Machine Learning : Recognition of Spoken Digits

Kanchan Kumari

1 Introduction

The Data set for this project consists recordings of spoken digits in .wav files at 8 kHz. The recordings are trimmed so that they have near minimal silence at the beginnings and ends. There are 6 different speakers who pronounce each digit from 0 to 9 for 50 times. Each audio file has a name of the form $\{digit\}\_\{speaker\ name\}\_\{attempt\ number\}.wav$. The data set with total 3000 audio files , is split into training and validation(test) sets with the ratio 80 : 20. The data labels are digits, from 0 to 9.

2 Objective

Our goal for this project is to classify what digit was spoken in a particular audio file using multiclass classification algorithms. We will train the model using our training data and validate it on Test data. Any Machine algorithm used should be Supervised machine learning algorithm, as we are provided with data labels.

3 Visualizing the Data

The libraries used for visualizing the audio data are - Matplotlib, IPython.display(ipd) , scipy.io.wavefile.

1. Listening to a few audio files using "ipd"

```
ipd.Audio('AudioMNIST/Audio/Training/0_lucas_46.wav')
```


```
ipd.Audio( 'AudioMNIST/Audio/Training/1_nicolas_46.wav')
```


```
ipd.Audio('AudioMNIST/Audio/Training/5_theo_46.wav')
```

2. Normalised energy graph of file "0-lucas-46.wav"

```
def normalize_audio(audio):
    audio = audio / np.max(np.abs(audio))
    return audio

sample_rate, audio = wavfile.read('AudioMNIST/Audio/Training/0_lucas_46.wav')

audio = normalize_audio(audio)
plt.figure(figsize=(15,4))
plt.plot(np.linspace(0, len(audio) / sample_rate, num=len(audio)), audio)
plt.grid(True)
```

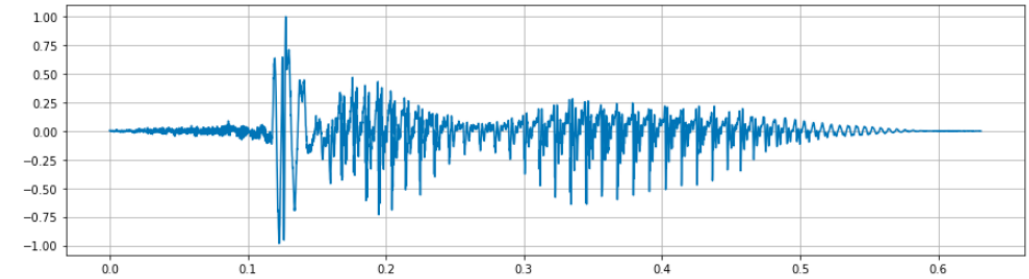

Figure 1: Audio Wave containing digit Zero.

3. Normalised energy graph of file "2-nicolas-46.wav"

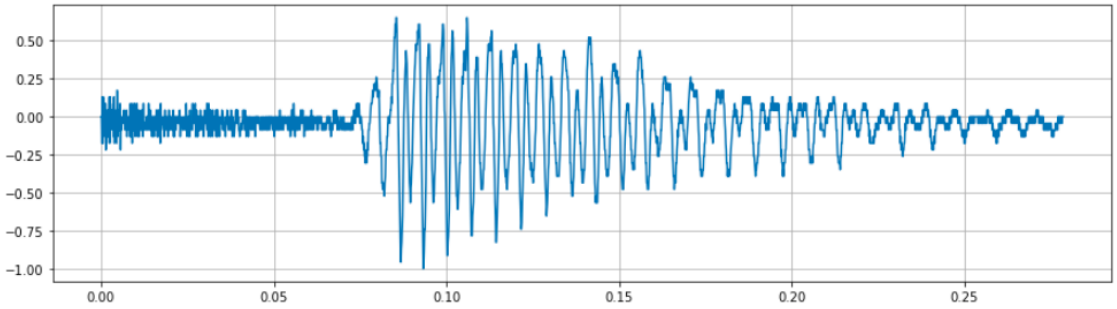

Figure 2: Audio Wave containing digit Two.

4. Normalised energy graph of file "3-theo-46.wav"

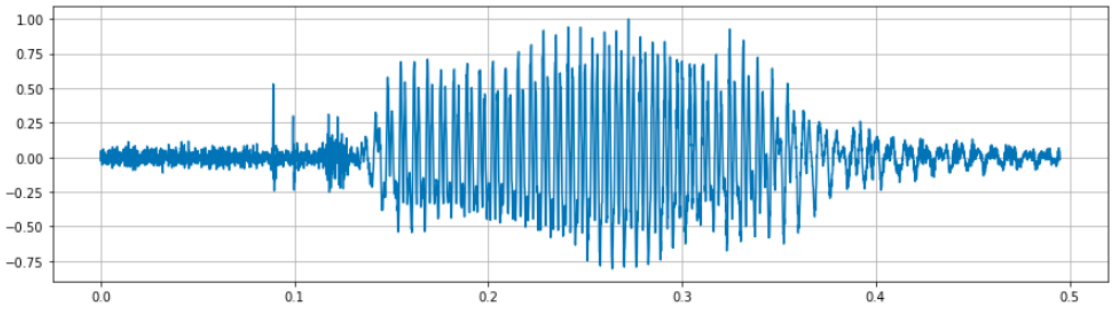

Figure 3: Audio Wave containing digit Three.

5. Normalised energy graph of file "7-ywewler-21.wav"

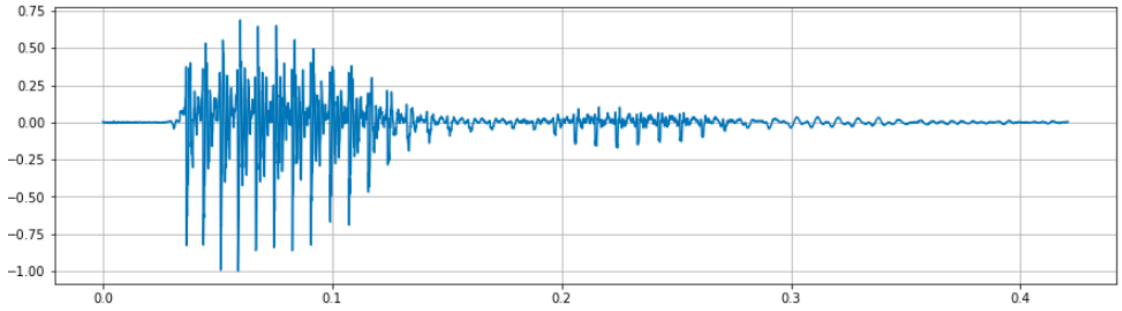

Figure 4: Audio Wave containing digit Seven.

6. Normalised energy graph of file "8-jackson-42.wav"

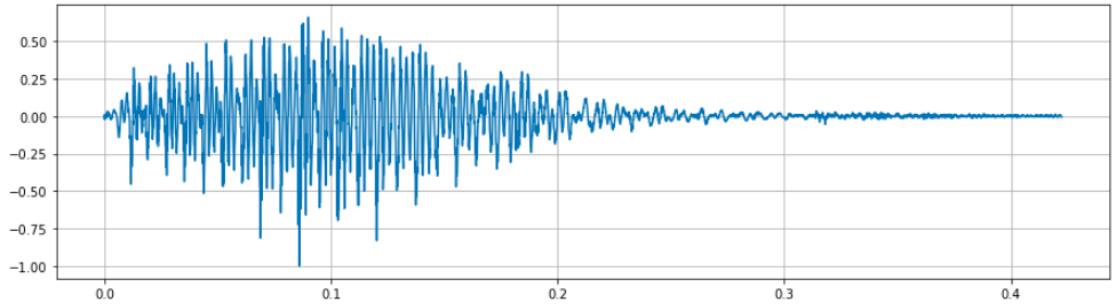

Figure 5: Audio Wave containing digit Eight.

Similarly, other files can also be visualized. The above files were picked randomly for visualization.

4 Extracting Features

Library used for audio feature extraction is "Librosa". We can extract Mel-Frequency Cepstral Coefficients (MFCC) using function "librosa.feature.mfcc".

The MFCC feature extraction technique includes windowing the signal, applying the DFT(Direct Fourier Transform), taking the log of the magnitude, and then warping the frequencies on a Mel scale, followed by applying the inverse DCT(discrete cosine transform). The MFCC coefficients contain information about the rate changes in the different spectrum bands. If a cepstral coefficient has a positive value, the majority of the spectral energy is concentrated in the low-frequency regions.

The following code can be used to extract features from each audio file and we can add them to a n-dimension Numpy array to create our training and testing data.

```
data, sr = librosa.load('AudioMNIST/Audio/Training/0_george_10.wav')
mfcc = librosa.feature.mfcc(y=data, sr=sr, S=None, n_mfcc = 900, dct_type = 2, norm = 'ortho')
```

In this project We are provided with 900 Mel-Frequency Cepstral Coefficients (MFCC) for each audio file, divided in Training and Test set. So we will use the following code to read them -

```
#For Training Data
audio_mnist_training_mfccs = np.genfromtxt(
    'AudioMNIST/MFCC/Training/training_mfccs.txt')
audio_mnist_training_labels = np.genfromtxt(
    'AudioMNIST/MFCC/Training/training_labels.txt').reshape(-1, 1)

#For Test Data

audio_mnist_testing_mfccs = np.genfromtxt(
    'AudioMNIST/MFCC/testing/testing_mfccs.txt')
audio_mnist_testing_labels = np.genfromtxt(
    'AudioMNIST/MFCC/testing/testing_labels.txt').reshape(-1, 1)
```

5 Training ML models

I train 2 machine learning models to classify to which class (0 to 9) a particular audio file belongs to. The approach is different for both models but classification accuracy achieved is same.

5.1 Multinomial Logistic Regression Classification Model

After going through pre-processing steps such as standardising the input data and One hot vector encoding of labels, we write the following code to achieve optimal weights for Multinomial Logistic Regression Classification Model. We run 5,000 iterations of the gradient descent procedure to find optimal weights (audio-optimal-weights).

```
audio_objective = lambda weights: multinomial_logistic_regression_cost_function(audio_data_matrix, weights, audio_OHV)

audio_gradient = lambda weights: multinomial_logistic_regression_gradient(audio_data_matrix, weights, audio_OHV)

audio_step_size = 3.9/(np.linalg.norm(audio_data_matrix))**2

initial_weight_matrix = np.zeros((audio_data_matrix.shape[1], audio_OHV.shape[1]))

audio_optimal_weights,audio_objective_values =gradient_descent(audio_objective,audio_gradient,initial_weight_matrix,
                                                               audio_step_size,no_of_iterations=5000,print_output=1000)
audio_accuracy_rate =classification_accuracy(multinomial_prediction_function(audio_data_matrix,audio_optimal_weights)
                                             ,audio_mnist_training_labels)
```

```
At  1000 iteration value of objective function is:  454.2979094294481
At  2000 iteration value of objective function is:  287.2082008750173
At  3000 iteration value of objective function is:  217.5317493559777
At  4000 iteration value of objective function is:  177.65860255561455
At  5000 iteration value of objective function is:  151.31060707815132
```

5.2 Multinomial Lasso logistic Regression Classification Model

We change objective function to lasso cost function and use Lasso regularisation parameter, which is equal to 1e-5. We also use proximal map (soft-thresholding) which is defined as -

$$\text{soft}_\tau(x) = x - \tau \cdot H'_\tau(x) = \{\, x - \text{sgn}(x) \cdot \tau, |x| \geq \tau, 0, |x| < \tau.$$

In this case we will run 5,000 iterations of the proximal gradient descent procedure to find optimal weights (audio-proximal-weights).

```
LASSO_regularisation_parameter = 1e-5

audio_step_size = 3.9/(np.linalg.norm(audio_data_matrix))**2

audio_threshold = LASSO_regularisation_parameter * audio_step_size

audio_objective = lambda weights: lasso_logistic_regression_cost_function(audio_data_matrix, weights, audio_OHV,
                                                                          LASSO_regularisation_parameter)

audio_gradient = lambda weights: multinomial_logistic_regression_gradient(audio_data_matrix, weights, audio_OHV)

audio_proximal_map = lambda weights : soft_thresholding(weights, audio_threshold)

initial_weight_matrix = np.zeros((audio_data_matrix.shape[1], audio_OHV.shape[1]))

audio_proximal_weights, audio_lasso_objective_values = proximal_gradient_descent(
    audio_objective,audio_gradient,audio_proximal_map,initial_weight_matrix, audio_step_size, 5000, 1000)
```

```
At 1000 iteration, value of obejective function is:  454.00397600881485
At 2000 iteration, value of obejective function is:  287.1133053237283
At 3000 iteration, value of obejective function is:  217.48358926575742
At 4000 iteration, value of obejective function is:  177.62943546406544
At 5000 iteration, value of obejective function is:  151.29121939943028
```

6 Conclusion

The Classification Accuracy achieved by both algorithms found to be same. For Training data, accuracy is **99.38%** and for Test data, it is **95.33%**. We can use either of the weights to predict labels for new data. I have used audio-optimal-weights(Obtained from model 1) for MNIST-model-function. There are many other ways to approach this problem such as using - ridge regression or neural networks. The algorithms used here are fastest and provide excellent classification accuracy.