Speech Technology (EE 624)

# **Course Project Report**

Problem statements:

Record twenty five utterances of each of the isolated English digits (0-9), vowels (a,e,i,o,u), and the given sentence (I'm registered with speech technology course and recording this data for course project) in your voice. Record speech data over PC using Audacity software in PCM format with a sampling frequency of 16 kHz and a precision of 16-bits. Store the recorded data as: <Roll.No>\_<digit>/<vowel>/sentence\_<recording\_num>.wav.Human voice characteristics usually differ slightly over the time and it is referred to as the session variation. So you are advised to record different entities atleast in two sessions.

## Project 1: Perform DTW based isolated vowel recognition

For each of the vowels, choose one of the utterance as the reference template and the remaining ones as the test templates. Compute 39-dimensional MFCC features comprising base, delta, and delta-delta components for all the recordings using MATLAB. Using DTW command of MATLAB, write a script that computes DTW scores for any two similar or dissimilar test utterances and also plots the corresponding DTW curve.

Procedure :

1. Find the MFCC features of reference audio file and test files using python\_speech\_features library
2. Perform padding if required
3. Apply DTW technique from dtw library between test and reference file
4. Print the output i.e both the dtw score and the graph between ref and test file

## CODE :

import numpy as np

from dtw import dtw

import matplotlib.pyplot as plt

import scipy.io.wavfile as wav

from python\_speech\_features import mfcc

data = ['a','e','i','o','u']

#for padding the array

def pad(arr):

    pad\_width = 100000-len(arr)

    return np.pad(arr,(0,pad\_width),mode='constant')

for x in data:

    #give the path of the audio file

    path = 'data/'+str(x)+'/224102324\_'+str(x)+'\_1.wav'

    #load the audio file

    rate1, ref\_audio = wav.read(path)

    print(len(ref\_audio))

    print(len(ref\_audio))

    # select number of mfcc features

    numCoeffs = 39

    len\_of\_frame = int(round(0.025 \* rate1))

    shift\_in\_frame = int(round(0.010 \* rate1))

    nfft = 2\*\*int(np.ceil(np.log2(len\_of\_frame)))

    # Preprocess the audio file

    ref\_audio = np.append(ref\_audio[0], ref\_audio[1:] - 0.97 \* ref\_audio[:-1])

    ref\_audio = pad(ref\_audio)

    # Get the mfcc coefficients

    ref\_mfcc = mfcc(ref\_audio, rate1, numcep=numCoeffs, winlen=len\_of\_frame/rate1, winstep=shift\_in\_frame/rate1, \

                  nfft=nfft, nfilt=numCoeffs, lowfreq=0, highfreq=None, preemph=0.97)

    i = 2

    fig, ax = plt.subplots(6,4,figsize=(15,15))

    sup\_title = "DTW results for vowel  : '"+ str(x)+"'"

    fig.suptitle(sup\_title, fontsize=13, fontweight='bold', y=0.95)

    for j in range(6):

        for k in range(4):

            path  = 'data/'+str(x)+'/224102324\_'+str(x)+'\_'+str(i)+'.wav'

            rate2, test\_audio = wav.read(path)

            # test\_audio = pad(test\_audio)

            test\_audio = np.append(test\_audio[0], test\_audio[1:] - 0.97 \* test\_audio[:-1])

            test\_audio = pad(test\_audio)

            test\_mfcc = mfcc(test\_audio, rate2, numcep=numCoeffs, winlen=len\_of\_frame/rate2, winstep=shift\_in\_frame/rate2, \

                  nfft=nfft, nfilt=numCoeffs, lowfreq=0, highfreq=None, preemph=0.97)

            dist, cost, acc, path = dtw(ref\_mfcc.T, test\_mfcc.T, dist=lambda x, y: np.linalg.norm(x - y, ord=1))

            ax[j,k].imshow(acc.T, origin='lower', cmap='gray', interpolation='nearest')

            ax[j,k].plot(path[0], path[1], '-b')

            ax[j,k].axis('off')

            ax[j,k].set\_title(round(dist,2))

            i = i+1

    plt.show()

# DTW between dissimilar items

for x in data:

    # give the path of the audio file

    path = 'data/'+str(x)+'/224102324\_'+str(x)+'\_1.wav'

    rate1, ref\_audio = wav.read(path)

    # ref\_audio = pad(ref\_audio)

    # select number of mfcc features

    numCoeffs = 39

    len\_of\_frame = int(round(0.025 \* rate1))

    shift\_in\_frame = int(round(0.010 \* rate1))

    nfft = 2\*\*int(np.ceil(np.log2(len\_of\_frame)))

    # preprocess the audio file

    ref\_audio = np.append(ref\_audio[0], ref\_audio[1:] - 0.97 \* ref\_audio[:-1])

    ref\_audio = pad(ref\_audio)

    # Get the mfcc coefficients

    ref\_mfcc = mfcc(ref\_audio, rate1, numcep=numCoeffs, winlen=len\_of\_frame/rate1, winstep=shift\_in\_frame/rate1, \

                  nfft=nfft, nfilt=numCoeffs, lowfreq=0, highfreq=None, preemph=0.97)

    for y in data:

        fig, ax = plt.subplots()

        print(x,y)

        path  = 'data/'+str(y)+'/224102324\_'+str(y)+'\_1.wav'

        rate2, test\_audio = wav.read(path)

        # test\_audio = pad(test\_audio)

        test\_audio = np.append(test\_audio[0], test\_audio[1:] - 0.97 \* test\_audio[:-1])

        test\_audio = pad(test\_audio)

        test\_mfcc = mfcc(test\_audio, rate2, numcep=numCoeffs, winlen=len\_of\_frame/rate2, winstep=shift\_in\_frame/rate2, \

              nfft=nfft, nfilt=numCoeffs, lowfreq=0, highfreq=None, preemph=0.97)

        dist, cost, acc, path = dtw(ref\_mfcc.T, test\_mfcc.T, dist=lambda x, y: np.linalg.norm(x - y, ord=1))

        ax.imshow(acc.T, origin='lower', cmap='gray', interpolation='nearest')

        ax.plot(path[0], path[1], '-b')

        ax.axis('off')

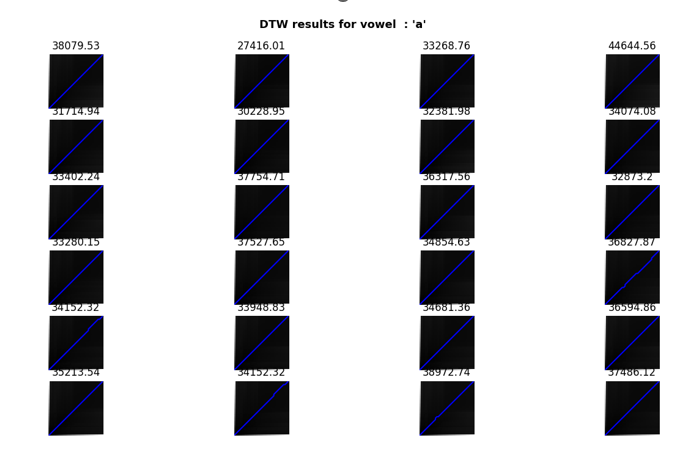
        title = str(round(dist,2))+' between '+str(x)+" "+str(y)

        ax.set\_title(title)

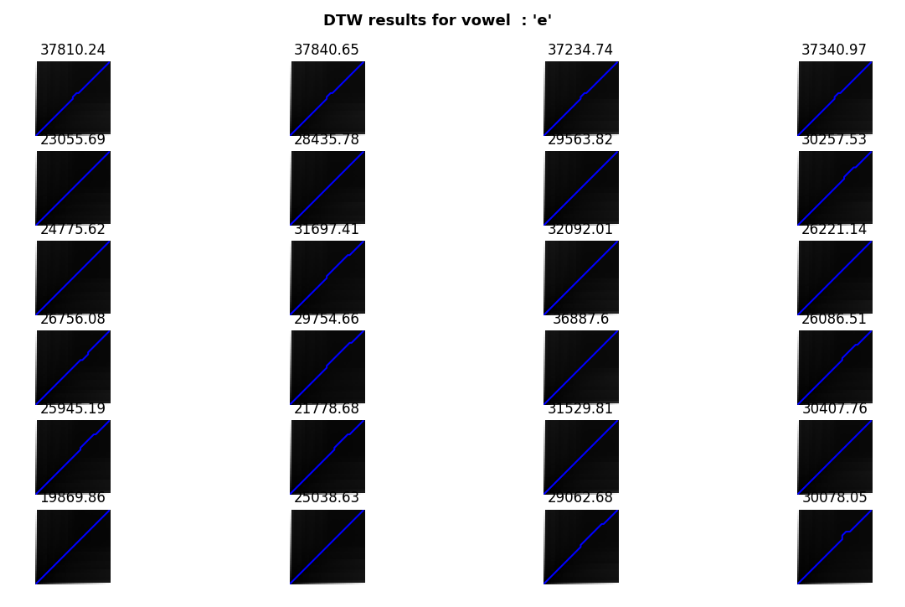
        plt.show()

Results : Each figure is shown with dtw distance.

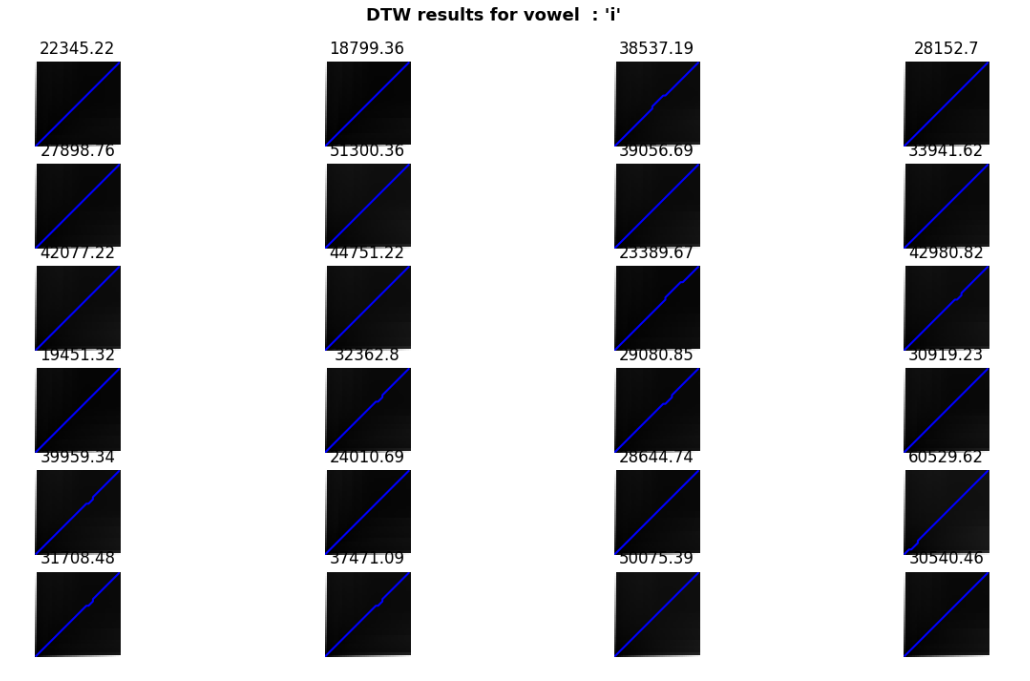
For vowel ‘a’ :



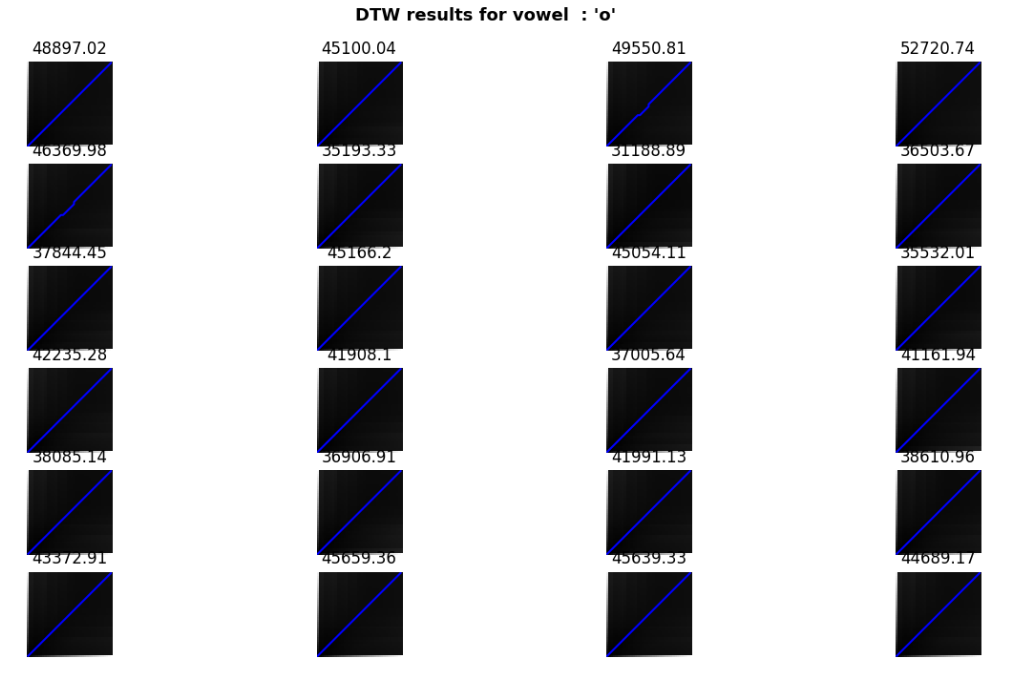
For vowel ‘e’ :



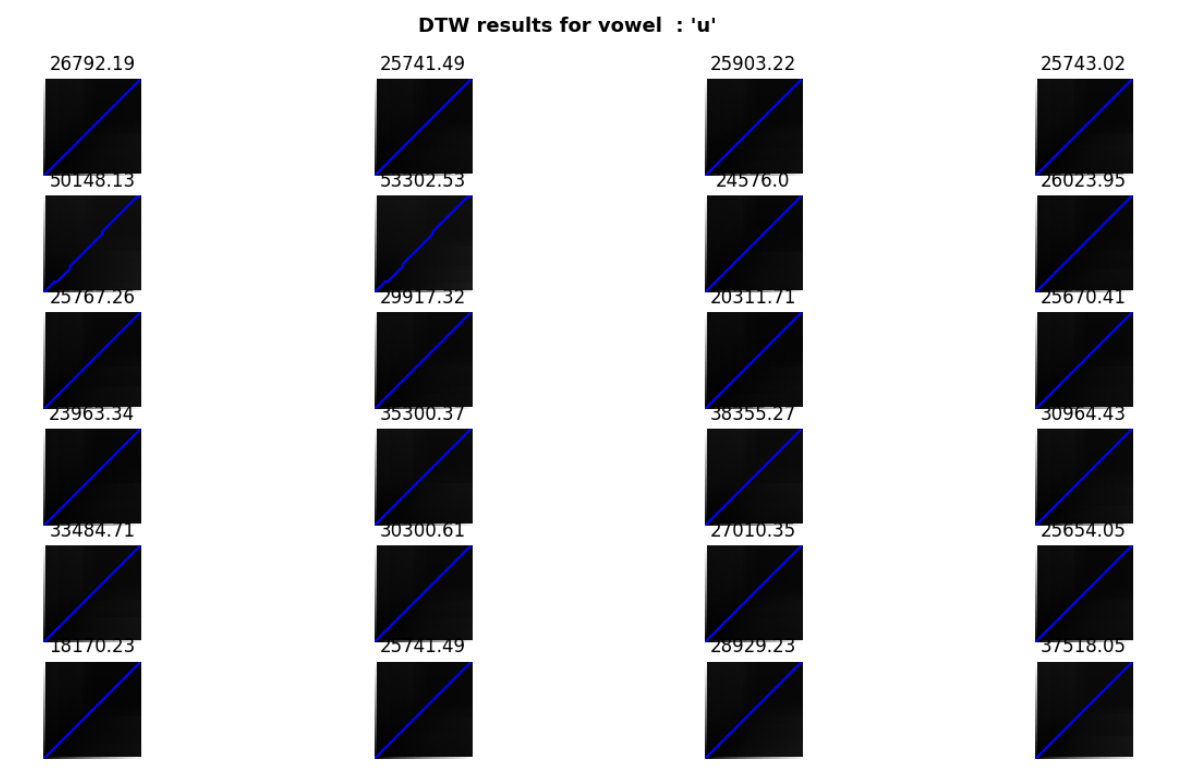
For vowel ‘i’ :



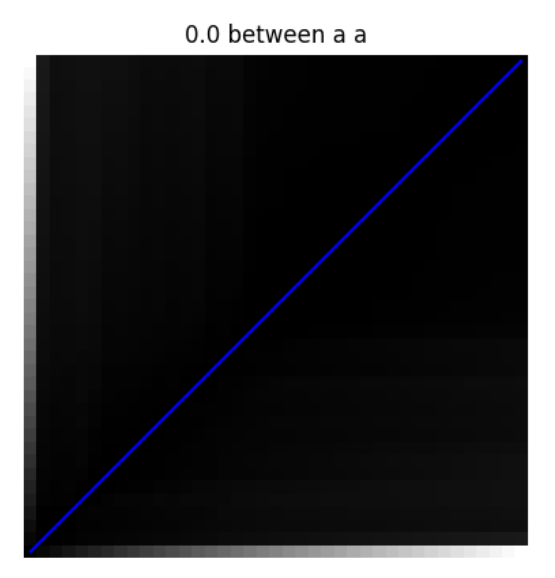
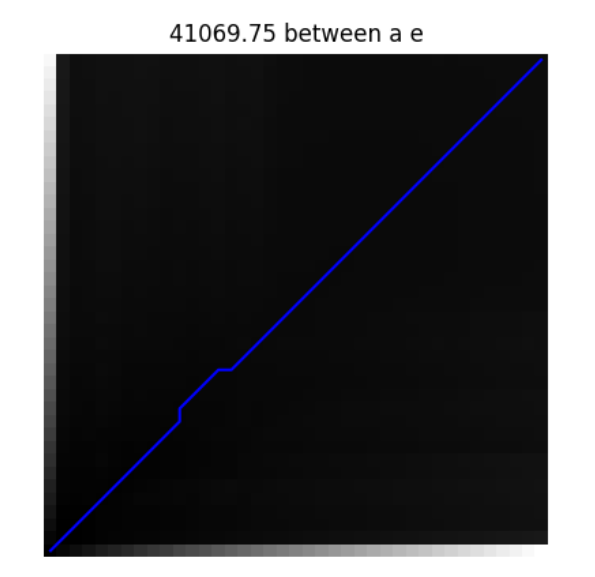
For vowel ‘o’ :



For vowel ‘u’ :



**DTW between dissimilar elements**

There should be zero distance between similar elements and non zero between dissimilar elements.

# Project 2 : Perform GMM based isolated digit recognition

Given the training and testing file lists, compute 39-dimensional MFCC features and learn a GMM having 16/32 densities for each of the isolated digits using training data.

Identify the digits in test data using the maximum likelihood rule over the trained GMMs and report the confusion matrix for the recognition task.

**PROCEDURE :**

1. Define the number of Gaussians
2. Extract MFCC features
3. Load train audio files and start training GMM using GaussianMixture from sklearn.mixture module
4. Get the MFCC features of test file and then feed it to the GMM and get the predicted class

**CODE :**

import scipy.io.wavfile as wav

import matplotlib.pyplot as plt

import numpy as np

import librosa

from sklearn.mixture import GaussianMixture

from sklearn.metrics import confusion\_matrix

from dtw import dtw

#select number of mfcc coefficients

n\_mfcc = 39

#number of gaussians

no\_of\_gaussians = 32

no\_of\_iter = 100

# Find the MFCC features

def compute\_mfcc(audio\_file):

    # Give the path and load the audio file

    signal, sr = librosa.load(audio\_file, sr=None)

    # Get the MFCC features

    mfcc = librosa.feature.mfcc(y=signal, sr=sr, n\_mfcc=n\_mfcc)

    return mfcc.T

# start training

digit\_gmms = {}

# select some random files for training in between 0 to 25

data = [1,3,5,10,13,15,18,20,21,25] # 10 samples for training

for digit in range(10):

    # mfcc features

    train\_files = []

    for i in data:

        path = 'data/'+str(digit)+'/224102324\_'+str(digit)+'\_'+str(i)+'.wav'

        train\_files.append(path)

    train\_mfcc = []

    for train\_file in train\_files:

        mfcc1 = compute\_mfcc(train\_file)

        train\_mfcc.append(mfcc1)

    # get mfcc of current working digit

    digit\_mfcc = train\_mfcc

    gmm = GaussianMixture(n\_components=no\_of\_gaussians, max\_iter=no\_of\_iter)

    gmm.fit(np.vstack(digit\_mfcc))

    digit\_gmms[digit] = gmm

mfcc\_of\_test\_samples = []

labels\_in\_test = []

for digit in range(10):

    for i in range(1,26):

        # test all the data samples which are not taken for training (for remaining 15 samples)

        if(i in data):

            continue

        path = 'data/'+str(digit)+'/224102324\_'+str(digit)+'\_'+str(i)+'.wav'

        mfcc = compute\_mfcc(path)

        mfcc\_of\_test\_samples.append(mfcc)

        labels\_in\_test.append(digit)

classes\_pred = []

for mfcc in mfcc\_of\_test\_samples:

    log\_likelihoods = []

    for digit, gmm in digit\_gmms.items():

        log\_likelihood = gmm.score(mfcc)

        log\_likelihoods.append(log\_likelihood)

    class\_predicted = np.argmax(log\_likelihoods)

    classes\_pred.append(class\_predicted)

conf\_mat = confusion\_matrix(labels\_in\_test, classes\_pred)

print("Confusion matrix:\n", conf\_mat)

## RESULTS :

## Confusion matrix:

## [[14 0 0 0 1 0 0 0 0 0]

## [ 0 13 0 0 0 0 0 2 0 0]

## [ 0 0 15 0 0 0 0 0 0 0]

## [ 0 0 0 14 0 0 0 0 1 0]

## [ 1 0 0 0 13 0 0 0 0 1]

## [ 0 1 0 0 0 10 0 3 0 1]

## [ 0 0 0 0 0 0 15 0 0 0]

## [ 0 0 0 0 0 0 0 14 0 1]

## [ 0 0 0 0 0 0 0 0 15 0]

## [ 0 0 0 0 0 0 0 0 0 15]]

Accuracy = ( All true predictions / total predictions )

= 138/150

= 92.00 %