EE679: Assignment 4

Kalpesh Patil (130040019)

November 6, 2016

Abstract

The aim of this assignment is to develope a spoken digit recognition system. The input is given as sequence of utterences of 10 digits (0 to 9) by various male and female speakers sampled at 8kHz. First of all automated segmentation of digits from entire audio signal is carried out. MFCC features are used for recognition. Different approaches like 'Bag of frames' with all training vectors and with 'Vector Quantization Codebook' , 'Dynamic Time Warping' are implemented. Leave-one-out error (N-fold cross validation) is used as a performance measure. Word Error Rate and Confusion matrix are computed to study confusions causing recognition errors. Major code snippets of python implementation ar appended at the end..

Q1 End-pointer detection

Input was given as a sequence of 0-9 each digit uttered twice by every user. We had to discard the silent zone in between the digits and crop out the sound corresponding to digits. Thresholding smoothened windowed energy specturum gives us require boundaries.

Energy of the signal is convolved with a hamming window. Then an empirical threshold value is selected in order to get segments corresponding to all the digits in the signal. Smoothening is required to remove the spurious peaks occuring in the energy spectrum (which may be wrongly detected in while thresholding). Later sounds corresponding to different digits were passed through a pre-emphasis filter to take care of lip radiation.

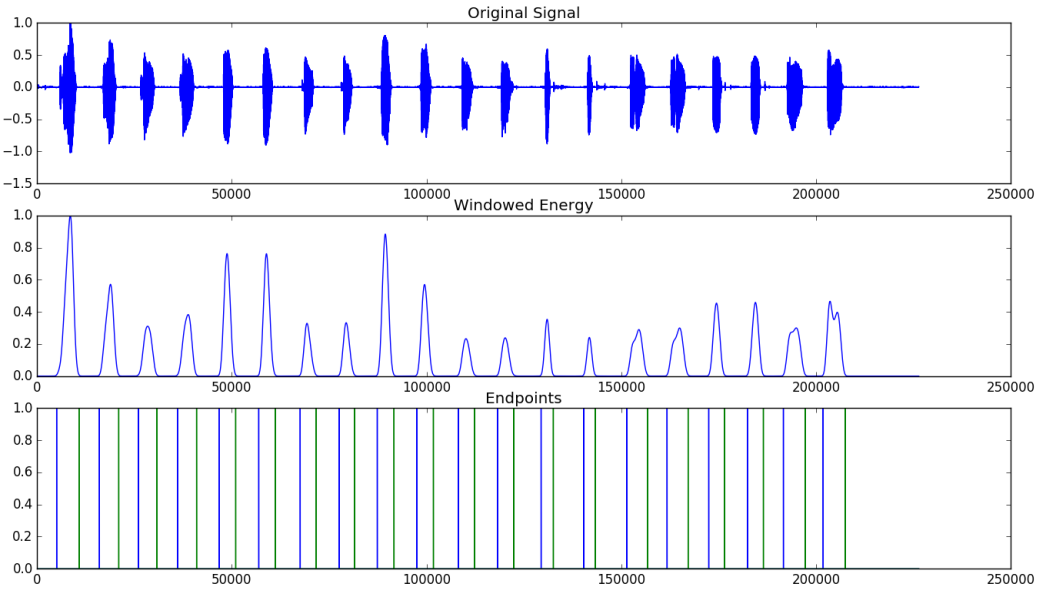

Figure 1: Sample image for segmentation

Q2 Feature Extractor

MFCC features were calculated for every 10ms frame of an utterance for all the digits and all the speakers. Following steps were followed:

- Framing: Signal was divided into overlaping frames with length equals to 25ms and hop equals to 10ms
- Sliding Window Spectral Estimate: For each of the frames, windowed power spectral estimate was computed by multiplying signal with a hamming window and taking magnitude square of DFT of the resultant signal. MEL coefficients

$$Mel(f) = 1125ln(1 + f/700)$$

$$Mel^{-1}(m) = 700(exp(m/1125) - 1)$$

The following figure shows MEL filter bank. MEL filters are uniform in MEL scale which is coherent with the human hearing perception. Parameters like f_{lower}, f_{upper} and number of filters were selected empirically. The follwing figure shows the filterbank used in the computation

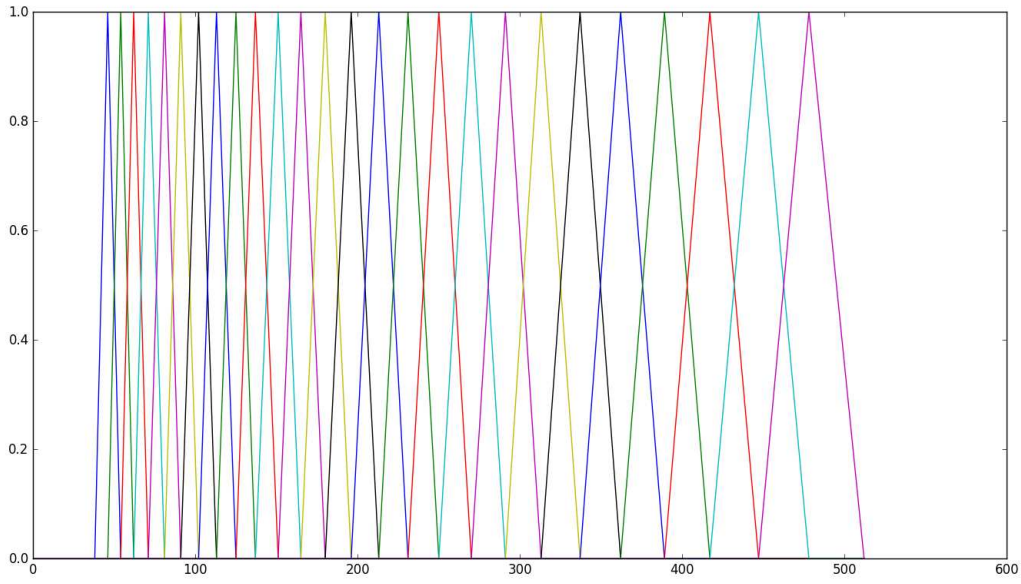

Figure 2: MEL filterbank

The Spectrum is passed through these filters and one coefficient for each filter as sum of product of corresponding values of spectrum and MEL

filter; reffered as MEL coefficients here onwards

$$Y_m = \sum W_m[k]S[k]$$

where $W_m$is m^{th}MEL filter and S is windowed spectrum of input signal

- Cepstral: $20log(Y_m)$ is computed for all filter coefficients. DCT of these coefficients is taken to obtain MFCC coefficients. (Other variant involves IDFT instead of DCT)
- Keep 13 coefficients and discard later coefficients
- Delta and delta-delta coefficients:

$$\triangle_i [n] = \frac{c_i[n+1] - c_i[n-1]}{2}$$

$$\triangle\triangle_i [n] = \frac{\triangle_i [n+1] - \triangle_i [n-1]}{2}$$

Delta and delta-delta coefficients are appended to the original feature vector to give information about spectral dynamics to the fileter explicitly.

Thus finally we obtain a 39 dimensional vector for a given frame

Q3 Bag of Frames approach

A large database containing MFCC features for every frame of every utterance of ever digit of every speaker is created using dictionary structure in python for easier data accessibility. 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 repeated for all N speakers.

Training vectors directly For every frame in the test pattern, minimum distortions from all reference vectors (codebooks of digits). The digit which eventually minimizes the sum of all such distortions is predicted as an output

$$\widehat{digit} = \min_{k} \sum_{j} d_{jk}$$

where $d_{jk}$is the minimum distortion of j^{th}test vector w.r.t. k^{th}codebook (digit). Here the codebook includes all the vectors of the given digit uttered by all the speakers. Hence computationally it is very inefficient. We will see in the results that a bit modification in the form of Vector Quantization reduces computational complexity drastically and preoduces appreciable results.

Vector Quantization Instead of having every possible vector in codebook, we include only few 'representative' vectors. These representative vectors are beasically the centres of the clusters. These clustering is done using k-means algorithm. The brief overview of k-means algorithm is as follows:

- 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

In case of speec recognition k-means performs pretty well. The accuracy is affected by number of clusters (which will be discussed in results section)

Results

Word Error Rate Following table summarises WER values for both the methods discussed above

Method	Word Error Rate
Training vectors directly (BOF)	0.1578
VQ (2 clusters)	0.3234
VQ (4 clusters)	0.2484
VQ (8 clusters)	0.1891
VQ (16 clusters)	0.1562

Table 1: WER for Q3

As we can observe that as the number of clusters increases, WER decreses. This signifies that more number of clusters can explain data better and hence less WER. In case of first method where vectors are trained directly, we observe that WER is lesser compared to that of VQ, becasue we are using all possible vectors instead of just few representative ones. VQ is computationally very simple compared to bag of frames (training on all vectors), yet it produces remarkably good results which are comparable with the ones produced by earlier method (bag of frames).

Test\Predicted	0	1	2	3	4	5	6	7	8	9
0	0.984	0.0	0.0	0.0	0.016	0.0	0.0	0.0	0.0	0.0
1	0.062	0.906	0.0	0.0	0.016	0.016	0.0	0.0	0.0	0.0
2	0.047	0.047	0.812	0.0	0.094	0.0	0.0	0.0	0.0	0.0
3	0.125	0.0	0.0	0.672	0.0	0.016	0.016	0.0	0.172	0.0
4	0.188	0.047	0.156	0.0	0.609	0.0	0.0	0.0	0.0	0.0
5	0.031	0.0	0.0	0.0	0.0	0.969	0.0	0.0	0.0	0.0
6	0.062	0.0	0.0	0.078	0.016	0.047	0.797	0.0	0.0	0.0
7	0.0	0.0	0.0	0.0	0.016	0.0	0.0	0.891	0.0	0.094
8	0.016	0.0	0.0	0.156	0.0	0.016	0.0	0.0	0.812	0.0
9	0.031	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.969

Table 2: Confusion Matrix for training vectors directly (BOF)

Test\Predicted	0	1	2	3	4	5	6	7	8	9
0	0.953	0.0	0.016	0.0	0.0	0.031	0.0	0.0	0.0	0.0
1	0.031	0.797	0.109	0.0	0.047	0.0	0.0	0.0	0.0	0.016
2	0.0	0.047	0.875	0.0	0.078	0.0	0.0	0.0	0.0	0.0
3	0.016	0.0	0.0	0.484	0.0	0.031	0.062	0.0	0.406	0.0
4	0.0	0.078	0.188	0.0	0.734	0.0	0.0	0.0	0.0	0.0
5	0.156	0.078	0.0	0.016	0.0	0.594	0.016	0.0	0.0	0.141
6	0.0	0.0	0.0	0.109	0.0	0.047	0.719	0.0	0.125	0.0
7	0.0	0.0	0.0	0.0	0.0	0.047	0.016	0.891	0.0	0.047
8	0.0	0.0	0.0	0.078	0.0	0.047	0.094	0.0	0.75	0.031
9	0.016	0.078	0.0	0.031	0.0	0.125	0.016	0.0	0.0	0.734

Table 3: Confusion Matrix for VQ (4 clusters)

Test\Predicted	0	1	2	3	4	5	6	7	8	9
0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.078	0.859	0.047	0.0	0.016	0.0	0.0	0.0	0.0	0.0
2	0.078	0.078	0.75	0.0	0.094	0.0	0.0	0.0	0.0	0.0
3	0.031	0.0	0.0	0.609	0.0	0.016	0.047	0.0	0.297	0.0
4	0.047	0.062	0.109	0.0	0.781	0.0	0.0	0.0	0.0	0.0
5	0.047	0.047	0.0	0.0	0.0	0.891	0.0	0.0	0.0	0.016
6	0.016	0.0	0.0	0.062	0.0	0.078	0.797	0.0	0.047	0.0
7	0.0	0.0	0.0	0.0	0.016	0.016	0.031	0.906	0.0	0.031
8	0.031	0.0	0.0	0.156	0.0	0.062	0.016	0.0	0.703	0.031
9	0.0	0.0	0.0	0.031	0.0	0.031	0.047	0.0	0.0	0.891

Table 4: Confusion Matrix for VQ (8 clusters)

Test\Predicted	0	1	2	3	4	5	6	7	8	9
0	0.984	0.0	0.0	0.0	0.0	0.016	0.0	0.0	0.0	0.0
1	0.0	0.984	0.0	0.0	0.016	0.0	0.0	0.0	0.0	0.0
2	0.094	0.031	0.719	0.0	0.156	0.0	0.0	0.0	0.0	0.0
3	0.109	0.0	0.0	0.625	0.0	0.031	0.078	0.0	0.156	0.0
4	0.094	0.0	0.125	0.0	0.781	0.0	0.0	0.0	0.0	0.0
5	0.078	0.031	0.0	0.0	0.0	0.859	0.0	0.0	0.0	0.031
6	0.016	0.0	0.0	0.031	0.031	0.0	0.875	0.0	0.047	0.0
7	0.0	0.0	0.0	0.0	0.031	0.016	0.0	0.922	0.0	0.031
8	0.0	0.0	0.0	0.031	0.0	0.0	0.125	0.0	0.844	0.0
9	0.0	0.016	0.0	0.016	0.0	0.016	0.016	0.0	0.0	0.938

Table 5: Confusion Matrix for VQ (16 clusters)

Confusion Matrix From the confusion matrix in both the cases, we can observe following majorly dominating confusions:

- 'three' is confused as 'eight'. This can be regarded to common sound /t/
- 'eight' is confused as 'three'. This can be regarded to common sound /t/
- 'zero' has the least confusion, it can be said that all the models work very well for 'zero'
- 'nine' is confused with 'five' (for VQ 4 clusters). This can be regarded to the common sound /v/
- 'five' is confused with 'one' (for VQ 4 clusters). This can be regarded to the common sound /v/

Q4 Dynamic Time Warping

DTW aligns the whole word by finding the time warping which minimizes the total distortion based on the sum of individual frame distances. This is implemented using DTW matrix whose dimensions are determined by lengths of reference pattern and test pattern. The path starting from the left top corner is allowed to mske certain fixed transitions like unit horizontal to the right, unit diagonal to the downright etc. We try to find the path which minimizes the cost. A dynamic programming based approach is available to solve this problem which relies on the fact that best path passing through a point is also the best path till that point. The speciality of this approach is that it incorporates time sequencial information of frames which was missing the above two approaches. Computationally it is very expensive if we have large number of reference patterns.

Results WER = 0.109
Confusion Matrix

Test\Predicted	0	1	2	3	4	5	6	7	8	9
0	0.984	0.0	0.0	0.0	0.0	0.0	0.016	0.0	0.0	0.0
1	0.0	0.891	0.0	0.0	0.062	0.047	0.0	0.0	0.0	0.0
2	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.703	0.0	0.0	0.25	0.0	0.047	0.0
4	0.0	0.0	0.109	0.0	0.859	0.0	0.031	0.0	0.0	0.0
5	0.0	0.031	0.0	0.0	0.0	0.938	0.016	0.0	0.0	0.016
6	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.125	0.875	0.0	0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.234	0.0	0.766	0.0
9	0.0	0.0	0.0	0.0	0.0	0.062	0.047	0.0	0.0	0.891

Table 6: Confusion Matrix for DTW

As we can observe that WER is least for the DTW method. DTW method takes into accound time sequential information unlike other two methods in previous section. Hence WER is expected to reduce. Also the confusion matrix closer to diagonal for DTW than other methods. Major confusions are

- 'three' confused as 'six'
- 'eight' confused as 'six'
- 'seven' confused as 'six'

The major confusions are observed to be different than other template based methods. The previous two methods used to get confused between similar phonemes, but this method takes into account time information as well. Majority of confusions are with digit 'six', because 'six' wasn't segmented exactly

due to burst at the end, which is neglected in segmentation for few cases. If a phoneme of two digits are same and occur in the same relative time order, DTW is expected to confuse between the two. That is why 'seven' and 'six' (beginning with /s/), 'three' and 'six' (/i/ in middle and end) etc. confusions are found.

Major Codes

Q1 Endpoint detector

```
def find_enpoints(y):
        y = np.asarray(y,'float')
        y = y/max(y)
        window_len = 3501
        t = np.linspace(0,len(y)/Fs,len(y)+1)
        energy = 0;
        w = np.hamming(window_len)
        energy = np.convolve(y**2,w**2,'same')
        energy = energy/max(energy)
        thresh = 0.008;
        energy_thresh = (energy > thresh).astype('float')
        points = np.nonzero(abs(energy_thresh[1:] - energy_thresh[0:-1]))[0]
        start_points = [points[2*i] for i in range(len(points)/2)]
        end_points = [ points[2*i+1] for i in range(len(points)/2)]
        return start_points,end_points
```

Q2 MFCC feature extraction

```
def melFilter(no_of_filters,lowerf,upperf,Fs,no_of_DFT_points):
        no_of_DFT_points = n_fft
        mel_freq_lower = 2595*log10(1+(lowerf/700));
        mel_freq_upper = 2595*log10(1+(upperf/700));
        mel_spacing = (mel_freq_upper - mel_freq_lower)/(no_of_filters+1);
        mel_filter_cutoffs = [mel_freq_lower + i*mel_spacing for i in range(no_of_filters +
2)];
        linear_filter_cutoffs = [(10**(mel_filter_cutoffs[i]/2595) - 1)*700 for i in
range(len(mel_filter_cutoffs))]
        linear_filter_cutoffs_discrete = [np.round(i*(no_of_DFT_points)/Fs) for i in
linear_filter_cutoffs];

        filter_bank = np.zeros([no_of_filters,no_of_DFT_points/2+1]);
        for i in range(1,len(linear_filter_cutoffs_discrete)-1):
            for j in
range(int(linear_filter_cutoffs_discrete[i-1]),int(linear_filter_cutoffs_discrete[i+1])+1):
                if (j == linear_filter_cutoffs_discrete[i]):
                    filter_bank[i-1,j] = 1

                elif(j < linear_filter_cutoffs_discrete[i]):
                    filter_bank[i-1,j] =
(j-linear_filter_cutoffs_discrete[i-1])/(linear_filter_cutoffs_discrete[i] -
linear_filter_cutoffs_discrete[i-1]);
                else:
```

```
                    filter_bank[i-1,j] =
(linear_filter_cutoffs_discrete[i+1]-j)/(linear_filter_cutoffs_discrete[i+1] -
linear_filter_cutoffs_discrete[i]);
        # for k in range(no_of_filters):
        #       plt.plot(filter_bank[k,:])
        #       plt.hold
        #       plt.draw()
        # plt.show()
        return filter_bank
filter_bank = melFilter(no_of_filters,lowerf,upperf,Fs,n_fft)

def extract_features(signal):
        n_frames= 1+ math.floor((len(signal)-(frame_size*Fs))/(frame_hop*Fs))
        MFCC = []
        for i in range(int(n_frames)):
                frame= signal[i*(frame_hop*Fs):(frame_size*Fs)+i*(frame_hop*Fs)]
                w = np.hamming(len(frame))
                w_frame = [w[i]*frame[i] for i in range(len(frame))]
                temp = np.fft.fft(w_frame,n_fft)
                dft = temp[0:(n_fft)/2+1     ]
                dft_mag = [(abs(t))**2 for t in dft]
                mel_coefs = np.zeros(no_of_filters)
                mel_coefs = [np.sum([dft_mag[j]*filter_bank[t,j] for j in range(len(dft_mag))])
for t in range(no_of_filters)]
                # for tt in range(no_of_filters):
                #       for jj in range(len(dft_mag)):
                #               mel_coefs[tt] += dft_mag[jj]*filter_bank[tt,jj]

                log_mel_coefs = [20*log10(abs(coef)) for coef in mel_coefs]
                mfcc_temp = scipy.fftpack.dct(log_mel_coefs)
                mfcc = mfcc_temp[1:14]
                #mfcc=(abs(numpy.fft.ifft(mel_coefs)))[:13]
                MFCC.append(mfcc)

        MFCC = np.asarray(MFCC)
        delta_vecs = np.zeros(MFCC.shape)
        for i in range(1,MFCC.shape[1]-1):
            delta_vecs[:,i] = np.subtract(MFCC[:,i+1] ,MFCC[:,i-1])/2

        delta_delta_vecs = np.zeros(MFCC.shape)
        for i in range(1,MFCC.shape[1]-1):
            delta_delta_vecs[:,i] = np.subtract(delta_vecs[:,i+1], delta_vecs[:,i-1])/2

        final_feature_vecs = np.transpose(np.concatenate([MFCC,delta_vecs,delta_delta_vecs],1))

        return final_feature_vecs
```

Q3 (i) Bag of frames

```
fs=8000
d=['zero','one','two','three','four','five','six','seven','eight','nine']
male_sounds=os.listdir("../data/Digits male 8Khz Updated")
male_speakers=os.listdir("Male_segmented")
female_speakers=os.listdir("Female_segmented")
all_speakers = male_speakers + female_speakers

CodeBook = {}
n_frame_dict = {}

for digit in d:
	CodeBook[digit] = {}
	n_frame_dict[digit] = {}
	for speaker in all_speakers:
		if(speaker in male_speakers):
			fp = "Male_segmented/"+speaker
		if(speaker in female_speakers):
			fp = "Female_segmented/"+speaker
		CodeBook[digit][speaker] = []
		n_frame_dict[digit][speaker] = []
		for tt in range(1,5):
			Fs,signal = read(fp+'/'+digit+'_'+str(tt)+'.wav')
			feature_mat = extract_features(signal)
			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 digit
# pickle.dump(CodeBook,open('CodeBook_v1','wb'))
# pickle.dump(n_frame_dict,open('n_frame_dict_v1','wb'))
CodeBook = pickle.load(open('CodeBook_v1','r'))
n_frame_dict = pickle.load(open('n_frame_dict_v1','r'))
confusion_matrix = np.zeros([10,10])
# test_speaker = male_speakers[0]
for test_speaker in all_speakers:
	train_speakers = male_speakers+female_speakers
	train_speakers.remove(test_speaker)
	for test_digit in d:
		for utterance in range(1,5):
			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 d:
				for l in range(len(test_mat)):
					test_vec = np.asarray(test_mat)[l,:]
					min_dist = float('Inf')
					for speaker in train_speakers:
						temp_list = CodeBook[digit][speaker]
```

```
                                        curr_dist,index =
spatial.KDTree(temp_list).query(test_vec)
                                        if(curr_dist < min_dist):
                                                min_dist = curr_dist

                                sum_dist[d.index(digit)]+=min_dist
                        pred_digit = np.argmin(sum_dist)
                        print test_speaker,pred_digit,test_digit
                        confusion_matrix[d.index(test_digit),pred_digit] += 1
        np.save('BOF_confusion_matrix',confusion_matrix)
wer = 1 - np.trace(confusion_matrix)/640
print wer
```

Q 3(ii) Vector Quantization

```
fs=8000
d=['zero','one','two','three','four','five','six','seven','eight','nine']
male_sounds=os.listdir("../data/Digits male 8Khz Updated")
male_speakers=os.listdir("Male_segmented")
female_speakers=os.listdir("Female_segmented")
all_speakers = male_speakers + female_speakers
CodeBook = {}
n_frame_dict = {}
CodeBook = pickle.load(open('CodeBook_v1','r'))
n_frame_dict = pickle.load(open('n_frame_dict_v1','r'))
n_clusters = 16
VQCodeBook = {}
centroids = {}
confusion_matrix = np.zeros([10,10])
# test_speaker = male_speakers[0]
for test_speaker in all_speakers:
        train_speakers = male_speakers+female_speakers
        train_speakers.remove(test_speaker)
        for digit in d:
                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 d:
                for utterance in range(1,5):
```

```
                        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 d:
                                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[d.index(digit)]+=min_dist

                        pred_digit = np.argmin(sum_dist)
                        print test_speaker,pred_digit,test_digit
                        confusion_matrix[d.index(test_digit),pred_digit] += 1.0
        np.save('VQ_confusion_matrix_n_cluster'+str(n_clusters),confusion_matrix)
# print confusion_matrix/64
wer = 1 - np.trace(confusion_matrix)/640
confusion_matrix = confusion_matrix/64
d = np.around(confusion_matrix,decimals = 3)
np.savetxt("../report/VQ_confusion_matrix"+str(n_clusters)+".csv", d, fmt = '%s')
print wer
```

Q 4 DTW

```
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];
```