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];
```