213070010_Assignment3

ASR: Automatic Speech Recognition

¶

README

- Unzip all the '.npy' files in the ModifiedMFCC folder to obtain the feature matrix for every class and update the path in the "Loading Saved Features" section.
- DO NOT EXECUTE any cells in the "Feature transforming and Saving the features". It takes a lot of time.
- The data size is very large, thus couldn't be submitted with the assignment. Please used the follwoing data from the google drive: Google Drive folder

In [ ]:

```
%cd '/content/drive/MyDrive/Academics/Semester 1/Speech Processing/Assignment 3'
```

```
/content/drive/MyDrive/Academics/Semester 1/Speech Processing/Assignment 3
```

Data Preparation¶

Unzipping all the audio files within the train dataset

Data Analysis¶

In this block, we analyze the number of utterance in each class of the training and testing datasets.

In [ ]:

```
import os
import glob

files_and_folders = glob.glob('/content/drive/MyDrive/Academics/Semester 1/Speech Processing/Assignment 3/Commands Dataset/train/*')
files = glob.glob('/content/drive/MyDrive/Academics/Semester 1/Speech Processing/Assignment 3/Commands Dataset/train/*.zip')

train_folders = [i for i in files_and_folders if i not in files]
```

In [ ]:

```
train_dictionary = {}
for i in train_folders:
  print("Number of audio files in the training set folder ",
i.split("ain/")[1], ":", len(glob.glob(i + "/*")))
  train_dictionary[i.split("ain/")[1]] = glob.glob(i + "/*")
```

```
Number of audio files in the training set folder  right : 2367
Number of audio files in the training set folder  go : 2372
Number of audio files in the training set folder  yes : 2377
Number of audio files in the training set folder  no : 2375
Number of audio files in the training set folder  off : 2357
Number of audio files in the training set folder  on : 2367
Number of audio files in the training set folder  up : 2375
Number of audio files in the training set folder  down : 2359
Number of audio files in the training set folder  left : 2353
Number of audio files in the training set folder  stop : 2380
```

In [ ]:

```
test_noisy_folders =
glob.glob("/content/drive/MyDrive/Academics/Semester 1/Speech
Processing/Assignment 3/Commands Dataset/test_noisy/*")
test_clean_folders =
glob.glob("/content/drive/MyDrive/Academics/Semester 1/Speech
Processing/Assignment 3/Commands Dataset/test_clean/*")

test_noisy_dict = {}
for i in test_noisy_folders:
  print("Number of audio files in the test_noisy folder ",
i.split("isy/")[1], ":", len(glob.glob(i + "/*")))
  test_noisy_dict[i.split("isy/")[1]] = glob.glob(i + "/*")

print("-------------------------------------------------------------------
-----------------------------------------")

test_clean_dict = {}
for i in test_clean_folders:
  print("Number of audio files in the test_clean folder ",
i.split("ean/")[1], ":", len(glob.glob(i + "/*")))
  test_clean_dict[i.split("ean/")[1]] = glob.glob(i + "/*")
```

```
Number of audio files in the test_noisy folder  right : 259
Number of audio files in the test_noisy folder  up : 272
Number of audio files in the test_noisy folder  yes : 256
Number of audio files in the test_noisy folder  on : 246
Number of audio files in the test_noisy folder  down : 253
Number of audio files in the test_noisy folder  no : 252
Number of audio files in the test_noisy folder  stop : 249
Number of audio files in the test_noisy folder  left : 267
```

```
Number of audio files in the test_noisy folder  off : 262
Number of audio files in the test_noisy folder  go : 251
----------------------------------------------------------------------
-----------------------------------
Number of audio files in the test_clean folder  off : 262
Number of audio files in the test_clean folder  up : 272
Number of audio files in the test_clean folder  right : 259
Number of audio files in the test_clean folder  down : 253
Number of audio files in the test_clean folder  yes : 256
Number of audio files in the test_clean folder  left : 267
Number of audio files in the test_clean folder  on : 246
Number of audio files in the test_clean folder  no : 252
Number of audio files in the test_clean folder  go : 251
Number of audio files in the test_clean folder  stop : 249
```

In [ ]:

```
import scipy
from scipy.io.wavfile import read, write
import IPython
```

Pre-processing¶

In this block, we perform the pre-processing steps such as end-pointing and pre-emphasis which are necessary to get rid of the redundant information in the utterances (end-pointing), and also to make the utterance more distinct (pre-emphasis).

End-Pointing¶

Determining the start and end of an utterance in an audio sequence. For this, window the signal and extract frames with a small window size and frame rate. We compute the short time energy corresponding to every frame in the utterance. We know that as noise or silence part of the sound will have relatively less energy than the voiced part. Thus, the starting point is where there is a sharp increase in the ST energy of the signal from the average signal energy in the frames seen before. Similarly, the end point is the region where there is a sharp drop in the energy signal.

\

Algorithm ⬇¶

- Compute short term energy for the signal with a hop rate of 1 sample
- Loop the short-term energy for the utterance and check if the next sample is higher than the mean of the previous samples by a thrshold.
- If yes, the index of this frame is the start point
- Reverse the short term energy signal
- Loop the reversed short-time energy signal to determine overshoot above the threshold and get the end-point.

In [ ]:

```
import numpy as np
import matplotlib.pyplot as plt
```

In [ ]:

```
# Dividing the audio sequence into frames
def get_frames(x, step):
  arr = []
  for i in range(160,len(x)-160,step):
    arr.append(x[i-160:i+160])
  return arr
```

In [ ]:

```
# Short time energy for every frame
def get_st_energy(x):
  frames = get_frames(x,1)
  st_energy = []
  for frame in frames:
    st_energy.append(np.linalg.norm(frame) ** 2)
  template = np.zeros(len(x))
  template[160:len(x) -160] = st_energy
  return template
```

In [ ]:

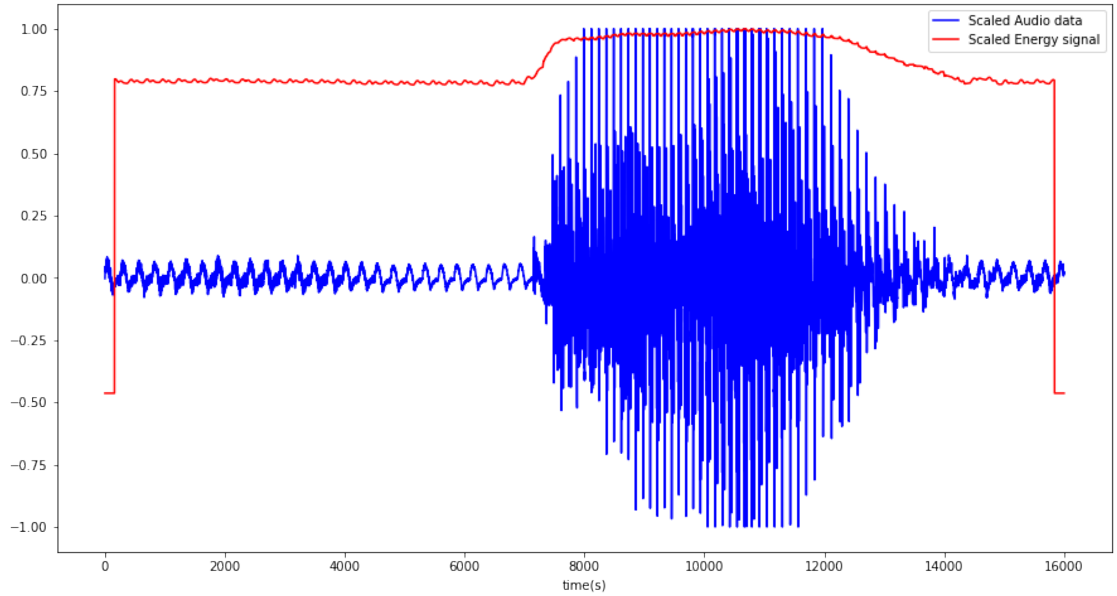
```
fs, data = read(train_dictionary['down'][200])
st_energy = get_st_energy(data) # This is simple energy
```

In [ ]:

```
# Normalizing the audio signal and the log scaled ST energy along with
the output to get the solution.
plt.figure(figsize = (15,8))
plt.plot(data/np.max(data), '-b', label = "Scaled Audio data")
plt.plot(np.log10(st_energy + 1e-5)/np.max(np.log10(st_energy + 1e-
5)), '-r', label="Scaled Energy signal")
plt.xlabel('time(s)')
plt.legend()
```

Out[ ]:

```
<matplotlib.legend.Legend at 0x7fd3c8037d50>
```



From the graph above we can see, that the voice content begins at the place where there is a sudden transition in the short term energy signal, or where it encounters the largest peak in the energy plot.

In [ ]:

```
'''
This function will look at points spaced 80 samples apart and try to
look for
changes in the energy signal, if there is a large change corresponding
to a signal
then the that will correspond to an audio signal
'''
import scipy.signal as sig

# End Pointing
def detecting_end_points(energy_sig):
    x1 = 0
    for i in range(161,len(energy_sig)-160):
      average = np.mean(energy_sig[160:i])
      if( energy_sig[i+100] - average > 0.1):
        x1 = i
        break

    x2 = len(energy_sig)
    rev_en_sig = list(reversed(energy_sig))
    for j in range(161,len(rev_en_sig)-x1):
      average = np.mean(rev_en_sig[160:j])
      if(average > 0.8):
        x2 = j
        break
      else:
```

```
        if(rev_en_sig[j+100] - average > 0.1):
          x2 = j
          break

    return x1,len(energy_sig) - x2 - 160
```

In [ ]:

```
# Checking end pointing for a sample from the training set
fs, data = read(train_dictionary['down'][200])
st_energy = get_st_energy(data)
x1,x2 = detecting_end_points(np.log10(st_energy +
1e-5)/np.max(np.log10(st_energy + 1e-5)))
print("Starting point the utterance : ", x1)
print("Ending point of the utterance :", x2)
```

```
Starting point the utterance :  7259
Ending point of the utterance : 13346
```

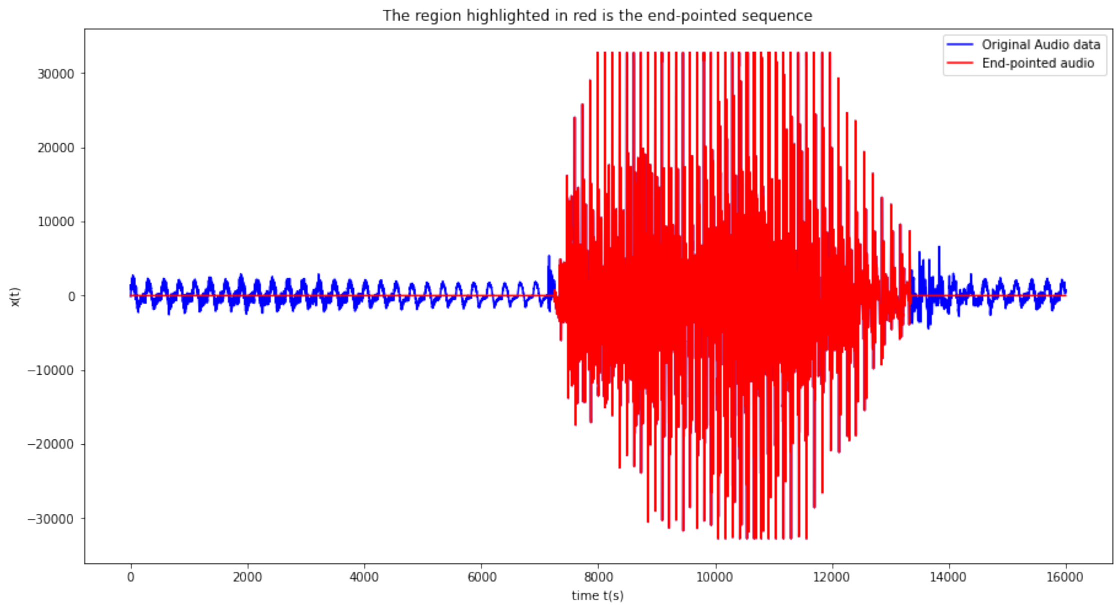
In [ ]:

```
# Playing plotting the end pointed audio audio
ref_array = np.zeros_like(data)
ref_array[x1:x2] = data[x1:x2]
plt.figure(figsize = (15,8))
plt.plot(data, '-b', label = "Original Audio data")
plt.plot(ref_array, '-r', label="End-pointed audio")
plt.title('The region highlighted in red is the end-pointed sequence')
plt.xlabel('time t(s)')
plt.ylabel("x(t)")
plt.legend()

print("End pointed Utterance : ")
IPython.display.Audio(data[x1:x2], rate = 16000)
```

```
End pointed Utterance :
```

Out[ ]:

Your browser does not support the audio element.



Applying Pre-emphasis¶

Pre-emphasis is a filtering method which passes the signal through a first-order high pass filter to give higher weightage to the high frequency band, and lower weightage to the lower frequency bands. The filter transfer function is usually of the form $H(z) = 1 - αz^{-1}$.

Pre-emphasis is done with the idea that higher frequencies have more contrasting features that are more important for signal disambiguation. This aids our model in distinguishing between different sounds better.

In [ ]:

```
# Helper function for pre-emphasis

def pre_emphasis(alpha, data):
  a = [1.0, -alpha]
  b = [1.0]
  audio_pre_emph = sig.lfilter(a,b,data)
  return audio_pre_emph
```

In [ ]:

```
# Applying pre-ephasis on the end-pointed signal
pre_emphasised = pre_emphasis(0.95,data[x1:x2])
```

In [ ]:

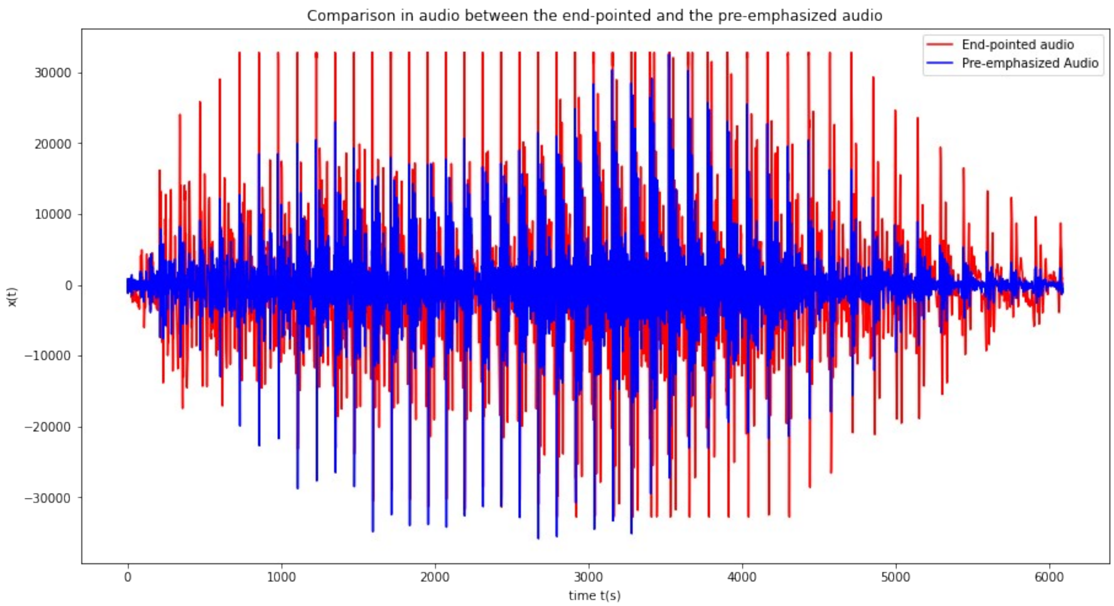
```
# Observing the effect of the end-pointed singal
plt.figure(figsize = (15,8))
plt.plot(data[x1:x2], '-r', label="End-pointed audio")
plt.plot(pre_emphasised, '-b', label = "Pre-emphasized Audio")
plt.title('Comparison in audio between the end-pointed and the pre-
```

```
emphasized audio')
plt.xlabel('time t(s)')
plt.ylabel("x(t)")
plt.legend()

# Playing the pre-emphasized audio
print("Pre-emphasized signal : ")
IPython.display.Audio(pre_emphasised.astype('int16'), rate = 16000)
```

```
Pre-emphasized signal :
```

Out[ ]:

Your browser does not support the audio element.



Feature Extraction¶

Features used : MFCC (Mel-Freqeuncy Cepstral Coefficients) and its derivatives

\

Algorithm ⬇¶

Getting the 39 dimensional MFCC vectors

- First 13 features are the MFCC coefficients obtained using the `librosa` package for speech processing in Python.
- Next 13 features are the velocity terms, computed by taking the first derivative of the cepstral coefficients by taking differences of coefficients in adjacent frames. ($Δ_{n} = \frac{c_{n+1} - c_{n-1}}{2}$). This is done by left shifting, and right shifting the cepstral coefficients, and taking their differences to get the gradient.

- Last 13 features are the acceleration terms computed taking the derivate of the velocity terms. ($a_{n} = \frac{Δ_{n+1} - Δ_{n-1}}{2}$). We compute the acceleration terms by taking gradient of the velocity terms as we did in the previous step.

In [ ]:

```
'''
Helper functions::

Function for computing the MFCC features from the path of the training
instances
'''
import librosa


def feature_extraction(path):
  fs,data = read(path)

  # Pre Processing to get pre-emphasized signal
  st_energy = get_st_energy(data)
  x1,x2 = detecting_end_points(np.log10(st_energy +
1e-5)/np.max(np.log10(st_energy + 1e-5)))
  pre_emphasised = pre_emphasis(0.95,data[x1:x2])

  # Setting the desired MFCC parameters
  n_mfcc = 13
  n_mels = 40
  n_fft = 320
  hop_length = 160
  fmin = 0
  fmax = None
  sr = 16000
  y = pre_emphasised
  features = librosa.feature.mfcc(y=y, sr=sr, n_fft=n_fft,
                                    n_mfcc=n_mfcc, n_mels=n_mels,
                                    hop_length=hop_length,
                                    fmin=fmin, fmax=fmax, htk=False)

  # Creating array to store the obtained MFCC vectors
  arr = np.zeros((features.T.shape[0], 39))

  # Computing velocity terms
  delta1 = np.zeros_like(features.T)
  delta2 = np.zeros_like(features.T)
  delta1[1:len(delta1),:] = features.T[0:len(delta1)-1, :]
  delta2[0:len(delta2)-1,:] = features.T[1:len(delta2),:]
  delta = (delta2 - delta1)/2

  # Computing Acceleration terms
```

```
  acc1 = np.zeros_like(delta1)
  acc2 = np.zeros_like(delta1)
  acc1[1:len(acc1),:] = delta1[0:len(acc1) -1, :]
  acc2[0:len(acc2)-1,:] = delta1[1:len(acc2),:]
  acc = (acc2 - acc1)/2

  # Adding the MFCCs along with their velocity and acceleration to the
feature matrix
  arr[:,0:13] = features.T
  arr[:,13:26] = delta
  arr[:,26:39] = acc

  # Returning the feature matrix as a list
  return arr.tolist()


# Function for computing distance between two vectors
def get_distance(x1,x2):
  return np.linalg.norm(x1-x2)
```

Feature Transforming and Saving the features¶

WARNING

Time Intensive Processes Ahead: Please do not execute any cells in this section¶

In this step, we compute features for every sample in every class in the dataset. We independently store all the features for a particular class in .npy format to aid future reusability.

In [ ]:

```
down = []
for i in range(len(train_dictionary['down'])):
  print("Sample Index :", i)
  features = feature_extraction(train_dictionary['down'][i])
  down = down + features
```

In [ ]:

```
down_arr = np.asarray(down)
np.save('/content/drive/MyDrive/Academics/Semester 1/Speech
Processing/Assignment 3/modifiedMFCC/down.npy', down_arr)
```

In [ ]:

```
right = []
for i in range(len(train_dictionary['right'])):
```

```
  print("Sample Index :", i)
  features = feature_extraction(train_dictionary['right'][i])
  right = right + features
```

In [ ]:

```
right_arr = np.asarray(right)
np.save('/content/drive/MyDrive/Academics/Semester 1/Speech
Processing/Assignment 3/modifiedMFCC/right.npy', right_arr)
```

In [ ]:

```
left = []
for i in range(len(train_dictionary['left'])):
  print("Sample Index :", i)
  features = feature_extraction(train_dictionary['left'][i])
  left = left + features
```

In [ ]:

```
left_arr = np.asarray(left)
np.save('/content/drive/MyDrive/Academics/Semester 1/Speech
Processing/Assignment 3/modifiedMFCC/left.npy', left)
```

In [ ]:

```
# Restart from here on 18th
go = []
for i in range(len(train_dictionary['go'])):
  print("Sample Index :", i)
  features = feature_extraction(train_dictionary['go'][i])
  go = go + features
```

In [ ]:

```
go_arr = np.asarray(go)
np.save('/content/drive/MyDrive/Academics/Semester 1/Speech
Processing/Assignment 3/modifiedMFCC/go.npy', go)
```

In [ ]:

```
yes = []
for i in range(len(train_dictionary['yes'])):
  print("Sample Index :", i)
  features = feature_extraction(train_dictionary['yes'][i])
  yes = yes + features
```

In [ ]:

```
yes_arr = np.asarray(yes)
np.save('/content/drive/MyDrive/Academics/Semester 1/Speech
Processing/Assignment 3/modifiedMFCC/yes.npy', yes_arr)
```

In [ ]:

```
no = []
for i in range(len(train_dictionary['no'])):
  print("Sample Index :", i)
  features = feature_extraction(train_dictionary['no'][i])
  no = no + features
```

In [ ]:

```
no_arr = np.asarray(no)
np.save('/content/drive/MyDrive/Academics/Semester 1/Speech
Processing/Assignment 3/modifiedMFCC/no.npy', no_arr)
```

In [ ]:

```
off = []
for i in range(len(train_dictionary['off'])):
  print("Sample Index :", i)
  features = feature_extraction(train_dictionary['off'][i])
  off = off + features
```

In [ ]:

```
off_arr = np.asarray(off)
np.save('/content/drive/MyDrive/Academics/Semester 1/Speech
Processing/Assignment 3/modifiedMFCC/off.npy', off_arr)
```

In [ ]:

```
on = []
for i in range(len(train_dictionary['on'])):
  print("Sample Index :", i)
  features = feature_extraction(train_dictionary['on'][i])
  on = on + features
```

In [ ]:

```
on_arr = np.asarray(on)
np.save('/content/drive/MyDrive/Academics/Semester 1/Speech
Processing/Assignment 3/modifiedMFCC/on.npy', on_arr)
```

In [ ]:

```
up = []
for i in range(len(train_dictionary['up'])):
  print("Sample Index :", i)
  features = feature_extraction(train_dictionary['up'][i])
  up = up + features
```

In [ ]:

```
up_arr = np.asarray(up)
np.save('/content/drive/MyDrive/Academics/Semester 1/Speech
Processing/Assignment 3/modifiedMFCC/up.npy', up_arr)
```

In [ ]:

```
stop = []
for i in range(len(train_dictionary['stop'])):
  print("Sample Index :", i)
  features = feature_extraction(train_dictionary['stop'][i])
  stop = stop + features
```

In [ ]:

```
stop_arr = np.asarray(stop)
np.save('/content/drive/MyDrive/Academics/Semester 1/Speech
Processing/Assignment 3/modifiedMFCC/stop.npy', stop_arr)
```

Loading the saved features¶

Alernatively, the feature arrays can be loaded from this folder as well : Google Drive Folder

In [ ]:

```
right_arr = np.load('/content/drive/MyDrive/Academics/Semester
1/Speech Processing/Assignment 3/modifiedMFCC/right.npy')
go_arr = np.load('/content/drive/MyDrive/Academics/Semester 1/Speech
Processing/Assignment 3/modifiedMFCC/go.npy')
yes_arr = np.load('/content/drive/MyDrive/Academics/Semester 1/Speech
Processing/Assignment 3/modifiedMFCC/yes.npy')
no_arr =np.load('/content/drive/MyDrive/Academics/Semester 1/Speech
Processing/Assignment 3/modifiedMFCC/no.npy')
off_arr = np.load('/content/drive/MyDrive/Academics/Semester 1/Speech
Processing/Assignment 3/modifiedMFCC/off.npy')
on_arr = np.load('/content/drive/MyDrive/Academics/Semester 1/Speech
Processing/Assignment 3/modifiedMFCC/on.npy')
up_arr = np.load('/content/drive/MyDrive/Academics/Semester 1/Speech
Processing/Assignment 3/modifiedMFCC/up.npy')
down_arr = np.load('/content/drive/MyDrive/Academics/Semester 1/Speech
Processing/Assignment 3/modifiedMFCC/down.npy')
left_arr = np.load('/content/drive/MyDrive/Academics/Semester 1/Speech
Processing/Assignment 3/modifiedMFCC/left.npy')
stop_arr = np.load('/content/drive/MyDrive/Academics/Semester 1/Speech
Processing/Assignment 3/modifiedMFCC/stop.npy')
```

Implementing VQ-Codebook Matching¶

Methodology¶

We will implement the VQ Codebook matching algorithm which uses a 'Bag of Frames' approach to perform ASR. Feature vectors corresponding to all the frames for every utterance in a particular class are clustered to determine some reference vectors which best represent that class. This creates a codebook of vectors corresponding to every class.

The matching happens by comparing the least, minimum average distance between the features of every frame of the test utterance with the protypical reference vectors of each class.

Training¶

We will be creating codebooks with differnt number of prototypical vectors for each class in each of the 4 codebook sets.

Note : Only 4 values of k were chosen because of the computational and time constraints.

The 4 codebook sets will have 5, 6, 7 and 8 prototypical reference vectors for every class respectively. The training of the codebook and finding of the optimal reference vector will be done using the K-means clustering algorithm.

Getting clusters for differnt every class, for different values of k (number of clsuter centroids or the number of reference vectors in each class)

In [ ]:

```
# Importing K-means
from sklearn.cluster import KMeans

# Data list
list_data = [right_arr, go_arr, yes_arr, no_arr, off_arr, on_arr,
up_arr, down_arr, left_arr, stop_arr]

# Getting the clusters (prototypical reference vectors) for each class
for different values of k
k = [5,6,8,10]
clusters_k = []
for m in k:
  print("Getting the cluster centres for k = ", m, " ....")
  clusters_per_class = []
  for j in range(len(list_data)):
    kmeans = KMeans(n_clusters=m, max_iter=1000,
random_state=0).fit(list_data[j])
    clusters_per_class.append(kmeans.cluster_centers_)
  clusters_k.append(clusters_per_class)
```

```
Getting the cluster centres for k =  5  ....
Getting the cluster centres for k =  6  ....
Getting the cluster centres for k =  8  ....
Getting the cluster centres for k =  10  ....
```

In [ ]:

```
for i in range(len(k)):
  print("Shape of the Codebook for k = ", k[i] ," : ",
np.shape(clusters_k[i]))
```

```
Shape of the Codebook for k =  5  :  (10, 5, 39)
Shape of the Codebook for k =  6  :  (10, 6, 39)
Shape of the Codebook for k =  8  :  (10, 8, 39)
Shape of the Codebook for k =  10  :  (10, 10, 39)
```

In [ ]:

```
'''
Helper function for getting the prediction label for a test vector

This function takes in the test feature vector, and the set of
prototypical
vectors corresponding to every class as input. They compute the
euclidean
distancebetween every frame the test feature vector with every frame
of the
protypical vectors and find the minimum average distance from every
class.
The index of the class corresponding to the minimum of these average
distances
is the class to which the test vector is assigned.
'''

def predict(test_vector, clusters_per_class):
  avg_dist = []
  for proto_frames in clusters_per_class:
    min_dist = 0
    for test_frame in test_vector:
      distances = list(map(lambda x: get_distance(x, test_frame),
proto_frames))
      min_dist = min_dist + np.min(distances)
    mean_distance = min_dist/(len(test_vector))
    avg_dist.append(mean_distance)
  index = np.argmin(avg_dist)
  return list(train_dictionary.keys())[index]
```

In [ ]:

```
'''
Helper function for predicting the metrics
```

```
This function get's the prediction corresponding to the test datset
Gets the predicion and the true labels as well
'''
def metrics(dictionary,proto_vectors):
  sum = 0
  samples = 0
  labels = []
  predictions = []
  for i in list(dictionary.keys()):
    for j in range(len(dictionary[i])):
      samples += 1
      features = feature_extraction(dictionary[i][j])
      labels.append(i)
      predictions.append(predict(features, proto_vectors))
      if(predict(features, proto_vectors) == i):
        sum += 1
  return (sum/samples)*100, predictions, labels
```

Results¶

Task A: Testing on clean test dataset¶

With each of the codebooks we have created for differnt classes, we compare the accuracies when we increase the number of reference vectors for each. Our objective is to see how well a higher set of reference vectors perform in comparison to a lower set.

In [46]:

```
from sklearn.metrics import confusion_matrix
import pandas as pd
import seaborn as sn

acc_arr_clean = []
preds_clean = []
labels_clean = []
for i in range(len(clusters_k)):
  acc1, preds1, labels1 = metrics(test_clean_dict, clusters_k[i])
  preds_clean.append(preds1)
  labels_clean.append(labels1)
  print("Accuracy on the clean test dataset for k = ", k[i] ," is ",
acc1)
  acc_arr_clean.append(acc1)
```

```
Accuracy on the clean test dataset for k =  5  is  49.20140241527074
Accuracy on the clean test dataset for k =  6  is  49.474094273470975
Accuracy on the clean test dataset for k =  8  is  52.23996883521621
Accuracy on the clean test dataset for k =  10  is  56.330346708219714
```

In [47]:

```
# Saving the above
np.save('/content/drive/MyDrive/Academics/Semester 1/Speech
Processing/Assignment 3/acc_arr_clean.npy', acc_arr_clean)
np.save('/content/drive/MyDrive/Academics/Semester 1/Speech
Processing/Assignment 3/preds_clean.npy', preds_clean)
np.save('/content/drive/MyDrive/Academics/Semester 1/Speech
Processing/Assignment 3/labels_clean.npy', labels_clean)
```

In [48]:

```
# Creating the confusion matrix for predictions based on differnt K-
means models
df_conf_mats_clean = []
for p in range(len(preds_clean)):
  conf_mat = confusion_matrix(labels_clean[p], preds_clean[p])
  df_cm = pd.DataFrame(conf_mat, index = [i for i in
list(test_clean_dict.keys())],columns = [i for i in
list(test_clean_dict.keys())])
  df_conf_mats_clean.append(df_cm)
```

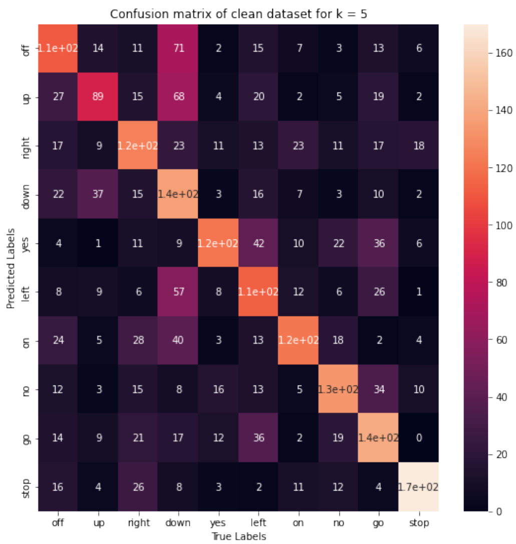
Confusion matrices on the test datset for different values of k

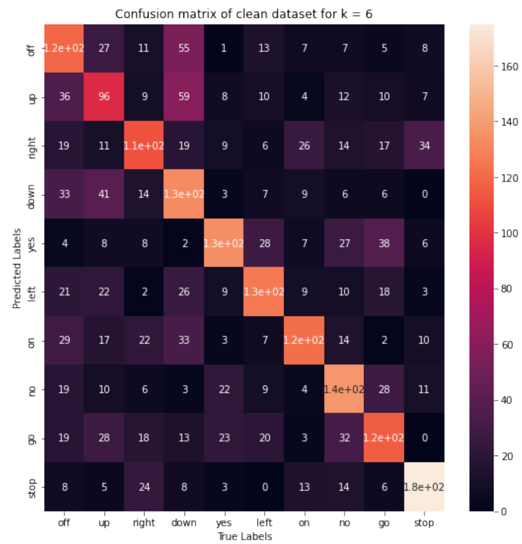
In [53]:

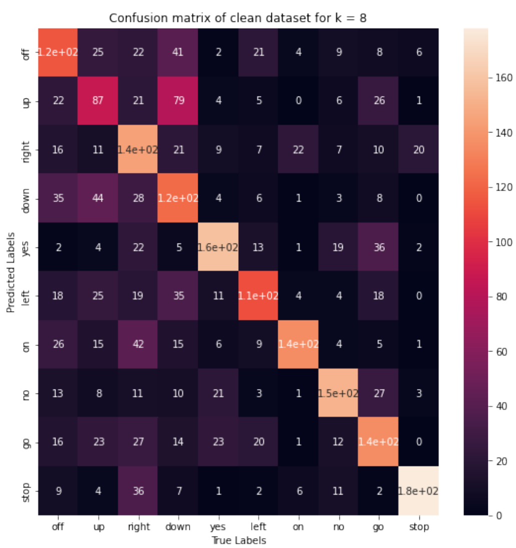
```
plt.figure(figsize = (20,20))
plt.subplot(2,2,1)
plt.title('Confusion matrix of clean dataset for k = 5')
sn.heatmap(df_conf_mats_clean[0], annot=True)
plt.xlabel('True Labels')
plt.ylabel('Predicted Labels')

plt.subplot(2,2,2)
plt.title('Confusion matrix of clean dataset for k = 6')
sn.heatmap(df_conf_mats_clean[1], annot=True)
plt.xlabel('True Labels')
plt.ylabel('Predicted Labels')

plt.subplot(2,2,3)
plt.title('Confusion matrix of clean dataset for k = 8')
sn.heatmap(df_conf_mats_clean[2], annot=True)
plt.xlabel('True Labels')
plt.ylabel('Predicted Labels')

plt.subplot(2,2,4)
plt.title('Confusion matrix of clean dataset for k = 10')
sn.heatmap(df_conf_mats_clean[3], annot=True)
plt.xlabel('True Labels')
plt.ylabel('Predicted Labels')
```

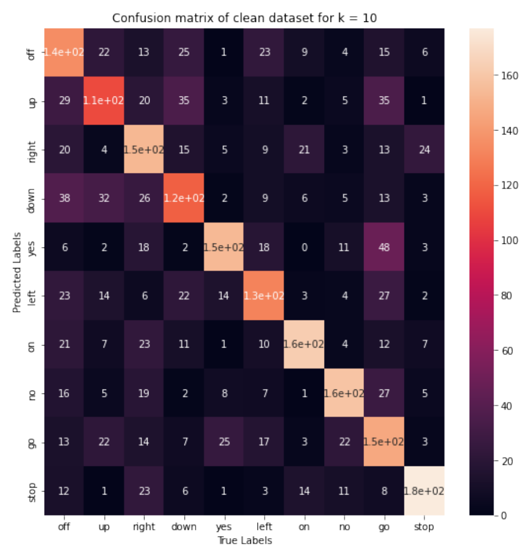
Out[53]:

```
Text(767.7272727272725, 0.5, 'Predicted Labels')
```









Trends from Confusion Matrices¶

Order of accuracies for different class of sounds, for different values of k

\

k = 5 ▶ stop > go > down > no > on = yes = right > left = off > up

k = 6 ▶ stop > no > down = yes = left > on > off > go > right > up

k = 8 ▶ stop > yes > no > go = on > right > down > off > left > up

k = 10 ▶ stop > on > no > right > yes > go > off > left > down > up

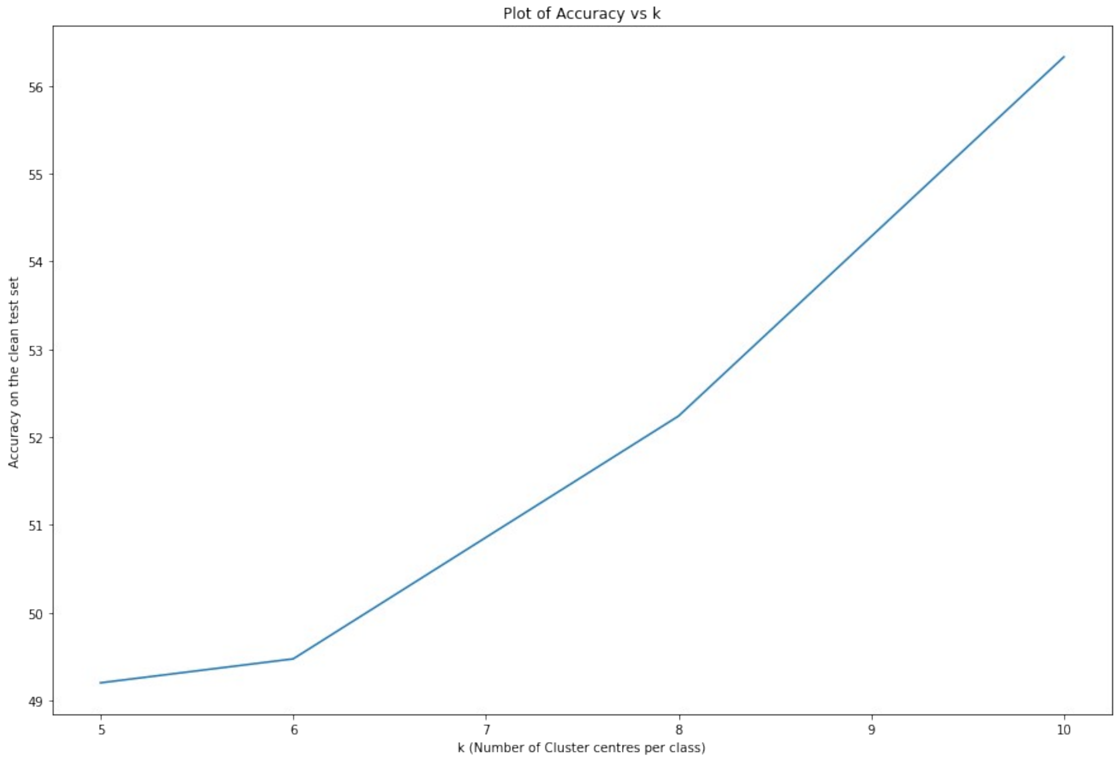
test accuracy vs k

In [55]:

```
plt.figure(figsize = (15,10))
plt.plot(k, acc_arr_clean)
plt.title('Plot of Accuracy vs k')
plt.xlabel('k (Number of Cluster centres per class)')
plt.ylabel('Accuracy on the clean test set')
```

Out[55]:

```
Text(0, 0.5, 'Accuracy on the clean test set')
```



Task B: Testing on noisy test dataset¶

In [ ]:

```
acc_arr_noisy = []
preds_noisy = []
labels_noisy = []

for i in range(len(clusters_k)):
  acc2, preds2, labels2 = metrics(test_noisy_dict,clusters_k[i])
  preds_noisy.append(preds2)
  labels_noisy.append(labels2)
  print("Accuracy on the noisy test dataset for k = ", k[i] ," is ",
acc2)
  acc_arr_noisy.append(acc2)
```

```
Accuracy on the noisy test dataset for k =  5  is  34.08648227502922
Accuracy on the noisy test dataset for k =  6  is  34.5539540319439
Accuracy on the noisy test dataset for k =  8  is  36.6186209583171
Accuracy on the noisy test dataset for k =  10  is  39.61823139851967
```

In [ ]:

```
np.save('/content/drive/MyDrive/Academics/Semester 1/Speech
Processing/Assignment 3/acc_arr_noisy.npy', acc_arr_noisy)
np.save('/content/drive/MyDrive/Academics/Semester 1/Speech
Processing/Assignment 3/preds_noisy.npy', preds_noisy)
np.save('/content/drive/MyDrive/Academics/Semester 1/Speech
Processing/Assignment 3/labels_noisy.npy', labels_noisy)
```

In [ ]:

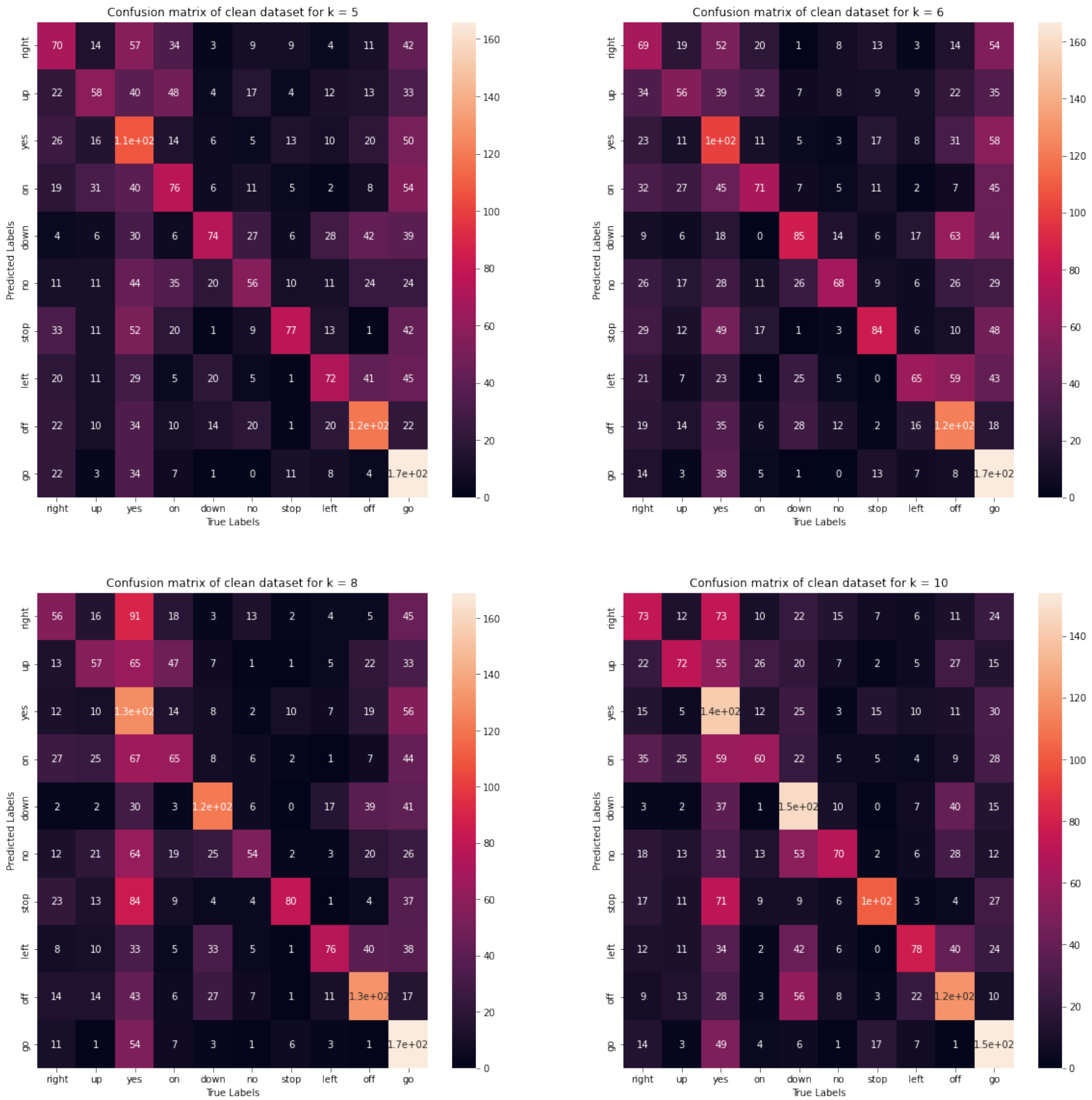
```
df_conf_mats_noisy = []
for p in range(len(preds_noisy)):
  conf_mat = confusion_matrix(labels_noisy[p], preds_noisy[p])
  df_cm_noisy = pd.DataFrame(conf_mat, index = [i for i in
list(test_noisy_dict.keys())],columns = [i for i in
list(test_noisy_dict.keys())])
  df_conf_mats_noisy.append(df_cm_noisy)
```

In [ ]:

```
plt.figure(figsize = (20,20))
plt.subplot(2,2,1)
plt.title('Confusion matrix of clean dataset for k = 5')
sn.heatmap(df_conf_mats_noisy[0], annot=True)
plt.xlabel('True Labels')
plt.ylabel('Predicted Labels')

plt.subplot(2,2,2)
plt.title('Confusion matrix of clean dataset for k = 6')
sn.heatmap(df_conf_mats_noisy[1], annot=True)
plt.xlabel('True Labels')
plt.ylabel('Predicted Labels')

plt.subplot(2,2,3)
plt.title('Confusion matrix of clean dataset for k = 8')
sn.heatmap(df_conf_mats_noisy[2], annot=True)
plt.xlabel('True Labels')
plt.ylabel('Predicted Labels')

plt.subplot(2,2,4)
plt.title('Confusion matrix of clean dataset for k = 10')
sn.heatmap(df_conf_mats_noisy[3], annot=True)
plt.xlabel('True Labels')
plt.ylabel('Predicted Labels')
```

Out[ ]:

```
Text(767.7272727272725, 0.5, 'Predicted Labels')
```



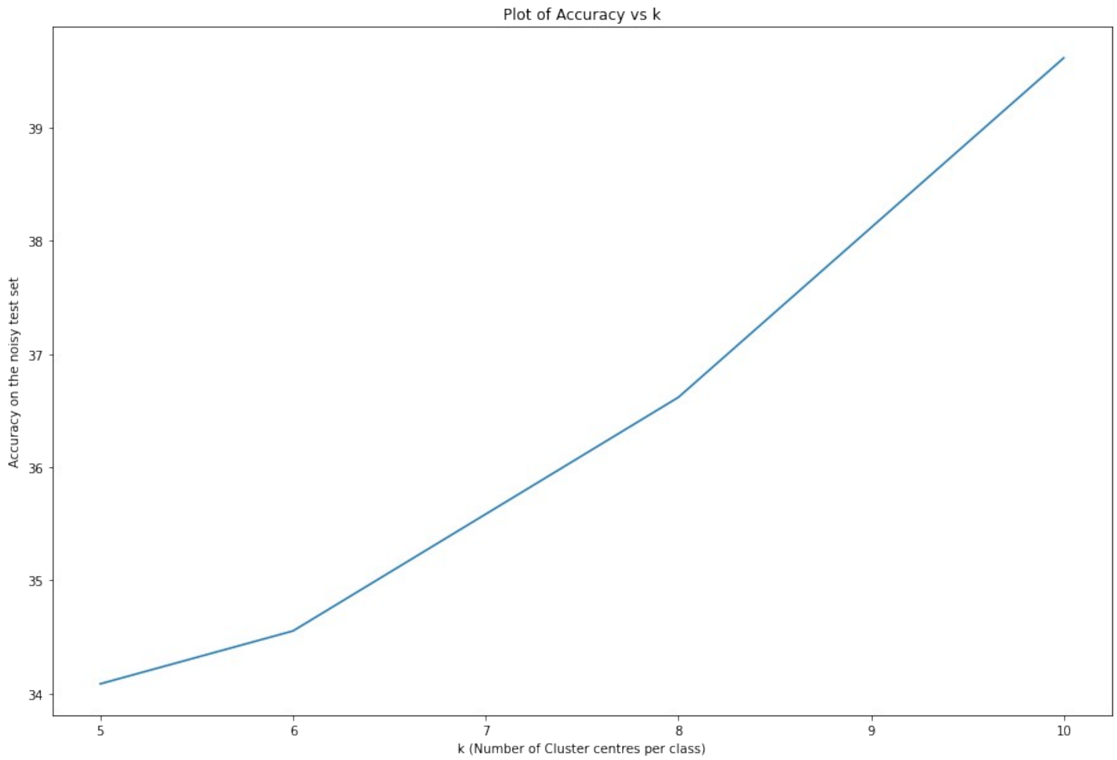
Trends for the noisy dataset are similar to those in the clean dataset, except that in the noisy dataset, the "go" class has the highest intelligibilty for the VQ codebook.

In [ ]:

```
plt.figure(figsize = (15,10))
plt.plot(k, acc_arr_noisy)
plt.title('Plot of Accuracy vs k')
plt.xlabel('k (Number of Cluster centres per class)')
plt.ylabel('Accuracy on the noisy test set')
```

Out[ ]:

```
Text(0, 0.5, 'Accuracy on the noisy test set')
```



The accuracy vs k trend is same as that for the clean dataset.

Observations and Discussion¶

Observations:

- From the graph of accuracy vs k in [5,6,8,10], we can see that the accuracy keeps increasing as we increase the number of reference vectors in each class. This increment can be observed for the clean as well as the noisy test dataset.
- From the confusion matrices, we can see that the our model gives the highest number of true postives for the "stop" sound in the clean test dataset and for the "go" sound in the noisy test datset. We can also see the true postives decreasing for some classes like "go" when we increase the value of k. Ideally, we decide upon the value of k by the number of distinct phones in the sound. So if we increase the number of reference vectors which doesn't have that many phones, the accuracy is likely to stagnate or even fall if overtraining happens.
- For other sounds, the trend of the number of true postives in general increses with k. Having more reference vectors improves the accuracy of our model, until it overfits. So we need to decide the ideal number of clusters which can represent each of the classes individually with good accuracy without over training.

Ways to Improve Existing Method¶

- Plotting the trend of accuracy vs k's for higher k values until the accuracy seems to be saturating on both the clean test and noisy test dataset. This will serve as an indicator of the optimal value of k which better represents each class.
- Adding noise to training data, and then creating the codebook for every class for different k values. Adding noise to the datset will allow our trained model to better generalize for the noisy test samples, thus getting better metrics.
- Using other methods which consider consider the sequence of phones during classfication, such as the DTW (Dynamic Time warping), and HMM (Hidden Markov Model)
- Using deep learning based sequence-classfication models like BERT. Which can utilize the sequential properties of the MFCC feature vectors and adaptive learning strategies to give better metrics.
- Getting a larger training set of vectors across diverse speakers is a trivial way to improve the existing method, but depending upon the implementation it can give good results.

Generating the HTML file¶

In [61]:

```
%cd '/content/drive/MyDrive/Academics/Semester 1/Speech Processing/Assignment 3'
```

```
/content/drive/MyDrive/Academics/Semester 1/Speech Processing/Assignment 3
```

In [62]:

```
!jupyter nbconvert --to html '/content/drive/MyDrive/Academics/Semester 1/Speech Processing/Assignment 3/213070010_Assignment3.ipynb'
```

```
[NbConvertApp] Converting notebook /content/drive/MyDrive/Academics/Semester 1/Speech Processing/Assignment 3/213070010_Assignment3.ipynb to html
[NbConvertApp] Writing 1077880 bytes to /content/drive/MyDrive/Academics/Semester 1/Speech Processing/Assignment 3/213070010_Assignment3.html
```