Indian Institute of Technology, Bombay

EE 679: Speech Processing

Computing Assignment: Automatic Speech Recognition

Submitted by:

Gourav Tilwankar 203070051

Course Instructor: **Dr. Preeti Rao**

Aim: To apply any of the studied automatic speech recognition algorithms on given data set and calculation of accuracy and confusion matrix

Procedure:

1. Mel Frequency Cepstral Coefficents (**MFCCs**)
2. Build Dataset of training data
3. Construct a model of method studied (HMM applied here)
4. Test data files and check for labels for correct output
5. Calculate Accuracy and plot confusion matrix

```
[23] from __future__ import print_function
     import warnings
     import os
     #from scikits.talkbox.features import mfcc
     import librosa
     from scipy.io import wavfile
     from hmmlearn import hmm
     import numpy as np
     import matplotlib.pyplot as plt
     import seaborn as sns
     from python_speech_features import mfcc,delta
     import pickle
     from sklearn.metrics import confusion_matrix
     import itertools
     warnings.filterwarnings('ignore')
```

Fig: importing functions

```python
def MFCC_CAL(audioname):
  #taking input
  str1 = audioname
  data , samplerate = librosa.load(str1,sr=None)
  #calculating the moving Average
  window_size = 100
  i = 0
  decimal_points = 7
  moving_averages = []
  while i < len(data) - window_size + 1:
    this_window = data[i : i + window_size]
    window_average = sum(this_window) / window_size
    moving_averages.append(window_average)
    i += 1

  #making the length of moving average signal to be equal to original signal length
  new_signal = np.zeros(len(data))
  new_signal[window_size-1:len(data)] = moving_averages

  #making a new signal from moving averaged signal multiplied by  energy of the signal at that point
  multiplied = new_signal*(data**2)

  #end-pointing
  end_point_min = 0
  for i in range(len(multiplied)):
    if (round(multiplied[i],decimal_points)>0):
      end_point_min = i
      break

  end_point_max = len(multiplied)-1
  for i in range(len(multiplied)-1,-1,-1):
    if (round(multiplied[i],decimal_points)>0):
      end_point_max = i
      break
  if (end_point_max - end_point_min>=0.08*len(data)):
    end_pointed_data = data[end_point_min:end_point_max]
  else:
    end_pointed_data = np.zeros(len(data))

  #MFCC Calculation
  mfccs = mfcc(end_pointed_data, samplerate=samplerate, winlen=0.02, winstep=0.01, numcep=13, nfilt=26, nfft=5
  d_mfcc = delta(mfccs, 2)
  d_d_mfcc = delta(d_mfcc,2)
  vector = np.hstack((mfccs,d_mfcc,d_d_mfcc))
  return vector
```

Fig: MFCC Calculations

Building DATASET

```
def buildDataSet(dir,w):
    # Filter out the wav audio files under the dir
    fileList = [f for f in os.listdir(dir) if os.path.splitext(f)[1] == '.wav']
    dataset = {}
    for fileName in fileList:
        #tmp = fileName.split('.')[0]
        label = w
        feature = MFCC_CAL(dir+fileName)
        if label not in dataset.keys():
            dataset[label] = []
            dataset[label].append(feature)
        else:
            exist_feature = dataset[label]
            exist_feature.append(feature)
            dataset[label] = exist_feature
    return dataset
```

Fig: Building data

```
[80] number_states= 7
     epochs = 300
     dimension=1
```

```
[74] def starting():
       startProbability = np.zeros(number_states)
       startProbability[0: dimension] = 1/float(dimension)
       return startProbability
```

```
[75] def transition():
       transition_mat = (1/float(dimension+1)) * np.eye(number_states)
       for i in range(number_states-dimension):
         for j in range(dimension):
           transition_mat[i,i+j+1] = 1.0/(dimension+1)
       j=0
       for i in range(number_states - dimension,number_states):
         for j in range(number_states-i-j):
           transition_mat[i,i+j] = 1.0/(number_states-i)

       return transition_mat
```

Fig:training data(a)

```
[76] from sklearn.cluster import KMeans
     def train_GMMHMM(dataset,mixes,n):
         GMMHMM_Models = {}
         dimension = 1
         transmatPrior = transition()
         startprobPrior = starting()

         for label in dataset.keys():
             #model = hmm.GMMHMM(n_components=number_states,n_mix=mixes, tra
             model = KMeans(n_clusters=n,random_state = 42,n_iter=)
             trainData = dataset[label]
             length = np.zeros([len(trainData), ], dtype=np.int)
             for m in range(len(trainData)):
                 length[m] = trainData[m].shape[0]
             trainData = np.vstack(trainData)
             model.fit(trainData)  # get optimal parameters
             GMMHMM_Models[label] = model
         return GMMHMM_Models
```

Fig:training data(b)

```
[ ] # Running this would take almost 6 hours, so do it if necessary only
    words = 'down up right left go stop no yes on off'.split()
    trainDataSet = {}
    for w in words:
      trainDir = '/content/TRAINING/'+w+'/'
      trainDataSet.update(buildDataSet(trainDir,w))
      print("Finished preparing the training data for : " + w)
      file = open('/content/drive/MyDrive/dataset/'+'traindata_'+w,'wb')
      pickle.dump(trainDataSet,file)
      file.close()

    testDataSet_clean = {}
    for w in words:
      testDir_clean = '/content/Test/test_clean/'+w+'/'
      testDataSet_clean.update(buildDataSet(testDir_clean,w))
      print("Finished preparing the testing clean data for : " + w)
      file = open('/content/drive/MyDrive/dataset/'+'test_clean_data_'+w,'wb')
      pickle.dump(testDataSet_clean,file)
      file.close()
```

Fig: creating train data

```
[36] #file = open('/content/drive/MyDrive/dataset/train_data','rb')
     file = open('/content/drive/MyDrive/train_data','rb')
     trainDataSet = pickle.load(file)
     file.close()

     #file = open('/content/drive/MyDrive/dataset/test_clean_data','rb')
     file = open('/content/drive/MyDrive/test_clean_data','rb')
     testDataSet_clean = pickle.load(file)
     file.close()

     #file = open('/content/drive/MyDrive/dataset/test_noisy_data','rb')
     file = open('/content/drive/MyDrive/test_noisy_data','rb')
     testDataSet_noisy = pickle.load(file)
     file.close()
```

Fig: uploading on google drive in form of pickle

```
#clean data testing
def do_testing(testDataSet_clean):
  success = 0
  predicted_label = []
  real_labels = []
  testclean =[]
  total =0
  label = []
  testD = '/content/Test/test_clean'
  word=0

  testdataset_clean = {}
  for key in testDataSet_clean.keys():
    testdataset_clean[key]=random.choices(testDataSet_clean[key],k=250)

  for f in testdataset_clean.keys():
    for w in testdataset_clean[f]:
      feature = w
      if len(feature)!=0:
        probs = {}
        for label in hmmModels:
          model = hmmModels[label]
          probs[label] = model.score(feature)
        result = max(probs,key = probs.get)
        predicted_label.append(result)
        real_labels.append(f)
      if (result == f):
        success += 1
```

Fig: testing function

```python
import random
data = {}
for key in trainDataSet.keys():
  data[key]=random.choices(trainDataSet[key],k=2500)
#number_states,mixes,epochs
nlist=[]
acc=[]
hmmModels = train_GMMHMM(data,3,5)
acc,success,predicted_label,real_labels=do_testing(testDataSet_clean))
#print("Finish training of the GMM_HMM models")
```

```
0.631
```

Fig: testing for clean data

```python
acc,success,predicted_label,real_labels=do_testing(testDataSet_clean)
different_words = testdataset_clean.keys()

#plotting Confusion Matrix for clean data
conf_mat1 = confusion_matrix(real_labels,predicted_label)
conf_mat1 = conf_mat1.astype('float')/conf_mat1.sum(axis=1)[:,np.newaxis]
#print(conf_mat1)

print("Accuracy: "+str(acc))
plt.figure(figsize=(10,10))
plt.imshow(conf_mat1,cmap=plt.cm.Blues)
plt.colorbar(cmap='Blues')
plt.title('CONFUSION MATRIX for clean data')
plt.xticks(range(len(different_words)),different_words,rotation=0.60)
plt.yticks(range(len(different_words)),different_words)

for i,j in itertools.product(range(conf_mat1.shape[0]),range(conf_mat1.shape[1])):
  plt.text(j,i,format(conf_mat1[i,j], '.2f'), horizontalalignment = "center",color="white" if i==j else "black")

plt.tight_layout()
plt.ylabel('Correct Label')
plt.xlabel('Predicted Label')
plt.show()
```

Fig: plotting data for clean

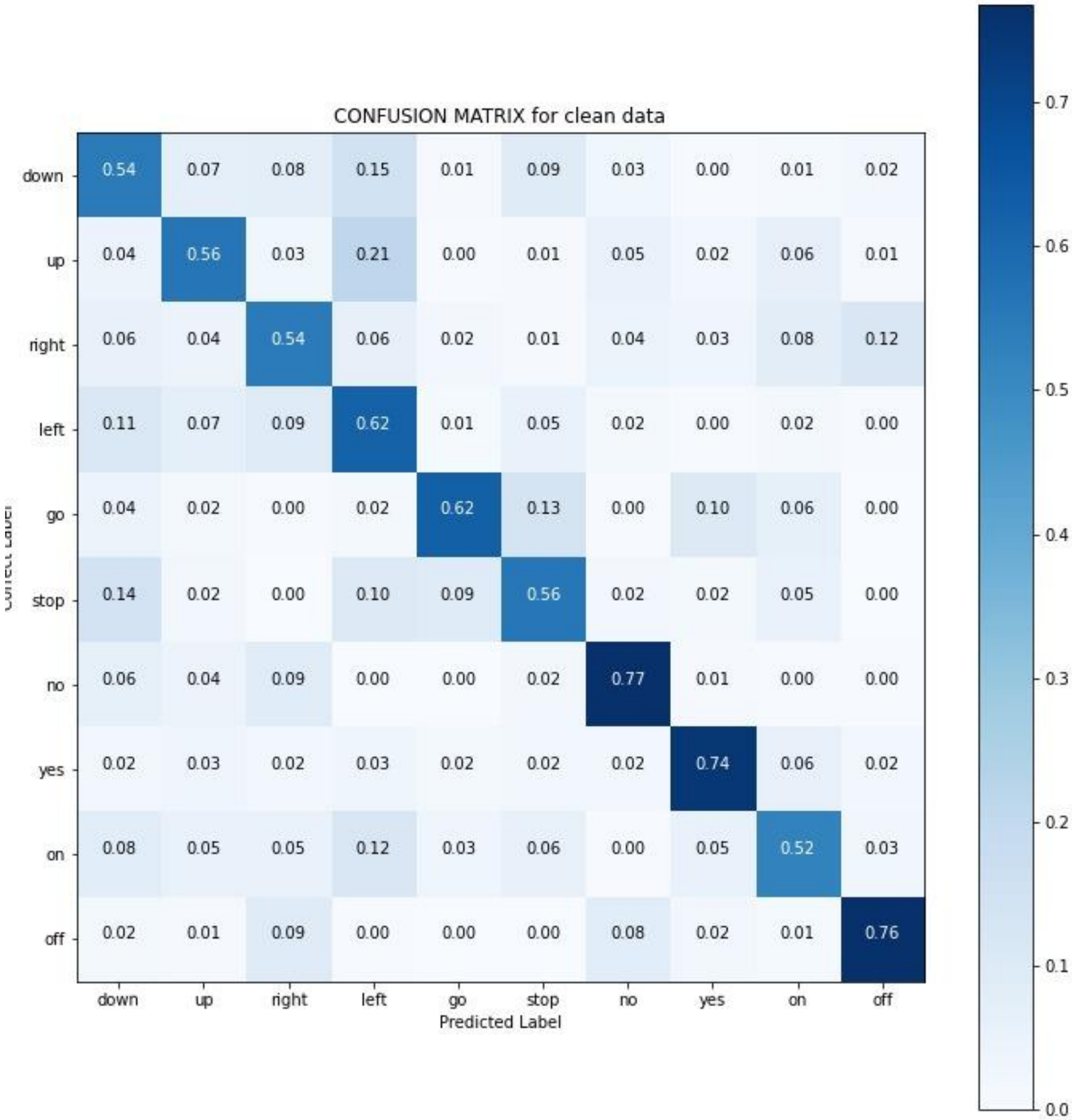

Fig: Confusion matrix for Clean data with accuracy of 63.1%

```
#noisy data testing

success_n = 0
predicted_label_n = []
real_labels_n = []
testnoisy =[]
total_n =0
testD_n = '/content/Test/test_noisy'
word=0

for f in testDataSet_noisy.keys():
  for w in testDataSet_noisy[f]:
    feature = w
    if len(feature)!=0:
      probs = {}
      for label in hmmModels:
        model = hmmModels[label]
        probs[label] = model.score(feature)
      result_n = max(probs,key = probs.get)
      predicted_label_n.append(result_n)
      real_labels_n.append(f)
    if (result_n == f):
      success_n += 1
    else:
      pass
  print(f'testing {f} is done')
```

Fig: testing for noisy data

```
acc,success_n,predicted_label_n,real_labels_n=do_testing(testDataSet_noisy)
different_words = testDataSet_noisy.keys()

#plotting Confusion Matrix for clean data
conf_mat2 = confusion_matrix(real_labels_n,predicted_label_n)
conf_mat2 = conf_mat2.astype('float')/conf_mat2.sum(axis=1)[:,np.newaxis]
#print(conf_mat1)

print('Accuracy is : '+str(acc))
plt.figure(figsize=(10,10))
plt.imshow(conf_mat2,cmap=plt.cm.Blues)
plt.colorbar(cmap='Blues')
plt.title('CONFUSION MATRIX for clean data')
plt.xticks(range(len(different_words)),different_words,rotation=0.60)
plt.yticks(range(len(different_words)),different_words)

for i,j in itertools.product(range(conf_mat2.shape[0]),range(conf_mat2.shape[1])):
  plt.text(j,i,format(conf_mat2[i,j], '.2f'), horizontalalignment = "center",color="white" if i==j else "black")

plt.tight_layout()
plt.ylabel('Correct Label')
plt.xlabel('Predicted Label')
plt.show()
```

```
Accuracy is : 0.4732
```

Fig: plotting data for clean

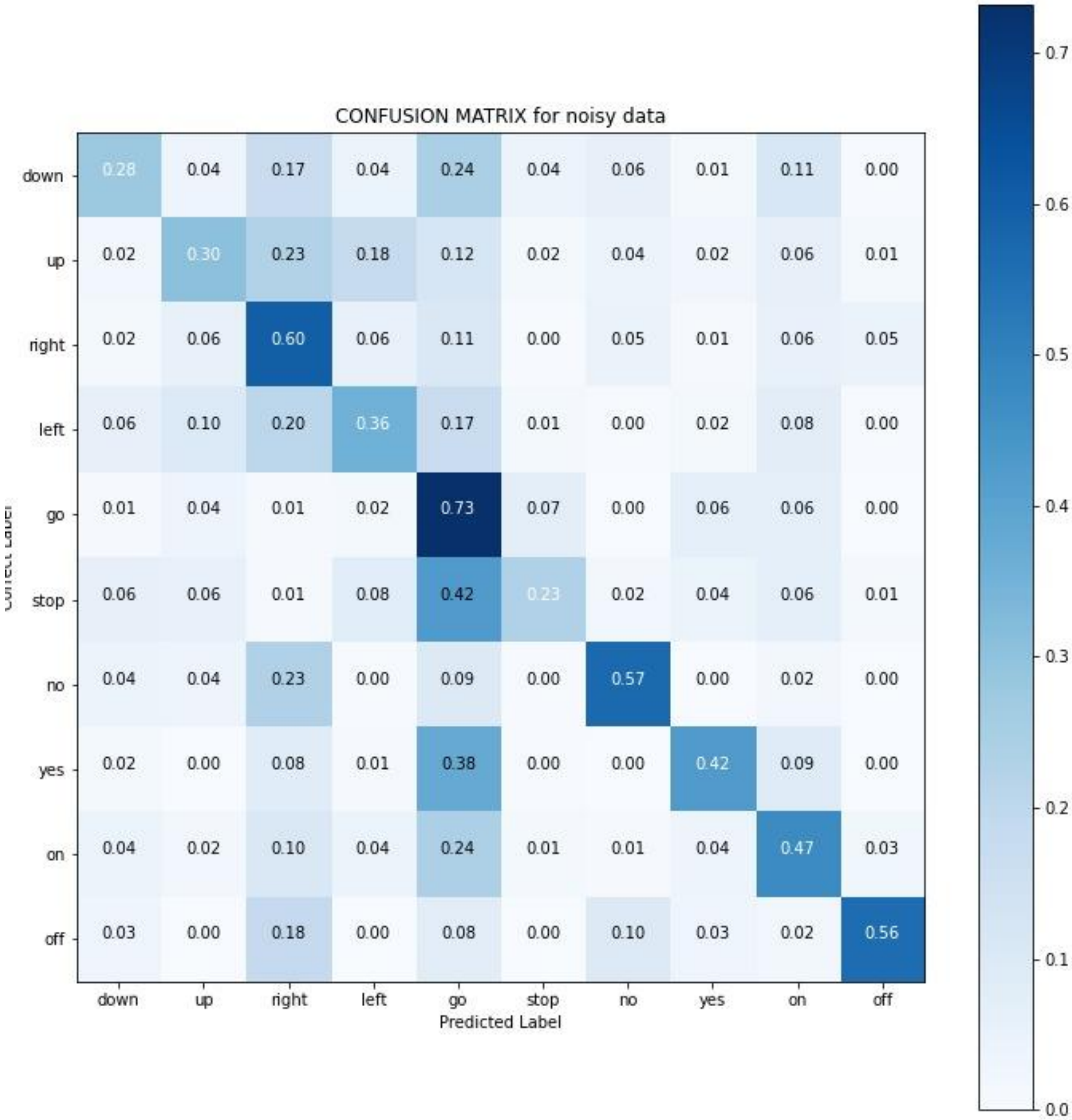

Fig: Confusion matrix for noisy data with accuracy of 47.32%