ICS2203 – Assignment
Task 2

Single Word Speech Recogniser

Daniel Magro

484497M

January 2017

Table of Contents

Step 1 – Word List

This system was designed to recognise 5 words, all of which are names of instruments. The words chosen are the following:

- Bass
- Drum
- Guitar
- Piano
- Violin

Step 2 – Recording Training Audio

This step required recording an audio clip of myself repeatedly saying each word.

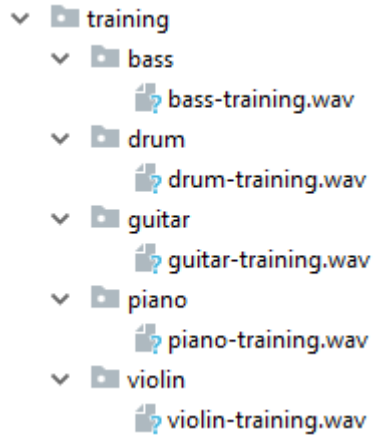
In order to achieve this, the Python file **II_RecordTraining.py** was created. This Python program defines a method named *wavrecord* which takes the sampling rate, duration of audio clip and output file name as parameters.

The main method of this Python file first asks the user which word from the word list he would like to record a training clip for, as follows:

```
Would you like to record training data for:
1) Bass
2) Drum
3) Guitar
4) Piano
5) Violin
```

The user can enter any number between 1 and 5, to choose the word for which he would like to record the training audio.

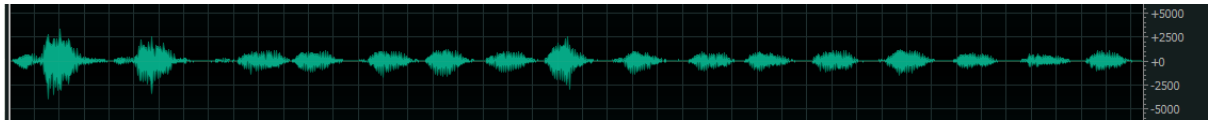
The 10 second training audio file is stored in its respective folder as follows:



The training audio is then opened in **ocenaudio** so that the audio can be cleaned by removing the silent parts of the sound file.

For example, the 10 second training audio for “bass” was changed from:

To this ~6 second clip:

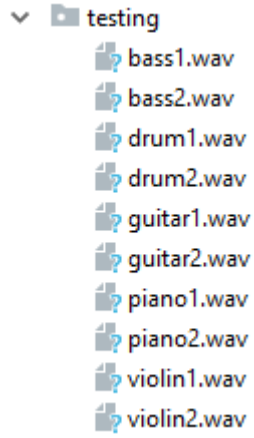


Step 3 – Recording Testing Audio

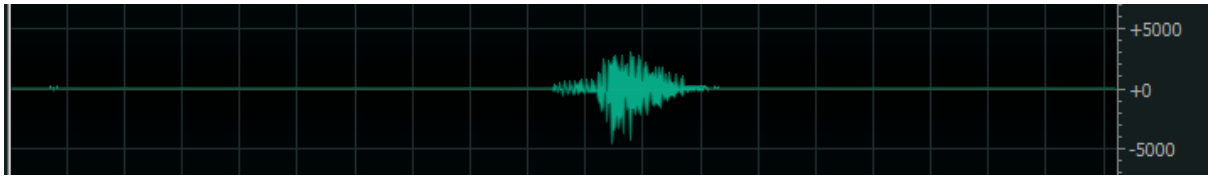
This step required recording an audio clip of myself saying one of the words from the word list. These recordings will be used to gauge how good the program is at recognising words.

The Python file **III_RecordTest.py** was created. This Python program again defines the *wavrecord* method.

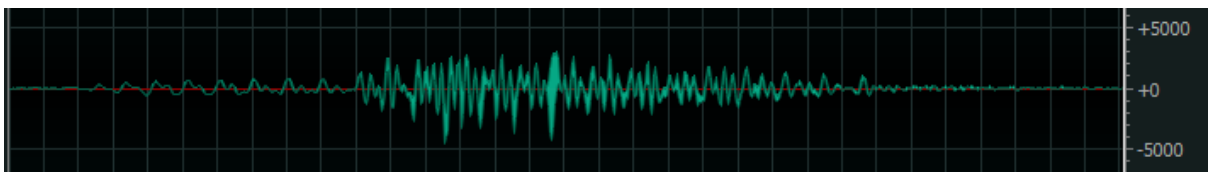
The main method of this Python file first asks the user what to name the output file. Then a 2 second clip is recorded and stored inside the **testing** folder, as follows:



Each clip is then opened in ocenaudio such that the silent parts may be removed. This 2 second audio clip of myself was turned from this:



To this 321ms audio clip:



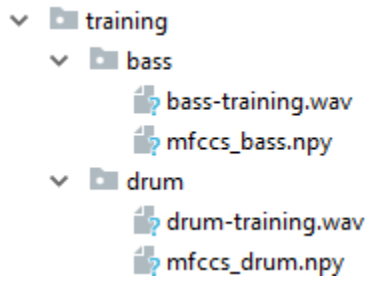
Step 4 – Feature Extraction

This step required first loading the recorded training data for each word from their respective folder. Then, the MFCCs were extracted from each audio file.
Finally, the extracted MFCCs were saved to disk, in the same folders as the training data.

The Python file **IV_ExtractMFCCs.py** was created. First, the training audio files for each word were loaded from disk using the **wavfile.read** function and specifying their path. For example the bass's training data was loaded using:
fs1, data1 = scipy.io.wavfile.read('training//bass//bass-training.wav')

Next, the **mfcc_extract** function was defined, which takes the signal and the sampling rate as parameters. The mfcc_extract function was called with each word's training audio, with the following command: mfccs_bass = mfcc_extract(signal=data1, fs=fs1)

Finally, once all the MFCCs were extracted, they were saved to disk, using **numpy.save**, in every word's respective training folder, as shown in the next screenshot, using:
numpy.save('training//bass//mfccs_bass.npy', mfccs_bass)



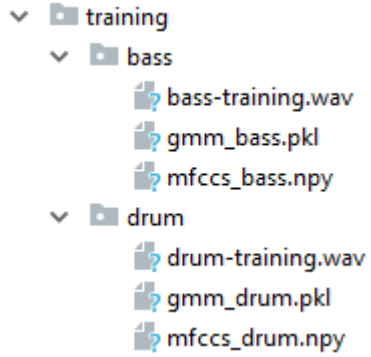
Step 5 – Building Word Acoustic Models

This step involved building a Gaussian Mixture Model (GMM) for each word, from each word's MFCCs, and then saving the GMM to disk.

The Python file **V_ConstructGMMs.py** was created. Firstly, the MFCCs for each word were loaded by using the **numpy.load** function and specifying their path. For example, the MFCCs for "bass" were loaded using: *mfccs_bass = numpy.load('training//bass//mfccs_bass.npy')*

Next, the **gmm_construct** function was defined, which takes the MFCCs and the number of components as parameters. The number of components used was 16. The gmm_construct was called with each word's MFCCs, using the command:
gmm_bass = gmm_construct(mfccs_bass, n_components=k)

Finally, after all the GMMs were constructed, they were saved to disk, using joblib.dump, in every word's respective training folder, as shown in the next screenshot, using:
joblib.dump(gmm_bass, 'training//bass//gmm_bass.pkl')



Step 6 – Testing

For this step, a program was written which goes through all the audio test files inside the "testing" folder and tries to recognise which word is being said in each file.

The Python file **VI_Testing** was created. First, The GMM for each word was loaded using the joblib.load function. For example, the GMM for "bass" was loaded using:
gmm_bass = joblib.load('training//bass//gmm_bass.pkl')

Next, a loop iterates over all the files inside the "testing" folder. If the file extension of the file is not *.wav*, then it is ignored, and the next file is considered. If the file extension is *.wav*, then the following process takes place:

First, the .wav file is loaded into memory using the **wavfile.read** function.
Then, the MFCCs are extracted from the test audio file using the **mfcc_extract** function.
Next, the MFCCs are scored under all the word models using **gmm_<word_n>.score(mfccs)**
eg: *gmm_bass.score(mfccs*).
The score for the MFCCs under each word model are stored inside a **scores** array.
Finally, depending on which element of the array has the highest score, the program recognises which word is being said in the test file.

The program managed to recognise 9/10 words. The only word which was not recognised is the second recording of "violin", which is a 2-syllable word.

Step 7 – Testing 2

For this step, the test audio files of another student using the same word list were copied into the "testing" folder, so as to be tested against the system. The names of the files were changed to the format *z<word>7_.wav*. The 'z' was inserted so that these audio files would group together at the end of the folder.

When the other student's test audio files were scored against my training data, only the word "violin" was recognised correctly, while the other 4 words were recognised incorrectly.

This inaccuracy is caused due to the fact that the training data of my system was gathered solely from my own voice, thus the model was trained to recognise my voice. The system has not just 'learnt' how the words are said, but rather how **I** say them.