Assignment 4: Automatic Speech Recognition

Pranav Sankhe— 150070009

19/11/2018

1 Question

Develop an end-pointer using speech/silence detection that enables the automatic segmentation of the individual digit utterances from the continuous audio record. Obtain the pre-emphasised signal corresponding to each utterance.

1.1 Answer

Pre-processing of Speech Signal serves various purposes in any speech processing application. It includes Noise Removal, Endpoint Detection, Pre-emphasis, Framing, Windowing, Echo Canceling etc. Out of these, silence/unvoiced portion removal along with endpoint detection is the fundamental step for applications like Speech and Speaker Recognition.
Endpoint detection is used to remove the DC offset value from the signal after silence removal process. Silence removal and Endpoint detection are main part of many applications such as speaker and speech recognition.
Conventional methods using the Zero Crossing Rate (ZCR) and Short Time Energy (STE) have been implemented here. There exist novel methods like the one that uses Probability Density Function (PDF) of the background noise and a Linear Pattern Classifier for classification of Voiced part of a speech from silence/unvoiced part which do indeed perform better than the convention STE method which has been implemented here.

The Short Time Energy (STE) has been calculated and thresholded. We detect speech activity at points where the energy crosses the threshold, and where the energy is below the threshold is labeled as silence. The threshold was decided emperically and the estimation of the threshold by eyeballing was easy since the noise was extremely low.
Later sounds corresponding to different digits were passed through a pre-emphas is filter to take care of lip radiation.

Here is the plot of STE vs time for a particular utterance.

Pre-emphais filter has been applied on each of the extracted segment. A pre-emphasis filter is useful in several ways:

- balance the frequency spectrum since high frequencies usually have smaller magnitudes compared to lower frequencies,
- avoid numerical problems during the Fourier transform operation and

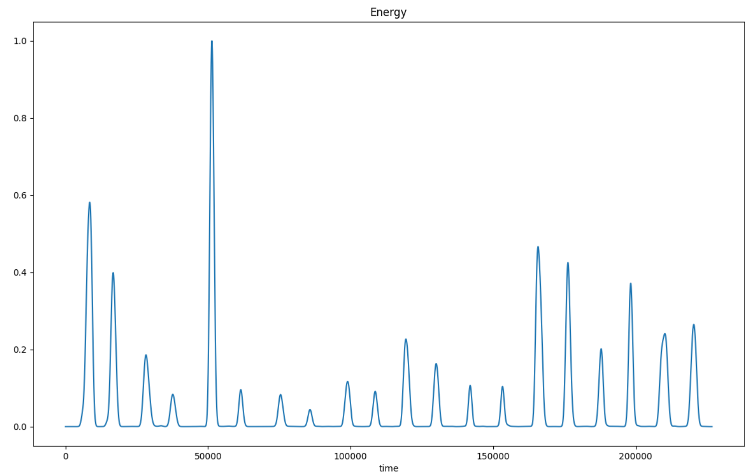
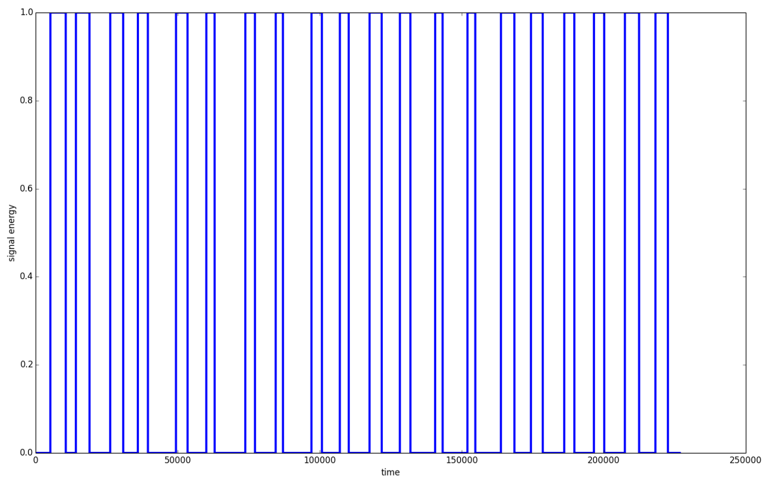


Figure 1: Windowed Energy



s

Figure 2: Endpoints

- may also improve the Signal-to-Noise Ratio (SNR).
- Equation: $y[t] = x[t] - \alpha x[t-1]$

Code for segmenting the audio files:

```
def pre_emphasis(input_signal):
    pre_emphasis_alpha = params.pre_emphasis_alpha
    pre_emphasized_signal = np.append(input_signal[0], input_signal[1:] - pre_emphasis_alpha * input_signal[:-1])
    return pre_emphasized_signal


def get_limiting_indices(y):
    y = y/np.max(y)
    energy_threshold = params.energy_threshold
    window_len = 3500

    window = np.hamming(window_len)
    sig_energy = np.convolve(y**2,window**2,'same')


    sig_energy = sig_energy/max(sig_energy)     #Normalize energy
    sig_energy_thresh = (sig_energy > energy_threshold).astype('float')
    import pdb; pdb.set_trace()
    #convert the bar graph to impulses by subtracting signal from it's shifted version
    indices = np.nonzero(abs(sig_energy_thresh[1:] - sig_energy_thresh[0:-1]))[0]

    start_indices = [indices[2*i] for i in range(len(indices)/2)]
    end_indices   = [indices[2*i+1] for i in range(len(indices)/2)]

    return start_indices, end_indices





digits = params.digits
male_files = np.sort(male_files)
male_names = np.sort(male_names)
female_files = np.sort(female_files)
female_names = np.sort(female_names)

for i in range(len(male_names)):
    print('Segmenting audio files of ' + male_names[i])
    y, sr = librosa.load( male_dir + '/' + male_files[i], sr=None)
    start_indices, end_indices = get_limiting_indices(y)

    if (not os.path.isdir("male_segmented/" + male_names[i])):
        os.mkdir("male_segmented/" + male_names[i])

    for p in range(len(end_indices)):
        sig = y[start_indices[p] : end_indices[p]]
        digit = pre_emphasis(sig)

        write("male_segmented/" + male_names[i] + '/' + digits[int(np.floor(p/2))] +'_' + str(p%2 + 1) +'.wav', sr, digit)

for i in range(len(female_names)):
    print('Segmenting audio files of ' + female_names[i])
    y, sr = librosa.load( female_dir + '/' + female_files[i], sr=None)
    start_indices, end_indices = get_limiting_indices(y)

```

```
    if (not os.path.isdir("female_segmented/" + female_names[i])):
        os.mkdir("female_segmented/" + female_names[i])

    for p in range(len(end_indices)):
        sig = y[start_indices[p] : end_indices[p]]
        digit = pre_emphasis(sig)

        write("female_segmented/" + female_names[i] + '/' + digits[
    int(np.floor(p/2))] +'_' + str(p%2 + 1) +'.wav', sr, digit)
```

2 Question

Develop a feature extractor that computes an MFCC feature vector for every 10 ms frame of an utterance.

2.1 Answer

Mel-Frequency Cepstral Coefficients (MFCCs) were very popular features for a long time for ASR systems. In a nutshell, a signal goes through a pre-emphasis filter; then gets sliced into (overlapping) frames and a window function is applied to each frame; afterwards, we do a Fourier transform on each frame (or more specifically a Short-Time Fourier Transform) and calculate the power spectrum; and subsequently compute the filter banks. To obtain MFCCs, a Discrete Cosine Transform (DCT) is applied to the filter banks retaining a number of the resulting coefficients while the rest are discarded. MFCC features were calculated for every 10ms frame of an utterance for all the digits and all the speakers. MEL filters are uniform in MEL scale which is coherent with the human hearing perception.

Visualize the MFCC series:

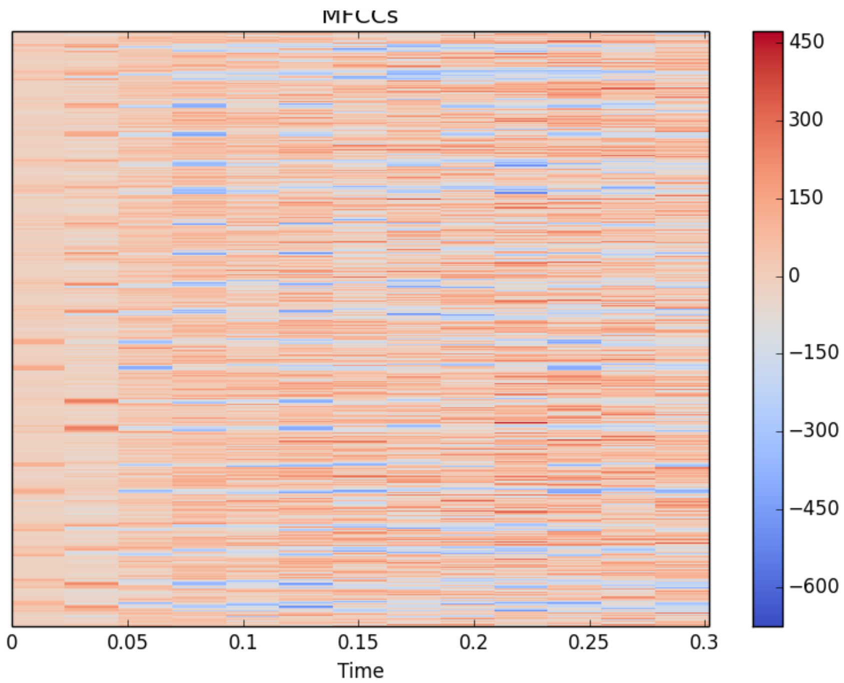


Figure 3: MFCCs

Code for extracting MFCC feature vector:

```python
import numpy as np
import librosa
import params
import scipy
import librosa.display
import matplotlib.pyplot as plt

frame_size = params.frame_size
frame_stride = params.frame_stride
num_ceps = params.num_ceps
def framing(input_signal, sample_rate):
    frame_length = frame_size * sample_rate
    frame_step = frame_stride * sample_rate
    signal_length = len(input_signal)
    frame_length = int(round(frame_length))
    frame_step = int(round(frame_step))
    num_frames = int(np.ceil(float(np.abs(signal_length - frame_length)) / frame_step))  # Make sure that we have at least 1 frame

    pad_signal_length = num_frames * frame_step + frame_length
    z = np.zeros((pad_signal_length - signal_length))
    pad_signal = np.append(input_signal, z) # Pad Signal to make sure that all frames have equal number of samples without truncating any samples from the original signal

    indices = np.tile(np.arange(0, frame_length), (num_frames, 1)) + np.tile(np.arange(0, num_frames * frame_step, frame_step), (frame_length, 1)).T
    frames = pad_signal[indices.astype(np.int32, copy=False)]
    frames = frames * np.hamming(frame_length)

    return frames


def frame_wise_fft(frames):
    fft_length = params.fft_length
    mag_frames = np.abs(np.fft.rfft(frames, fft_length))
    pow_frames = ((1.0 / fft_length) * ((mag_frames) ** 2))   # frame wise power spectrum

    return pow_frames

def filter_banks(frames, sample_rate):
    '''
    We can convert between Hertz (f) and Mel (m) using the following equation
    m = 2595* log10(1 + f * 700)
    '''
    fft_length = params.fft_length
    num_filters = params.num_filters


    low_freq_mel = 0
    high_freq_mel = (2595 * np.log10(1 + (sample_rate / 2) / 700))  # Convert Hz to Mel
    mel_points = np.linspace(low_freq_mel, high_freq_mel, num_filters + 2)  # Equally spaced in Mel scale
    hz_points = (700 * (10**(mel_points / 2595) - 1))  # Convert Mel to Hz
    bin = np.floor((fft_length + 1) * hz_points / sample_rate)
```

```

    fbank = np.zeros((num_filters, int(np.floor(fft_length / 2 + 1))))
    for m in range(1, num_filters + 1):
        f_m_minus = int(bin[m - 1])    # left
        f_m = int(bin[m])              # center
        f_m_plus = int(bin[m + 1])     # right

        for k in range(f_m_minus, f_m):
            fbank[m - 1, k] = (k - bin[m - 1]) / (bin[m] - bin[m - 1])
        for k in range(f_m, f_m_plus):
            fbank[m - 1, k] = (bin[m + 1] - k) / (bin[m + 1] - bin[m])
    filter_banks = np.dot(frames, fbank.T)
    filter_banks = np.where(filter_banks == 0, np.finfo(float).eps, filter_banks)  # Numerical Stability
    filter_banks = 20 * np.log10(filter_banks)  # dB
    return filter_banks


def mfcc(filter_banks, num_ceps):
    cep_lifter = 22
    mfcc = scipy.fftpack.dct(filter_banks, type=2, axis=1, norm='ortho')[:, 1 : (num_ceps + 1)] # Keep 2-13


    (nframes, ncoeff) = mfcc.shape
    n = np.arange(ncoeff)
    lift = 1 + (cep_lifter / 2) * np.sin(np.pi * n / cep_lifter)
    mfcc *= lift

    filter_banks -= (np.mean(filter_banks, axis=0) + 1e-8)


    mfcc -= (np.mean(mfcc, axis=0) + 1e-8)

    return mfcc


def main(filepath):
    input_signal, sample_rate = librosa.load(filepath, sr=None)
    frames = framing(input_signal, sample_rate)
    pow_frames = frame_wise_fft(frames)
    filter_banks_frames = filter_banks(pow_frames, sample_rate)
    mfcc_coeffs = mfcc(filter_banks_frames, num_ceps)
    # librosa.display.specshow(mfcc_coeffs, x_axis='time')
    # plt.colorbar()
    # plt.tight_layout()
    # plt.show()

    return np.transpose(mfcc_coeffs)
```

3 Question

Develop a digit recognizer based on the “bag of frames” approach with a codebook for each digit created out of training set speakers’ data. Provide the achieved word error rate (WER) in terms of % words incorrectly detected in the N-fold CV testing using a VQ codebook for each digit obtained via K-means clustering. Provide the achieved WER for with different numbers of clusters (e.g. 4, 8, 16, 64). Observe the common confusions, and comment on your results.

3.1 Answer

The MFCC feature vectors extracted for frames in the train set are used as a bag of frames for the recognition. We run a nearest neighbor classifier for an incoming frame MFCC vector. The output of the classifier for a given word is given by the majority vote of frame classification results.
The general N-fold cross validation technique is used to evaluate performance i.e. model is trained on N-1 speakers and tested on 1 left out speaker, which is iterated over for all N speakers. For every frame in the test pattern, minimum distortions from all reference vectors (codebooks of digits) are calculated and the digit which minimizes this distortion is predicted as an output.

If we include all the vectors of a given digit uttered by all the speakers, the task of predicting the correct digit becomes computationaly inefficient. Hence we will employ vector quantization which reduces all the utterances by fewwer vectors which we call as 'representative' vectors. We use Kmeans to get these 'representative' vectors. In the terminology of Kmeans, these representative vectors are the centres/centroids of the clusters.

Here’s a brief overview of k-means algorithm:

- Generate random k points from data as initial estimate of centroids
- Assign each vector to the cluster whose centroid yields the least distance
- Update the centroids of the clusters
- Repeat step 2 and step 3 until convergence

The accuracy is affected by number of clusters (which will be discussed in results section) We get the following WER for different number of clusters considered:

- WER for 4 clusters = 0.29
- WER for 8 clusters = 0.23
- WER for 16 clusters = 0.17
- WER for 64 clusters = 0.128

It is evident that as the number of clusters increases, the WER decreases which is something we can indeed expect since more number of clusters will be able to explain data better and hence less WER. There is, a trade-off between the WER and the number of clusters we can permit. The computational complexity of the system while testing increases with the number of clusters, but the word error rate decreases. We cannot expect very good WERs from either BOF or VQ directly, because they do not capture phoneme-level information. Two words may have the same phoneme, and a classifier like the one we built will get confused between the two.

The common confusions are represented in a confusion matrix, whose image is shown below:

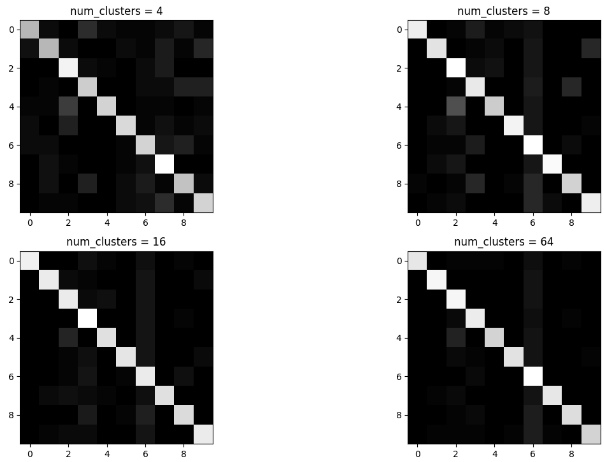


Figure 4: MFCCs

The row index in the confusion matrix indicates the ground truth, that is, the actual identity of the test utterance. The column index indicates the classification output.
1 is added to confusion matrix index (x+1, y+1) every time digit x is classified as y, thus the common classification outputs are brighter than the rest. Each row is scaled such that the maximum in the row is white. This indicates that the most common confusions were

- 3 confused as 8
- 4 confused as 2
- 8 confused as 6

Code for bag of frames approach:

```
import os
import scipy
import numpy as np
from scipy import signal
import matplotlib.pyplot as plt
from scipy import spatial
from scipy.cluster.vq import vq,kmeans
import params
import feature_extractor

sample_rate = params.sample_rate

digits = ['zero','one','two','three','four','five','six','seven','eight','nine']

seg_male_dir = params.seg_male_dir
seg_female_dir = params.seg_female_dir

male_speakers = os.listdir(seg_male_dir)
female_speakers = os.listdir(seg_female_dir)

all_speakers = male_speakers + female_speakers
all_speakers.remove('.DS_Store')
codeBook = {}
n_frame_dict = {}

for digit in digits:
    print("Preparing codebook for digit", digit)
    codeBook[digit] = {}
    n_frame_dict[digit] = {}

    for speaker in all_speakers:
        print("speaker = ", speaker)
        parent_dir = ''
        if speaker in male_speakers:
            parent_dir = seg_male_dir

        if speaker in female_speakers:
            parent_dir = seg_female_dir
        codeBook[digit][speaker] = []
        n_frame_dict[digit][speaker] = []
        for iteration in range(1,3):
            file = parent_dir + '/' + speaker + '/' + str(digit) + '_' + str(iteration) + '.wav'
            feature_mat = feature_extractor.main(file)
            n_frame_dict[digit][speaker].append(feature_mat.shape[1])
            for i in range(feature_mat.shape[1]):
                codeBook[digit][speaker].append(feature_mat[:,i])

print('Codebook created')
confusion_matrix = np.zeros([10,10])


VQCodeBook = {}
centroids = {}
n_clusters = params.n_clusters


for test_speaker in all_speakers:
    train_speakers = all_speakers
```

```
    train_speakers.remove(test_speaker)

    for digit in digits:
        VQCodeBook[digit] = []
        centroids[digit] = []
        for speaker in train_speakers:
            mat = np.asarray(codeBook[digit][speaker])
            if(VQCodeBook[digit] == []):
                VQCodeBook[digit] = mat
            else:
                VQCodeBook[digit] = np.concatenate([VQCodeBook[digit],mat],0)

        centroids[digit] = (kmeans(VQCodeBook[digit],n_clusters))[0]

    for test_digit in digits:
        for utterance in range(1,3):

            n_frames = n_frame_dict[test_digit][test_speaker]
            test_mat = codeBook[test_digit][test_speaker][sum(n_frames[0:(utterance-1)]):sum(n_frames[0:utterance])]

            sum_dist = np.zeros(10)
            for digit in digits:
                for l in range(len(test_mat)):
                    test_vec = np.asarray(test_mat)[l,:]
                    temp_list = np.asarray(centroids[digit])
                    min_dist,index = spatial.KDTree(temp_list).query(test_vec)
                    sum_dist[digits.index(digit)]+=min_dist

            pred_digit = np.argmin(sum_dist)
            print( "For ", test_speaker, " predicted digit = ", pred_digit, " ground truth = ", test_digit)
            confusion_matrix[digits.index(test_digit),pred_digit] += 1.0


    np.save('VQ_confusion_matrix_n_cluster'+str(n_clusters), confusion_matrix)


wer = 1 - np.trace(confusion_matrix)/(len(all_speakers)*20)
print wer
```

4 Question

Develop a template-matching digit recognizer based on DTW alignment and distance computation. Provide the achieved WER in N-fold CV evaluation. Observe the common confusions, and comment on your results.

4.1 Answer

In this approach, we align frames after doing a non-linear time warp. This takes into account some kind of sequence information about the utterances, and hence is expected to perform better than the previous methods and indeed it does. With this approach we get a WER of 6.25%. The common confusions are represented in a confusion matrix, whose image is shown below:

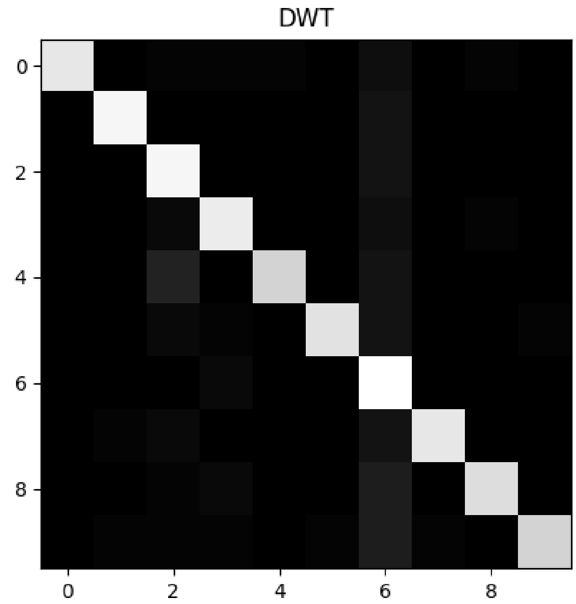


Figure 5: MFCCs

As earlier, the row index in the confusion matrix indicates the ground truth, that is, the actual identity of the test utterance. The column index indicates the classification output. 1 is added to confusion matrix index (x+1, y+1) every time digit x is classified as y, thus the common classification outputs are brighter than the rest. Each row is scaled such that the maximum in the row is white. This indicates that the most common confusions were

- 2 confused as 4
- 8 confused as 6

We notice that the confusion matrix in this case is much cleaner than the previous approaches. The WER is also much better. This is because this approach is more principled in capturing phoneme timing information. The previous two methods get confused between similar phonemes, but this method uses sequence information to disambiguate word identity.

Code for bag of DWT approach:

```
import os
import scipy
import numpy as np
from scipy import signal
import matplotlib.pyplot as plt
from scipy.cluster.vq import vq,kmeans
import params
import feature_extractor


def find_dtw_distance(test_pattern,ref_pattern):
    n = test_pattern.shape[1]
    m = ref_pattern.shape[1]

    distMat = np.zeros([n, m])

    for i in range(n):
        for j in range(m):

            distMat[i, j] = np.linalg.norm(np.subtract(test_pattern[:, i], ref_pattern[:, j]))

    DTW = np.zeros([n+1, m+1])

    for i in range(1,n+1):
        DTW[i, 0] = float('Inf')

    for i in range(1,m+1):
        DTW[0, i] = float('Inf')

    DTW[0,0] = 0

    for i in range(1,n+1):
        for j in range(1,m+1):
            cost = distMat[i-1, j-1]
            DTW[i, j] = cost + np.min([DTW[i-1, j], np.min([DTW[i-1, j-1], DTW[i, j-1]])])

    return DTW[n, m]


sample_rate = params.sample_rate

digits = ['zero','one','two','three','four','five','six','seven','eight','nine']


seg_male_dir = params.seg_male_dir
seg_female_dir = params.seg_female_dir

male_speakers = os.listdir(seg_male_dir)
female_speakers = os.listdir(seg_female_dir)
all_speakers = male_speakers + female_speakers
all_speakers.remove('.DS_Store')
codeBook = {}
n_frame_dict = {}



for digit in digits:
    print("Preparing codebook for digit", digit)
```

```
    codeBook[digit] = {}
    n_frame_dict[digit] = {}

    for speaker in all_speakers:
        print("speaker = ", speaker)
        parent_dir = ''
        if speaker in male_speakers:
            parent_dir = seg_male_dir

        if speaker in female_speakers:
            parent_dir = seg_female_dir
        codeBook[digit][speaker] = []
        n_frame_dict[digit][speaker] = []
        for iteration in range(1,3):
            file = parent_dir + '/' + speaker + '/' + str(digit) + '_' + str(iteration) + '.wav'
            feature_mat = feature_extractor.main(file)
            n_frame_dict[digit][speaker].append(feature_mat.shape[1])
            for i in range(feature_mat.shape[1]):
                codeBook[digit][speaker].append(feature_mat[:,i])

print('Codebook created')

confusion_matrix = np.zeros([10,10])

for test_speaker in all_speakers:
    train_speakers = all_speakers
    train_speakers.remove(test_speaker)

    for test_digit in digits:
        for utterance in range(1,3):

            n_frames = n_frame_dict[test_digit][test_speaker]
            test_mat = codeBook[test_digit][test_speaker][sum(n_frames[0:(utterance-1)]):sum(n_frames[0:utterance])]

            sum_dist = np.zeros(10)
            min_dist = float('Inf')
            for digit in digits:

                for speaker in train_speakers:
                    for ref_utterance in range(1,3):
                        n_ref_frames = n_frame_dict[digit][speaker]
                        ref_mat = codeBook[digit][speaker][sum(n_ref_frames[0:(ref_utterance-1)]):sum(n_ref_frames[0:ref_utterance])]

                        test_pattern = np.transpose(np.asarray(test_mat))
                        ref_pattern = np.transpose(np.asarray(ref_mat))
                        if np.sum(ref_pattern.shape) == 0:
                            continue
                        curr_dist = find_dtw_distance(test_pattern, ref_pattern)
                        if(curr_dist < min_dist):
                            min_dist = curr_dist
                            pred_digit = digits.index(digit)

            print( "For ", test_speaker, " predicted digit = ", pred_digit, " ground truth = ", test_digit)
```

```
            confusion_matrix[digits.index(test_digit),pred_digit]
    += 1

    np.save('DTW_confusion_matrix',confusion_matrix)

wer = 1 - np.trace(confusion_matrix)/(len(all_speakers)*20)
print wer
```