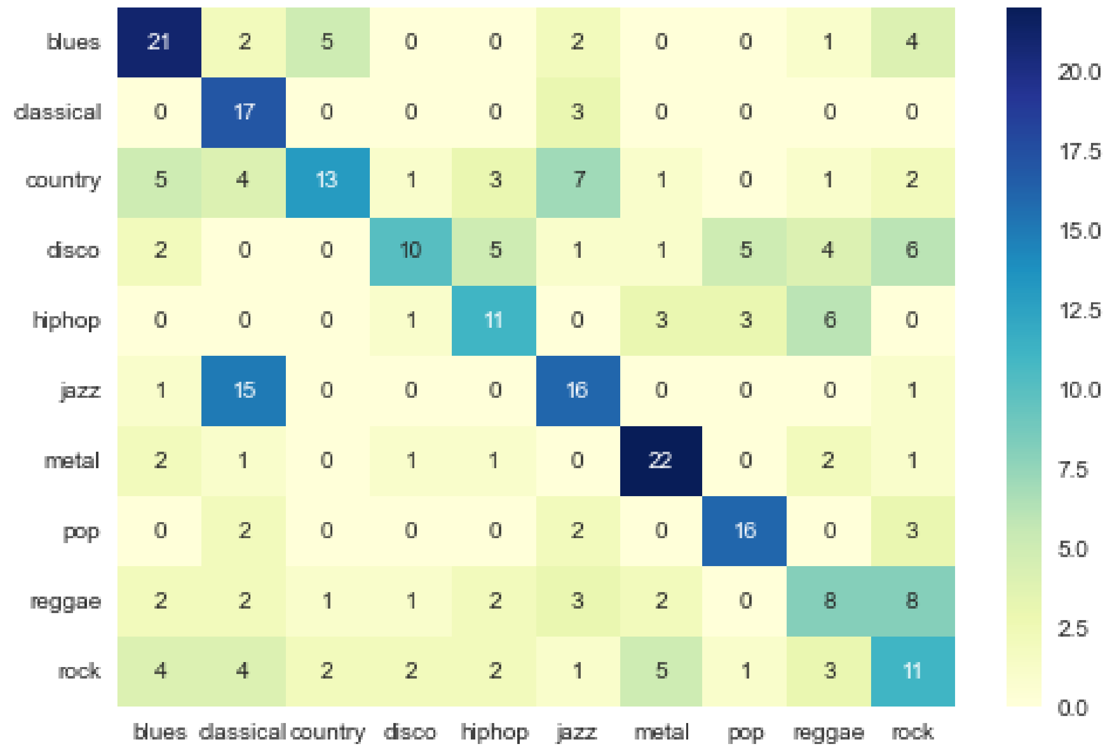
[ ]:
```
<tensorflow.python.keras.callbacks.History at 0x1bd254a83a0>
```

[ ]:
```
model.evaluate(X_test, y_test)
```

```
10/10 [==============================] - 0s 7ms/step - loss: 2.2669 - accuracy:
0.4833
```

[ ]:
```
[2.266932964324951, 0.4833333194255829]
```

[ ]:
```
result_visualise(y_test, np.argmax(model.predict(X_test), axis=-1), GENRES)
```



[ ]:
```
batch_size = 16
epochs = 50

model2 = tf.keras.models.Sequential()
model2.add(tf.keras.layers.Conv1D(64, 2, activation='relu',␣
↪input_shape=(11026,1))) # edited
model2.add(tf.keras.layers.MaxPooling1D(pool_size = 2))
model2.add(tf.keras.layers.Conv1D(128, 2, activation = 'relu'))
model2.add(tf.keras.layers.MaxPooling1D(pool_size = 2))
model2.add(tf.keras.layers.Conv1D(256, 2, activation = 'relu'))
model2.add(tf.keras.layers.MaxPooling1D(pool_size = 4))
model2.add(tf.keras.layers.Conv1D(512, 2, activation = 'relu'))
# model2.add(tf.keras.layers.MaxPooling1D(pool_size = 4))
# model2.add(tf.keras.layers.Conv1D(1024, 2, activation = 'relu'))
model2.add(tf.keras.layers.MaxPooling1D(pool_size = 4))
model2.add(tf.keras.layers.Flatten())
model2.add(tf.keras.layers.Dropout(0.05))
model2.add(tf.keras.layers.Dense(2048, activation = 'relu'))
model2.add(tf.keras.layers.Dropout(0.05))
model2.add(tf.keras.layers.Dense(1024, activation = 'relu'))
model2.add(tf.keras.layers.Dropout(0.05))
model2.add(tf.keras.layers.Dense(512, activation = 'relu'))
```

```
model2.add(tf.keras.layers.Dense(10, activation='softmax'))

optimiser = tf.keras.optimizers.Adam()
model2.compile(optimizer=optimiser, loss='sparse_categorical_crossentropy',␣
↪metrics=['accuracy'])
#model2.summary()
```

[ ]:
```
model2.fit(X_train, y_train, batch_size=batch_size, epochs=epochs)
```

```
Epoch 1/50
44/44 [==============================] - 7s 83ms/step - loss: 2.3033 - accuracy:
0.1871
Epoch 2/50
44/44 [==============================] - 3s 77ms/step - loss: 1.7405 - accuracy:
0.3757
Epoch 3/50
44/44 [==============================] - 3s 77ms/step - loss: 1.6242 - accuracy:
0.4414
Epoch 4/50
44/44 [==============================] - 3s 77ms/step - loss: 1.3350 - accuracy:
0.5214
Epoch 5/50
44/44 [==============================] - 3s 78ms/step - loss: 1.0783 - accuracy:
0.6143
Epoch 6/50
44/44 [==============================] - 3s 78ms/step - loss: 0.8610 - accuracy:
0.7000
Epoch 7/50
44/44 [==============================] - 3s 77ms/step - loss: 0.6669 - accuracy:
0.7757
Epoch 8/50
44/44 [==============================] - 3s 77ms/step - loss: 0.5560 - accuracy:
0.8114
Epoch 9/50
44/44 [==============================] - 3s 77ms/step - loss: 0.3582 - accuracy:
0.8829
Epoch 10/50
44/44 [==============================] - 3s 77ms/step - loss: 0.3496 - accuracy:
0.8943
Epoch 11/50
44/44 [==============================] - 3s 77ms/step - loss: 0.2585 - accuracy:
0.9229
Epoch 12/50
44/44 [==============================] - 3s 78ms/step - loss: 0.1172 - accuracy:
0.9657
Epoch 13/50
44/44 [==============================] - 3s 77ms/step - loss: 0.1708 - accuracy:
0.9571
```

```
Epoch 14/50
44/44 [==============================] - 3s 77ms/step - loss: 0.1454 - accuracy:
0.9529
Epoch 15/50
44/44 [==============================] - 3s 78ms/step - loss: 0.2127 - accuracy:
0.9429
Epoch 16/50
44/44 [==============================] - 3s 77ms/step - loss: 0.2637 - accuracy:
0.9300
Epoch 17/50
44/44 [==============================] - 3s 77ms/step - loss: 0.0863 - accuracy:
0.9686
Epoch 18/50
44/44 [==============================] - 3s 76ms/step - loss: 0.1647 - accuracy:
0.9557
Epoch 19/50
44/44 [==============================] - 3s 77ms/step - loss: 0.1807 - accuracy:
0.9529
Epoch 20/50
44/44 [==============================] - 3s 76ms/step - loss: 0.3906 - accuracy:
0.9214
Epoch 21/50
44/44 [==============================] - 3s 78ms/step - loss: 0.1729 - accuracy:
0.9529
Epoch 22/50
44/44 [==============================] - 3s 76ms/step - loss: 0.1286 - accuracy:
0.9657
Epoch 23/50
44/44 [==============================] - 3s 76ms/step - loss: 0.0640 - accuracy:
0.9829
Epoch 24/50
44/44 [==============================] - 3s 77ms/step - loss: 0.0370 - accuracy:
0.9900
Epoch 25/50
44/44 [==============================] - 3s 77ms/step - loss: 0.0250 - accuracy:
0.9914
Epoch 26/50
44/44 [==============================] - 3s 77ms/step - loss: 0.0262 - accuracy:
0.9929
Epoch 27/50
44/44 [==============================] - 3s 77ms/step - loss: 0.0246 - accuracy:
0.9943
Epoch 28/50
44/44 [==============================] - 3s 77ms/step - loss: 0.0215 - accuracy:
0.9957
Epoch 29/50
44/44 [==============================] - 3s 76ms/step - loss: 0.2604 - accuracy:
0.9500
```

```
Epoch 30/50
44/44 [==============================] - 3s 77ms/step - loss: 0.1824 - accuracy:
0.9557
Epoch 31/50
44/44 [==============================] - 3s 77ms/step - loss: 0.1070 - accuracy:
0.9671
Epoch 32/50
44/44 [==============================] - 3s 76ms/step - loss: 0.0333 - accuracy:
0.9914
Epoch 33/50
44/44 [==============================] - 3s 77ms/step - loss: 0.0274 - accuracy:
0.9914
Epoch 34/50
44/44 [==============================] - 3s 76ms/step - loss: 0.0196 - accuracy:
0.9929
Epoch 35/50
44/44 [==============================] - 3s 77ms/step - loss: 0.0199 - accuracy:
0.9929
Epoch 36/50
44/44 [==============================] - 3s 77ms/step - loss: 0.0137 - accuracy:
0.9971
Epoch 37/50
44/44 [==============================] - 3s 77ms/step - loss: 0.0128 - accuracy:
0.9971
Epoch 38/50
44/44 [==============================] - 3s 77ms/step - loss: 0.0143 - accuracy:
0.9971
Epoch 39/50
44/44 [==============================] - 3s 77ms/step - loss: 0.0120 - accuracy:
0.9957
Epoch 40/50
44/44 [==============================] - 3s 77ms/step - loss: 0.1778 - accuracy:
0.9700
Epoch 41/50
44/44 [==============================] - 3s 76ms/step - loss: 0.0788 - accuracy:
0.9786
Epoch 42/50
44/44 [==============================] - 3s 77ms/step - loss: 0.1340 - accuracy:
0.9700
Epoch 43/50
44/44 [==============================] - 3s 76ms/step - loss: 0.1412 - accuracy:
0.9686
Epoch 44/50
44/44 [==============================] - 3s 77ms/step - loss: 0.1097 - accuracy:
0.9729
Epoch 45/50
44/44 [==============================] - 3s 78ms/step - loss: 0.1337 - accuracy:
0.9686
```

```
Epoch 46/50
44/44 [==============================] - 3s 76ms/step - loss: 0.1090 - accuracy:
0.9800
Epoch 47/50
44/44 [==============================] - 3s 76ms/step - loss: 0.0448 - accuracy:
0.9886
Epoch 48/50
44/44 [==============================] - 3s 76ms/step - loss: 0.1014 - accuracy:
0.9786
Epoch 49/50
44/44 [==============================] - 3s 76ms/step - loss: 0.3486 - accuracy:
0.9786
Epoch 50/50
44/44 [==============================] - 3s 76ms/step - loss: 0.3571 - accuracy:
0.9329
```

[ ]: <tensorflow.python.keras.callbacks.History at 0x209adaef5e0>

[ ]:
```
model2.evaluate(X_test, y_test)
```

```
10/10 [==============================] - 1s 27ms/step - loss: 2.9693 - accuracy:
0.5300
```

[ ]: [2.9692740440368652, 0.5299999713897705]

1.8 Fourth Attempt

1.8.1 Make a larger dataset

Apply the following effects to all audio files to obtain a largest dataset: 1. Reverb / Chorus 2. Distortion 3. Pitch shift 4. Time stretch 5. Noise

[7]:
```
EFFECTED_DIR = 'effected'
```

[ ]:
```
if not os.path.exists(EFFECTED_DIR):
    os.makedirs(EFFECTED_DIR)

for g in GENRES:
    if not os.path.exists(f'{EFFECTED_DIR}/{g}'):
        os.makedirs(f'{EFFECTED_DIR}/{g}')
```

[ ]:
```
# Notice that the following code will use all of the CPU resources
# i7-9750H takes around 5 hours to run
if not os.path.exists(f'{EFFECTED_DIR}/effected_fft_amp.csv'):
    def generate_audio(path, genre):
        # in python multiprocess, libraries are required to import again
        import matplotlib.pyplot as plt
        import librosa
        import librosa.display
```

```
import soundfile as sf
from IPython.display import Audio
from pedalboard import Pedalboard, Convolution, Compressor, Chorus,␣
↪Gain, Reverb, Limiter, LadderFilter, Phaser, NoiseGate, Distortion
from audiomentations import Compose, AddGaussianNoise
import os

effects_1 = {"original": None,
        "reverb1": Reverb(room_size=0.25),
        #"reverb2": Reverb(room_size=0.75),
        "chorus": Chorus(),

        }

effects_2 = {"original": None,
            "distortion1": Distortion(drive_db=10),
            "distortion2": Distortion(drive_db=-10),
            }

effects_3 = {"original": None,
            "pitchshift1": 4,
            #"pitchshift2": 8,
            "pitchshift3": -4,
            #"pitchshift4": 8,
            }

effects_4 = {"original": None,
            "timestretch1": 0.85,
            "timestretch2": 1.15,
            }

effects_5 = {"noise1": Compose([AddGaussianNoise(min_amplitude=0.001,␣
↪max_amplitude=0.005)]),
            #"noise2": Compose([AddGaussianNoise(min_amplitude=0.001,␣
↪max_amplitude=0.05)]),
            #"original": None
            }
idx = 1
for filename in os.listdir(path):

    i = 1
    y, sr = librosa.load(f"{path}/{filename}")
    for n1, e1 in list(effects_1.items()):
        for n2, e2 in list(effects_2.items()):
            for n3, e3 in list(effects_3.items()):
                for n4, e4 in list(effects_4.items()):
                    for n5, e5 in list(effects_5.items()):
```

```
                                t = y
                                t = e1(t, sr) if e1 is not None else t
                                t = e2(t, sr) if e2 is not None else t
                                t = librosa.effects.pitch_shift(t, sr,␣
↪n_steps=e3) if e3 is not None else t
                                t = librosa.effects.time_stretch(t, e4) if e4␣
↪is not None else t
                                t = e5(t, sr) if e5 is not None else t
                                sf.write(f'effected/{genre}/
↪{idx}-{i}-{filename[:-4]}{f"_{n1}" if e1 is not None else ""}{f"_{n2}" if e2␣
↪is not None else ""}{f"_{n3}" if e3 is not None else ""}{f"_{n4}" if e4 is␣
↪not None else ""}{f"_{n5}" if e5 is not None else ""}.wav', t, sr)
                                i += 1
            idx += 1

    para = []
    for g in GENRES:
        para.append((f"{DATA_DIR}/genres/{g}", g))

    print("CPU NUMBER:", mp.cpu_count())
    pool = mp.Pool(mp.cpu_count()//2)
    pool.starmap(generate_audio, para)

    pool.close()
    pool.join()
```

1.8.2 Feature extraction

[ ]:
```
# Notice that the following code will use all of the CPU resources
# i7-9750H takes around 2.5 hours to run
if not os.path.exists(f'{EFFECTED_DIR}/effected_fft_amp.csv'):
    def produce_df_fft(path, label):
        def get_fft(df, filename, path, label):
            y, signal = scipy.io.wavfile.read(path)
            fft_spectrum = np.fft.rfft(signal)
            freq = np.fft.rfftfreq(signal.size, d=1./y)
            fft_spectrum_abs = np.abs(fft_spectrum)

            data = np.column_stack((freq, np.round(fft_spectrum_abs)))
            tmpdf = pd.DataFrame(data, columns=['freq', 'amp'])
            tmpdf.loc[:, 'freq'] = np.round(tmpdf['freq'])

            return df.append(
                pd.DataFrame(
                    np.array(tmpdf.groupby('freq').max('amp')).reshape(1, -1),
                    columns=[str(i) for i in range(0, 11025+1)],
                ).assign(filename=filename).assign(label=label),
```

```
                ignore_index=True
            )

        import pandas as pd, numpy as np, scipy.io.wavfile, os
        df = pd.DataFrame(columns=['filename', 'label'] + [str(i) for i in␣
↪range(0, 11025+1)], index=None)
        for file in os.listdir(path):
            exact_path = f"{path}/{file}"
            df = get_fft(df, file, exact_path, label)
        return df

    para = []
    for g in GENRES:
        para.append((f"{EFFECTED_DIR}/{g}", g))

    pool = mp.Pool(mp.cpu_count()//2)
    data = pool.starmap(produce_df_fft, para)
```

[ ]:
```
# i7-9750H takes around 0.5 hours to run
if not os.path.exists(f'{EFFECTED_DIR}/effected_fft_amp.csv'):
    df = pd.DataFrame(columns=['filename', 'label']+[str(i) for i in range(0,␣
↪11025+1)], index=None)
    for i in range(10):
        df = df.append(data[i], ignore_index=True)
    df.to_csv(f'{EFFECTED_DIR}/effected_fft_amp.csv')
    pool.close()
    pool.join()
```

1.8.3 Loading back data

[8]:
```
# The following code will take around 10 minutes to run
def read_fft_csv(path):
    df = pd.read_csv(path, index_col=0)
    X = df.loc[:, '0':]
    y = df.loc[:, 'label']
    return X, y
X, y = read_fft_csv(f'{EFFECTED_DIR}/effected_fft_amp.csv')
```

[9]:
```
# replace genre name to number labelling
for idx, label in enumerate(GENRES):
    y = y.replace(label, idx)
```

[ ]:
```
gc.collect()
```

[ ]: 4727

1.8.4 Classification

Seperate data into training set and testing set

[10]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
↪random_state=42)
X_train = X_train.astype(np.float32)
X_test = X_test.astype(np.float32)
```

Try reducing dimension before classification

[ ]:
```
# Principal component analysis is used for dimensionality reduction
pca = PCA(n_components=2000)
pca.fit(X_train)
```

[ ]:
```
PCA(n_components=2000)
```

[ ]:
```
X_train_pca = pca.fit_transform(X_train)
```

[ ]:
```
X_train_pca = np.reshape(X_train_pca, (X_train_pca.shape[0], X_train_pca.
↪shape[1], 1))
```

[ ]:
```
model3 = tf.keras.Sequential()

model3.add(tf.keras.layers.Flatten(input_shape=(X_train_pca.shape[1],␣
↪X_train_pca.shape[2])))
model3.add(tf.keras.layers.Dense(1024, activation='relu'))
model3.add(tf.keras.layers.Dense(512, activation='relu'))
model3.add(tf.keras.layers.Dense(256, activation='relu'))
model3.add(tf.keras.layers.Dense(128, activation='relu'))
model3.add(tf.keras.layers.Dense(10, activation='softmax'))

optimiser = tf.keras.optimizers.Adam()
model3.compile(optimizer=optimiser, loss='sparse_categorical_crossentropy',␣
↪metrics=['accuracy'])
```

[ ]:
```
model3.fit(X_train_pca, y_train, epochs=30, batch_size=64)
```

```
Epoch 1/30
886/886 [==============================] - 3s 3ms/step - loss: 260586.4688 -
accuracy: 0.5738
Epoch 2/30
886/886 [==============================] - 2s 3ms/step - loss: 11201.1475 -
accuracy: 0.7960
Epoch 3/30
886/886 [==============================] - 3s 3ms/step - loss: 10437.2061 -
accuracy: 0.8136
Epoch 4/30
886/886 [==============================] - 2s 3ms/step - loss: 7793.9131 -
accuracy: 0.8513
```

```
Epoch 5/30
886/886 [==============================] - 2s 3ms/step - loss: 8184.2476 -
accuracy: 0.8395
Epoch 6/30
886/886 [==============================] - 2s 3ms/step - loss: 4929.7827 -
accuracy: 0.8638
Epoch 7/30
886/886 [==============================] - 3s 3ms/step - loss: 6145.6729 -
accuracy: 0.8226
Epoch 8/30
886/886 [==============================] - 3s 3ms/step - loss: 1590.8177 -
accuracy: 0.7743
Epoch 9/30
886/886 [==============================] - 3s 3ms/step - loss: 355.1204 -
accuracy: 0.3790
Epoch 10/30
886/886 [==============================] - 3s 3ms/step - loss: 13.9998 -
accuracy: 0.1708
Epoch 11/30
886/886 [==============================] - 3s 3ms/step - loss: 11.4839 -
accuracy: 0.1480
Epoch 12/30
886/886 [==============================] - 3s 3ms/step - loss: 6.6083 -
accuracy: 0.1396
Epoch 13/30
886/886 [==============================] - 3s 3ms/step - loss: 215.7458 -
accuracy: 0.1018
Epoch 14/30
886/886 [==============================] - 3s 3ms/step - loss: 2.3012 -
accuracy: 0.1000
Epoch 15/30
886/886 [==============================] - 3s 3ms/step - loss: 2.3012 -
accuracy: 0.1020
Epoch 16/30
886/886 [==============================] - 4s 4ms/step - loss: 2.3012 -
accuracy: 0.1003
Epoch 17/30
886/886 [==============================] - 3s 4ms/step - loss: 2.3012 -
accuracy: 0.0999
Epoch 18/30
886/886 [==============================] - 3s 4ms/step - loss: 2.3012 -
accuracy: 0.0995
Epoch 19/30
886/886 [==============================] - 3s 4ms/step - loss: 2.3012 -
accuracy: 0.1003
Epoch 20/30
886/886 [==============================] - 3s 4ms/step - loss: 2.3012 -
accuracy: 0.0995
```

```
Epoch 21/30
886/886 [==============================] - 3s 3ms/step - loss: 2.3012 -
accuracy: 0.0997
Epoch 22/30
886/886 [==============================] - 3s 3ms/step - loss: 2.3012 -
accuracy: 0.0998
Epoch 23/30
886/886 [==============================] - 3s 3ms/step - loss: 2.3012 -
accuracy: 0.0994
Epoch 24/30
886/886 [==============================] - 3s 3ms/step - loss: 2.3012 -
accuracy: 0.0999
Epoch 25/30
886/886 [==============================] - 3s 3ms/step - loss: 2.3012 -
accuracy: 0.1011
Epoch 26/30
886/886 [==============================] - 3s 3ms/step - loss: 2.3012 -
accuracy: 0.0996
Epoch 27/30
886/886 [==============================] - 3s 3ms/step - loss: 2.3012 -
accuracy: 0.0991
Epoch 28/30
886/886 [==============================] - 3s 3ms/step - loss: 2.3012 -
accuracy: 0.1017
Epoch 29/30
886/886 [==============================] - 3s 3ms/step - loss: 2.3012 -
accuracy: 0.1016
Epoch 30/30
886/886 [==============================] - 3s 3ms/step - loss: 2.3012 -
accuracy: 0.1001
```

[ ]:
```
<tensorflow.python.keras.callbacks.History at 0x209af177520>
```

[ ]:
```
X_test_pca = pca.fit_transform(X_test)
X_test_pca = np.reshape(X_test_pca, (X_test_pca.shape[0], X_test_pca.shape[1],␣
↪1))

model3.evaluate(X_test_pca, y_test)
```

```
760/760 [==============================] - 2s 3ms/step - loss: 2.3030 -
accuracy: 0.0974
```

[ ]:
```
[2.3029704093933105, 0.09740740805864334]
```

Scaling data to have unit norm

[11]:
```
scaler = preprocessing.StandardScaler().fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

[12]:
```python
import joblib
joblib.dump(scaler, f'{EFFECTED_DIR}/scaler.pkl')
```

[12]: ['effected/scaler.pkl']

[ ]:
```python
X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))
X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))
```

CNN

[ ]:
```python
batch_size = 64
epochs = 50

model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Conv1D(64, 2, activation='relu',␣
 ↪input_shape=(11026,1))) # edited
model.add(tf.keras.layers.MaxPooling1D(pool_size = 2))
model.add(tf.keras.layers.Conv1D(128, 2, activation = 'relu'))
model.add(tf.keras.layers.MaxPooling1D(pool_size = 2))
model.add(tf.keras.layers.Conv1D(256, 2, activation = 'relu'))
model.add(tf.keras.layers.MaxPooling1D(pool_size = 4))
model.add(tf.keras.layers.Conv1D(512, 2, activation = 'relu'))
model.add(tf.keras.layers.MaxPooling1D(pool_size = 4)) #
model.add(tf.keras.layers.Conv1D(1024, 2, activation = 'relu')) #
model.add(tf.keras.layers.MaxPooling1D(pool_size = 4))
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dropout(0.05))
model.add(tf.keras.layers.Dense(2048, activation = 'relu'))
model.add(tf.keras.layers.Dropout(0.05))
model.add(tf.keras.layers.Dense(1024, activation = 'relu'))
model.add(tf.keras.layers.Dropout(0.05))
model.add(tf.keras.layers.Dense(512, activation = 'relu'))
model.add(tf.keras.layers.Dense(10, activation='softmax'))

optimiser = tf.keras.optimizers.Adam()
model.compile(optimizer=optimiser, loss='sparse_categorical_crossentropy',␣
 ↪metrics=['accuracy'])
```

[ ]:
```python
model.fit(X_train, y_train, batch_size=batch_size, epochs=epochs)
```

```
Epoch 1/50
886/886 [==============================] - 184s 198ms/step - loss: 1.4585 -
accuracy: 0.4699
Epoch 2/50
886/886 [==============================] - 187s 211ms/step - loss: 0.7457 -
accuracy: 0.7375
Epoch 3/50
886/886 [==============================] - 179s 202ms/step - loss: 0.4243 -
accuracy: 0.8546
```

```
Epoch 4/50
886/886 [==============================] - 179s 202ms/step - loss: 0.2580 -
accuracy: 0.9142
Epoch 5/50
886/886 [==============================] - 180s 204ms/step - loss: 0.1947 -
accuracy: 0.9373
Epoch 6/50
886/886 [==============================] - 178s 201ms/step - loss: 0.1674 -
accuracy: 0.9470
Epoch 7/50
886/886 [==============================] - 179s 202ms/step - loss: 0.1222 -
accuracy: 0.9625
Epoch 8/50
886/886 [==============================] - 178s 201ms/step - loss: 0.1057 -
accuracy: 0.9667
Epoch 9/50
886/886 [==============================] - 178s 201ms/step - loss: 0.0987 -
accuracy: 0.9689
Epoch 10/50
886/886 [==============================] - 178s 201ms/step - loss: 0.0916 -
accuracy: 0.9725
Epoch 11/50
886/886 [==============================] - 191s 215ms/step - loss: 0.0970 -
accuracy: 0.9712
Epoch 12/50
886/886 [==============================] - 200s 226ms/step - loss: 0.0748 -
accuracy: 0.9775
Epoch 13/50
886/886 [==============================] - 181s 205ms/step - loss: 0.0716 -
accuracy: 0.9784
Epoch 14/50
886/886 [==============================] - 179s 202ms/step - loss: 0.0694 -
accuracy: 0.9799
Epoch 15/50
886/886 [==============================] - 200s 225ms/step - loss: 0.0671 -
accuracy: 0.9799
Epoch 16/50
886/886 [==============================] - 198s 223ms/step - loss: 0.0599 -
accuracy: 0.9827
Epoch 17/50
886/886 [==============================] - 199s 225ms/step - loss: 0.0719 -
accuracy: 0.9797
Epoch 18/50
886/886 [==============================] - 200s 226ms/step - loss: 0.0653 -
accuracy: 0.9809
Epoch 19/50
886/886 [==============================] - 202s 228ms/step - loss: 0.0503 -
accuracy: 0.9849
```

```
Epoch 20/50
886/886 [==============================] - 191s 215ms/step - loss: 0.0549 -
accuracy: 0.9835
Epoch 21/50
886/886 [==============================] - 160s 181ms/step - loss: 0.0472 -
accuracy: 0.9862
Epoch 22/50
886/886 [==============================] - 176s 199ms/step - loss: 0.0550 -
accuracy: 0.9845
Epoch 23/50
886/886 [==============================] - 189s 213ms/step - loss: 0.0499 -
accuracy: 0.9861
Epoch 24/50
886/886 [==============================] - 189s 213ms/step - loss: 0.0602 -
accuracy: 0.9830
Epoch 25/50
886/886 [==============================] - 189s 214ms/step - loss: 0.0415 -
accuracy: 0.9878
Epoch 26/50
886/886 [==============================] - 189s 214ms/step - loss: 0.0544 -
accuracy: 0.9858
Epoch 27/50
886/886 [==============================] - 190s 214ms/step - loss: 0.0357 -
accuracy: 0.9895
Epoch 28/50
886/886 [==============================] - 189s 214ms/step - loss: 0.0441 -
accuracy: 0.9883
Epoch 29/50
886/886 [==============================] - 190s 214ms/step - loss: 0.0442 -
accuracy: 0.9879
Epoch 30/50
886/886 [==============================] - 190s 214ms/step - loss: 0.0314 -
accuracy: 0.9912
Epoch 31/50
886/886 [==============================] - 190s 214ms/step - loss: 0.0477 -
accuracy: 0.9870
Epoch 32/50
886/886 [==============================] - 190s 214ms/step - loss: 0.0389 -
accuracy: 0.9893
Epoch 33/50
886/886 [==============================] - 190s 214ms/step - loss: 0.0428 -
accuracy: 0.9884
Epoch 34/50
886/886 [==============================] - 189s 213ms/step - loss: 0.0428 -
accuracy: 0.9889
Epoch 35/50
886/886 [==============================] - 189s 214ms/step - loss: 0.0379 -
accuracy: 0.9893
```

```
Epoch 36/50
886/886 [==============================] - 189s 213ms/step - loss: 0.0442 -
accuracy: 0.9878
Epoch 37/50
886/886 [==============================] - 189s 213ms/step - loss: 0.0375 -
accuracy: 0.9905
Epoch 38/50
886/886 [==============================] - 189s 213ms/step - loss: 0.0362 -
accuracy: 0.9902
Epoch 39/50
886/886 [==============================] - 189s 213ms/step - loss: 0.0333 -
accuracy: 0.9908
Epoch 40/50
886/886 [==============================] - 179s 202ms/step - loss: 0.0461 -
accuracy: 0.9888
Epoch 41/50
886/886 [==============================] - 178s 201ms/step - loss: 0.0419 -
accuracy: 0.9897
Epoch 42/50
886/886 [==============================] - 178s 201ms/step - loss: 0.0292 -
accuracy: 0.9922
Epoch 43/50
886/886 [==============================] - 178s 201ms/step - loss: 0.0364 -
accuracy: 0.9911
Epoch 44/50
886/886 [==============================] - 178s 201ms/step - loss: 0.0370 -
accuracy: 0.9905
Epoch 45/50
886/886 [==============================] - 178s 201ms/step - loss: 0.0310 -
accuracy: 0.9917
Epoch 46/50
886/886 [==============================] - 178s 201ms/step - loss: 0.0409 -
accuracy: 0.9902
Epoch 47/50
886/886 [==============================] - 178s 201ms/step - loss: 0.0494 -
accuracy: 0.9881
Epoch 48/50
886/886 [==============================] - 178s 201ms/step - loss: 0.0322 -
accuracy: 0.9916
Epoch 49/50
886/886 [==============================] - 178s 201ms/step - loss: 0.0293 -
accuracy: 0.9926
Epoch 50/50
886/886 [==============================] - 177s 200ms/step - loss: 0.0432 -
accuracy: 0.9897
```

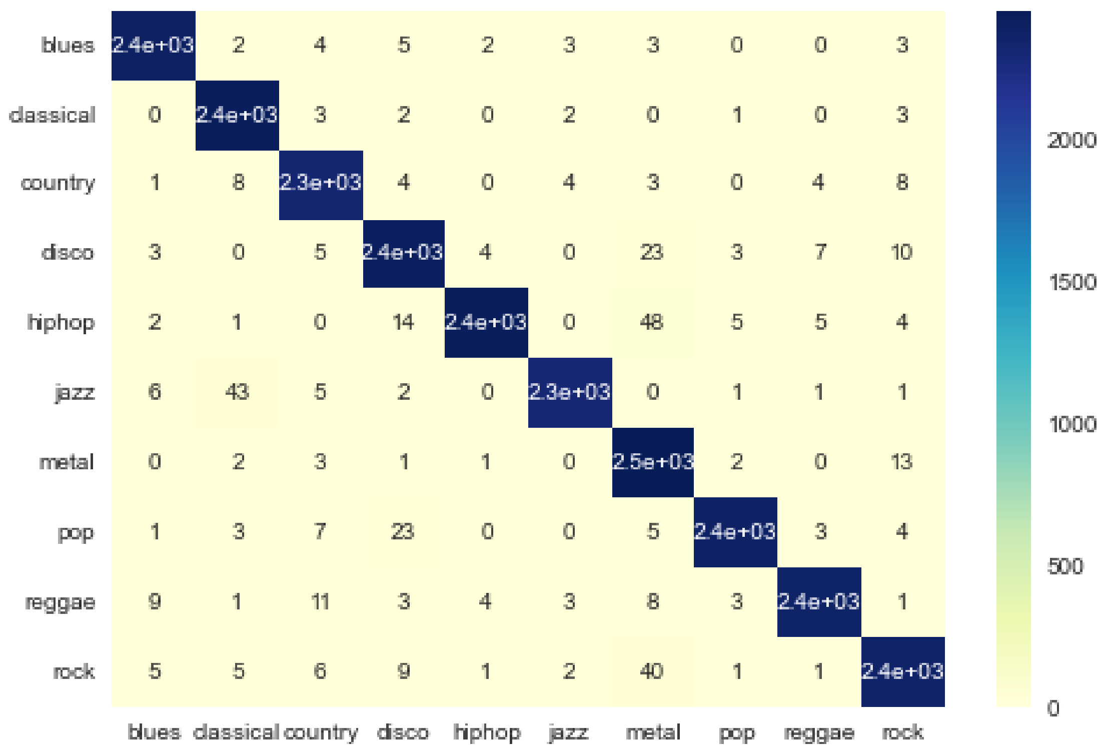
```
[ ]: <tensorflow.python.keras.callbacks.History at 0x21649282bb0>
```

```
[ ]: model.evaluate(X_test, y_test)
```

```
760/760 [==============================] - 31s 39ms/step - loss: 0.0653 -
accuracy: 0.9819
```

```
[ ]: [0.06530376523733139, 0.981934130191803]
```

```
[ ]: result_visualise(y_test, np.argmax(model.predict(X_test), axis=-1), GENRES)
```



Convert the Tensorflow lite model

```
[ ]: converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()

# Save the model
with open('model2.tflite', 'wb') as f:
  f.write(tflite_model)
```

```
INFO:tensorflow:Assets written to:
C:\Users\YEEKII~1\AppData\Local\Temp\tmp0h52dfu5\assets
```