This small python project is an attempt to detect a certain sound in an audio sample.

Software Requirements

Python 3+

Pythonic packages

pyaudio
matplotlib
numpy
scipy

Note: ensure they are for python 3

General notes and instructions

1. Let all your sample files have the word sample in their file names. For e.g

   `data/clap_`**`sample`**`_472ms_16bit_mono.wav`
2. It is also good to mention the length of audio clip, its bit depth, and channels.
3. Always record with 16 bit, 1 channel in WAV format
4. For test clips which contain any number of the sample sounds inside create the wav file with with **test_**. As before embed the length bit depth and channel in the file name.

Plotting the WAV file

This is only for inspection purposes. The graph plotter (matplotlib) will generate a wave form. If you want to read a y-value off it, just home the mouse pointer to that specific point and read it off the status bar.

Usage:

```
./plotwav.py <input_audio_file> Int16
```

`Int16` is mandatory here

Note: y-value noted here is only for the `splitwav2.py` script to properly capture short audio clips from the test clips.

Generating FFT data from WAV file

There are two important scripts:

`1. fftanalyze.py`

This is meant to be run on the sample audio clip.

The output csv file will be stored inside the **samples** directory. Files will be saved in format `data_fft_<input_audio_filename>.csv`

Usage:

```
./fftanalyze.py <input_audio> Int16
```

`Int16` is mandatory here

`2. batch_generate_wav_fft.py`

This is meant to run on audio files in the processed folder

Usage:

```
./batch_generate_wav_fft.py
```

This will create a series of csvfiles containing fft data for each audio clip in the **processed** directory

Splitting test audio file into audio clips

Usage:

```
./splitwav2.py <input_audio> Int16
```

`Int16` is mandatory here

This will output wav files in the **processed** folder. It detects peaks in the wav file, creates several clips. The assumption here is that the peak in the wav file might correspond to some sound we are interested in. We need to subject them to further analysis.

Since the script is not, at the moment, designed to accept all necessary values as arguments, some preparation is necessary before running this script.

Setting the threshold y-value

If the wav file amplitude is greater than this threshold capture the clip. The **plotwav.py** script will aide in determining this value.

Edit line 125 in file with this value

Setting the sample length

Usually half the length of the sample audio file. Can be determined using audacity. This is to ensure that the clip we are recording contains the sound we are interested in.

Edit line 170 in file with this value.

Generating the fft data for all audio clips in the processed folder

Simply run **./batch_generate_wav_fft.py**

Detecting if processed clips is a positive match

Script: batch_analyze.py

This script will process the csv files containing fft data of short audio clips in the **processed** folder and spit out a result.

Usage:

```
./batch_analyze.py <sample_fft_data_csv_file>
```

The output will look like

```
samples/data_fft_clap_sample_472ms_16bit_mono.csv
        Width: 4749.68746994
processed/data_fft_clap_9.721s_16bit_mono_24_03.csv
        Width: 5702.79626205
        Ratio: 1.20066768564
processed/data_fft_clap_9.721s_16bit_mono_05_01.csv
        Width: 5034.43040407
        Ratio: 1.05994982532
processed/data_fft_clap_9.721s_16bit_mono_15_02.csv
        Width: 4999.03587444
        Ratio: 1.05249785508
processed/data_fft_clap_9.721s_16bit_mono_32_04.csv
        Width: 3883.3493485
        Ratio: 1.2230904417
```

The hypothesis that that if the `ratio` is between `[1, 2)` , then that clip has a positive match.