CS448 - Lab 8: Audio Classification

In [ ]:
```python
### IMPORTS & SETUP ###

import os
import random

import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = (6, 3)

import numpy as np

from scipy.io import wavfile

from sklearn.model_selection import train_test_split as tts
from sklearn.mixture import GaussianMixture as  GMM

import librosa

DRAW_GRAPHS = True
```

In [ ]:
```python
# STFT from Lab 1
def stft(input_sound, dft_size, hop_size, zero_pad, window):
    # Creating the n-1 frames
    frames = []
    idx = 0
    for idx in range(0, len(input_sound) - dft_size, hop_size):
        frames.append(np.multiply(input_sound[idx : idx + dft_size], window))
    idx += hop_size

    # Creating the last frame accounting for padding
    last_frame = np.multiply(
        np.append(input_sound[idx:-1], np.zeros(idx + dft_size - len(input_sound) + 1)),
        window,
    )
    frames.append(last_frame)

    # Convert to numpy array
    frames = np.array(frames, dtype=float)

    # Compute the DFT of each frame
    dft_frames = np.fft.rfft(frames, dft_size + zero_pad)
    return dft_frames


# STFT parameters
DFT_SIZE = 1024
HOP_SIZE = DFT_SIZE // 4
ZERO_PAD = 0
WINDOW = np.hanning(DFT_SIZE)
```

Part 1: Making a speech detector

In this section we will design a simple classifier that will let us know if its input is speech or non-speech. Download the data archive from: [https://drive.google.com/file/d/1Z8tj-HHQCvT54Mr20Dc1UfV48SPY449r/view] In this part we will use the dataset in data/SpeechMusic. In it you will find two directories, speech/ and music/ containing data from each class.

Randomly select 50 soundfiles from each directory to use as training data, and use the remaining sounds as testing data. For all of the sounds we will compute a representation that makes the classification easier and we will use a simple Gaussian model to classify them. Do the following:

- Perform an STFT for each sound, take it's magnitude and raise it to 0.3 to improve contrast
  - We will consider each spectral slice of that to be a data point
- Using the training data of each sound:
  - Calculate the mean column and the diagonal covariance of the columns
  - You will thus get two sets of Gaussian parameters that model each sound class
- For each testing data point:
  - Calculate the likelihood of each column based on the above models
  - To calculate the entire file likelihood add all the frame likelihoods
  - Assign each soundfile to the class that gets the highest likelihood

For extra credit implement the parameter estimation and model likelihood yourself. If you are too lazy for that you can instead use `sklearn.mixture.GaussianMixture` to learn a diagonal single-Gaussian model per class.

How do the results look like? If you rerun this with a different training/testing set, is there an appreciable difference? On average over multiple training/testing sets what accuracy do you get?

In [ ]:
```python
# Determine which files to use
path = "./data/SpeechMusic"
music_stfts = []
speech_stfts = []

for fname in os.listdir(path + "/music"):
    fs, data = wavfile.read(path + "/music/" + fname)

    # Perform an STFT for each sound, take it's magnitude and
    # raise it to 0.3 to improve contrast
    data_stft = stft(data, DFT_SIZE, HOP_SIZE, ZERO_PAD, WINDOW)
    music_stfts.append(np.abs(data_stft) ** 0.3)

for fname in os.listdir(path + "/speech"):
    fs, data = wavfile.read(path + "/speech/" + fname)

    # Perform an STFT for each sound, take it's magnitude and
    # raise it to 0.3 to improve contrast
    data_stft = stft(data, DFT_SIZE, HOP_SIZE, ZERO_PAD, WINDOW)
    speech_stfts.append(np.abs(data_stft) ** 0.3)
```

In [ ]:
```python
for i in range(5):
    random_state = random.randint(0, 100)
    speech_train, speech_test, music_train, music_test = tts(
        speech_stfts, music_stfts, train_size=50, random_state=random_state
    )

    temp = []
    for speech in speech_train:
        temp.extend(speech)
    speech_train = temp

    temp = []
    for music in music_train:
        temp.extend(music)
    music_train = temp

    speech_gm = GMM(5, covariance_type="diag").fit(speech_train)
    music_gm = GMM(5, covariance_type="diag").fit(music_train)

    speech_correct = sum(
        speech_gm.score(speech) > music_gm.score(speech) for speech in speech_test
    )
    music_correct = sum(
        speech_gm.score(music) < music_gm.score(music) for music in music_test
    )
    total_correct = speech_correct + music_correct
    accuracy = total_correct / (len(speech_test) + len(music_test))

    print(
        "Random State:",
        random_state,
        "Correct:",
        total_correct,
        "Test Size:",
        (len(speech_test) + len(music_test)),
        "Accuracy:",
        accuracy,
    )
```

```
Random State: 22 Correct: 18 Test Size: 20 Accuracy: 0.9
Random State: 76 Correct: 19 Test Size: 20 Accuracy: 0.95
Random State: 20 Correct: 18 Test Size: 20 Accuracy: 0.9
Random State: 61 Correct: 18 Test Size: 20 Accuracy: 0.9
Random State: 40 Correct: 20 Test Size: 20 Accuracy: 1.0
```

Conclusions

I ran the predictions around 15 times and the results were always in the range 0.85-0.95.

Part 2: Making a music genre classifier

We will repeat the above, but this time we will perform music genre classification. To do so we will use a slightly more elaborate feature representation, and a stronger classification model. If you downloaded the data archive pointed to above, you will find a subset of the CTZAN dataset in the data/genre folder, this is a benchmark data set for music genre classification.

Just as before, you will find a set of directories with examples of each sound class that we want to recognize. For each class, split the soundfiles into a training set (50% of data) and testing set (remaining 50% of data).

For a representation we will use MFCC features. For extra credit, code these yourself otherwise you can use the implementation from the `librosa` library. Once all the files are transformed we will have a series of MFCC frames for each recording (as opposed to spectral frames as is in the case of the STFT). We will use these as the data to classify.

For each class learn a Gaussian model (with a diagonal covariance again). This will be the same process as above. In order to evaluate how good this works we will use the following procedure. For each sound in the training data, get the likelihood of each MFCC frame based on the learned Gaussian models and sum these over the entire file just as we did before. Use the resulting values to get a classification result for each . Report how accurate your results are. Now report the accuracy using your testing data instead.

Now will use a better classifier to hopefully get better accuracy. We will use a Gaussian Mixture Model (`sklearn.mixture.GaussianMixture`). Just as before you should learn one such model for each class using the corresponding training data.

How many Gaussians do you need in your GMM to get the best results? Do the MFCC parameters make a difference? Play around with the numbers to get the best possible results. You should be able to get at least 70% accuracy on average.

In [ ]:
```python
def get_mfccs(dir):
    mfccs = []
    for fname in os.listdir("./data/genres/" + dir):
        data, fs = librosa.core.load("./data/genres/" + dir + "/" + fname)
        mfcc = np.array(
            librosa.feature.mfcc(y=np.array(data), sr=fs, n_mfcc=60))
        mfccs.append(mfcc)
    return mfccs


random_state = random.randint(0, 100)

classical_training, classical_testing = tts(get_mfccs('classical'),
                                            test_size=0.5,
                                            random_state=random_state)
disco_training, disco_testing = tts(get_mfccs('disco'),
                                    test_size=0.5,
                                    random_state=random_state)
metal_training, metal_testing = tts(get_mfccs('metal'),
                                    test_size=0.5,
                                    random_state=random_state)
pop_training, pop_testing = tts(get_mfccs('pop'),
                                test_size=0.5,
                                random_state=random_state)
reggae_training, reggae_testing = tts(get_mfccs('reggae'),
                                      test_size=0.5,
                                      random_state=random_state)

classical_training = np.concatenate(classical_training, axis=1)
disco_training = np.concatenate(disco_training, axis=1)
metal_training = np.concatenate(metal_training, axis=1)
pop_training = np.concatenate(pop_training, axis=1)
reggae_training = np.concatenate(reggae_training, axis=1)

gm_classical = GMM(n_components=10,
                   covariance_type="diag").fit(classical_training.T)
gm_disco = GMM(n_components=10, covariance_type="diag").fit(disco_training.T)
gm_metal = GMM(n_components=10, covariance_type="diag").fit(metal_training.T)
gm_pop = GMM(n_components=10, covariance_type="diag").fit(pop_training.T)
gm_reggae = GMM(n_components=10, covariance_type="diag").fit(reggae_training.T)


def check_accuracy(comp_1, comp_2, data):
    num_points = 0

    for datapoint in data:
        if comp_1.score(datapoint.T) > comp_2.score(datapoint.T):
            num_points += 1

    return num_points


testing_sets = [(gm_classical, gm_disco, classical_testing),
                (gm_classical, gm_metal, classical_testing),
                (gm_classical, gm_pop, classical_testing),
                (gm_classical, gm_reggae, classical_testing),
                (gm_disco, gm_classical, disco_testing),
                (gm_disco, gm_metal, disco_testing),
                (gm_disco, gm_pop, disco_testing),
                (gm_disco, gm_reggae, disco_testing),
                (gm_metal, gm_classical, metal_testing),
                (gm_metal, gm_disco, metal_testing),
                (gm_metal, gm_pop, metal_testing),
                (gm_metal, gm_reggae, metal_testing),
                (gm_pop, gm_classical, pop_testing),
                (gm_pop, gm_disco, pop_testing),
                (gm_pop, gm_metal, pop_testing),
                (gm_pop, gm_reggae, pop_testing),
                (gm_reggae, gm_classical, reggae_testing),
                (gm_reggae, gm_disco, reggae_testing),
                (gm_reggae, gm_metal, reggae_testing),
                (gm_reggae, gm_pop, reggae_testing)]

points = [check_accuracy(c1, c2, data) for (c1, c2, data) in testing_sets]

total_points_mfcc = sum(points)
accuracy = total_points_mfcc / (
    (len(classical_testing) + len(disco_testing) + len(metal_testing) +
     len(pop_testing) + len(reggae_testing)) * 5)
print("Accuracy:", round(accuracy, 2), "Random State:", random_state)
```

```
Accuracy: 0.75 Random State: 97
```

Part 3: Make it better (extra credit, required for 4-hour registrants)

There is no shortage of techniques (and free code) to use for classification. Revisit the two problems above and use any other type of classifier you want (Neural Nets, Boosting, Decision Trees, whatever). Also feel free to use any feature you want. Can you improve on the results you got before? How much higher can you get your accuracy for either case?

In [ ]:
```python
# YOUR CODE HERE
raise NotImplementedError()
```

```
---------------------------------------------------------------------------
NotImplementedError                       Traceback (most recent call last)
Cell In[6], line 2
      1 # YOUR CODE HERE
----> 2 raise NotImplementedError()

NotImplementedError:
```