EE 679: Computational Assignment 3

Automatic Command Recognition

Archishman Biswas, 180070009

November 18, 2021

Contents

1 Command Recognition Problem

The aim is to recognise the command words in a subset of Google Speech Commands Dataset version 0.01. Given a large set of training audio files, we are required to come-up with a model and train it such that given a test audio file we can predict the word with good enough accuracy.

2 Training and Testing Data

The subset of the Google Speech Commands Dataset version 0.01 contains audio files in .wav format for each of the command word utterances. Sampling frequency is $F_s = 16\ KHz$, duration of each recording ≤ 1 *second* and they are stored as 16-bit PCM(signed integers). There are a total of 10 words/classes, namely: "down", "go", "left", "no", "off", "on", "right", "stop", "up", "yes".

2.1 Training Dataset

Around 2100 recorded instances for each pf the words/classes. Thus, we have around 21000 total audio files in the training dataset. Also, 6 noise files(each of approximately 1 minute duration) are given of different kinds which can be used for data augmentation purpose.

2.2 Testing Dataset

For the testing part of the data, we have a total of around 250 utterances for each word/class. Thus we have around a total of 2500 audio files in the test dataset.

Task A: Recognition of clean speech utterances. The testing audio files doesn't contain any additional noise(only ambient noise is present).
Task B: Recognition of noisy utterances. Contains the same utterances as in clean audio of task A, but here about half of the audio has added 10 dB noise.

3 Data Augmentation, Pre-processing and Feature Extraction

In this section, we discuss two of the used data-augmentation technique: noise injection and audio speed change. Also, two pre-processing techniques used is discussed: pre-emphasis and end-pointing.

3.1 Noise Injection

A noise signal is added to the speech audio. The SNR is uniformly chosen from a range that can be chosen by the user. The output set of audio files will then have a modified version of the original audio signals.

3.2 Audio Speed Change

Inbuilt librosa function for speed rate change is used. The speed rate is chosen uniformly from a range of speed rate that can be chosen by the user. Speed rate is chosen between 0.8 to 1.2(1 being same speed) for experiments shown in later sections.

3.3 Pre-emphasis

This is done for each of the audio in order to mitigate the effect of glottal spectral roll off. The basic equation is:

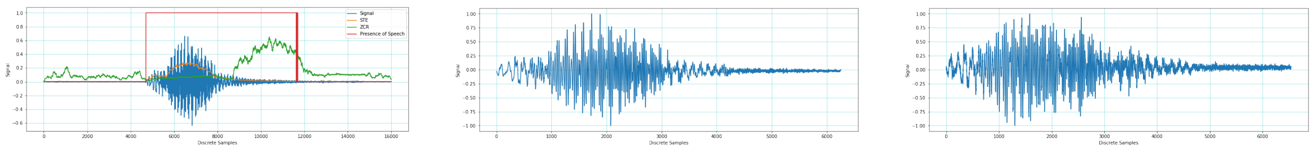
$$y[n] = x[n] - \alpha x[n-1]$$

It acts as a high pass filter, the parameter $\alpha = 0.99$ is chosen for experiments shown in later sections.

3.4 End-pointing

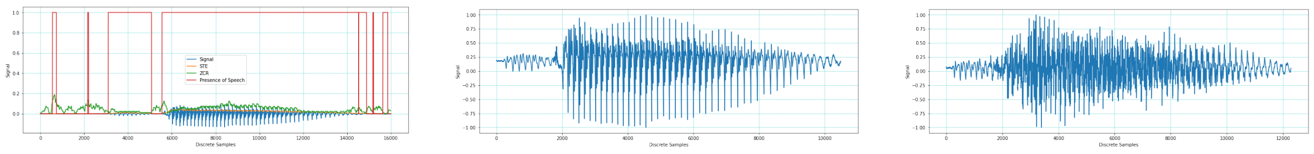
For end-pointing, both the STE and ZCR values are utilized. The computation is done via librosa in-built functions. When the STE or ZCR is above certain percentage of their respective maximum value, the signal is taken to be present. Other-wise it is taken as being absent, i.e. only silence.

The plots given below shows how the above functions work for three different words:



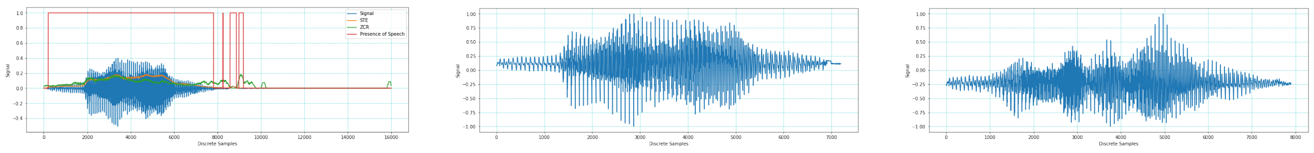
(a) Original audio (b) Audio after end-pointing (c) Speed and noise augmentation

Figure 1: Word = "Yes"



(a) Original audio (b) Audio after end-pointing (c) Speed and noise augmentation

Figure 2: Word = "Down"



(a) Original audio (b) Audio after end-pointing (c) Speed and noise augmentation

Figure 3: Word = "No"

3.5 Feature Extraction

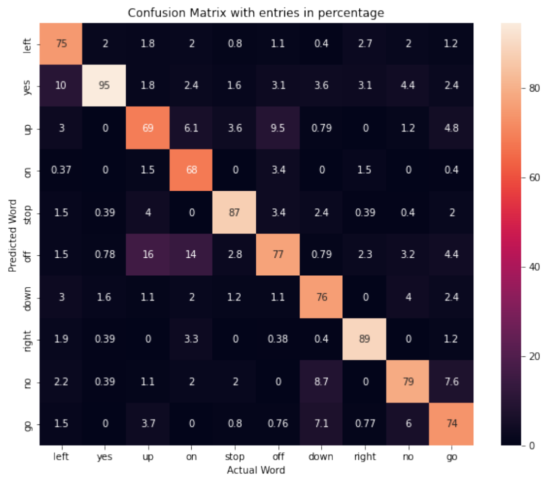
For the feature extraction part, first 13 MFCC coefficients are computed with a hop length of 10 ms and window length of 20 ms. Combined with the delta and delta delta features, we get 39 length feature vectors for each frame. These are stored in a long vertical matrix, one such matrix for each word. The matrix are collected in a list which is stored as a .npy file.

4 Method 1: Vector Quantization(Bag of Words)

For the VQ method, the MFCC features of each of the word has been clustered into total L representative vectors called centroids. Then, for each of the test utterances, the sum of minimum distortion is calculated w.r.t. each word. The word providing the minimum value is declared the predicted word. The table below shows the accuracy results for different number of centroids chosen:

Centroids (L)	Accuracy(clean,noisy)
16	(76,56)
32	(79,57)
64	(81,57)
128	(77,58)

From the above example, we can see that as we increase L, we get increment in both clean and noisy test accuracies. But, the noisy accuracy doesn't increase beyond 58%. For L = 128, the performance is worse than L = 64. The confusion matrices(both clean and noisy) for the case of L = 32 and L = 64 are shown below:


(a) Confusion matrix for testing on clean data

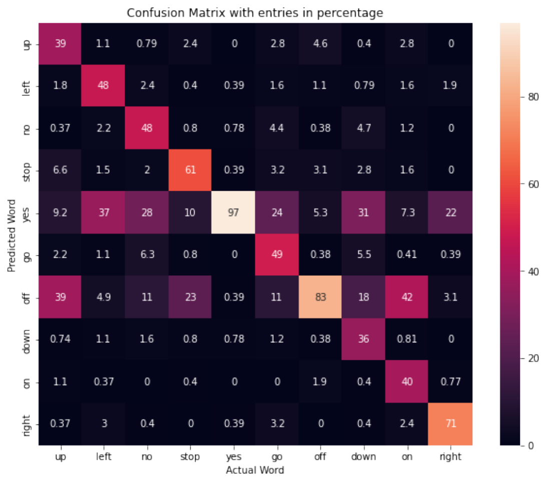
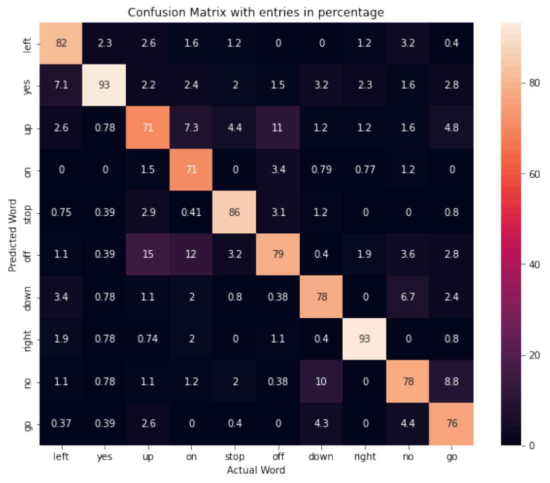

(b) Confusion matrix for testing on noisy data

Figure 4: Results for VQ with L = 32


(a) Confusion matrix for testing on clean data

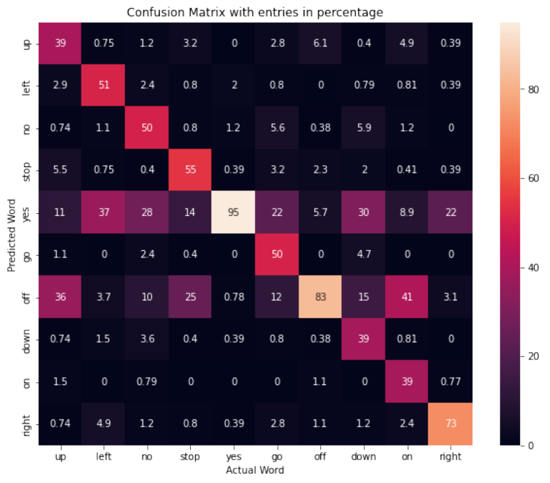

(b) Confusion matrix for testing on noisy data

Figure 5: Results for VQ with L = 64

Increasing L beyond 128 would be impractical as the memory of requirement code-book will be very high. Also, we can't expect the clean accuracy to increase anymore. Due to the low accuracy of baseline, we don't carry out data-augmentation in this and move on to implement GMM-HMMs as HMMs will learn the temporal/sequential variations in the data too.

5 Method 2: Word Level GMM-HMM

The previous VQ algorithm didn't take into account the sequential information of the MFCC vectors. This lead to poor classification accuracy. Using a HMM, we can encode the sequential information into our model. Hence, in this section we discuss the results obtained on using a GMM-HMM model.

The two major parameters that can be changed in the GMM-HMM are the number of states in the HMM(N_s) and the number of Gaussian mixtures(N_m).

5.1 Training Without Data Augmentation

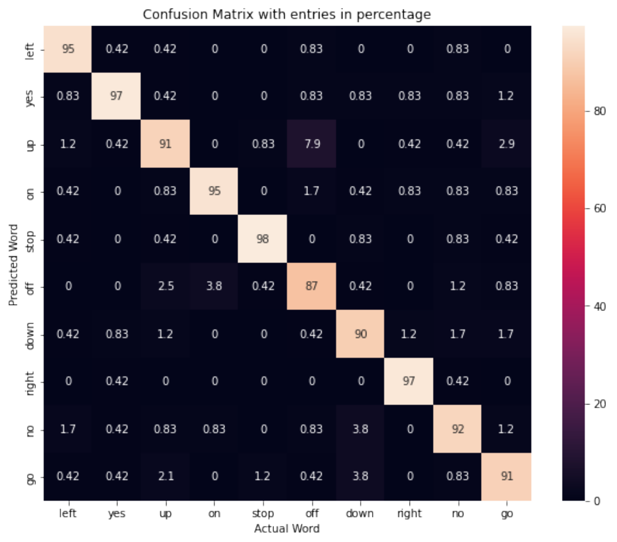
Training using data without any augmentation were carried out for different combination of N_s and N_m. The results are tabulated below:

N_s,N_m	Accuracy(clean,noisy)
(8,4)	(86,64)
(8,8)	(88,64)
(12,8)	(90,64)
(16,8)	(91,64)
(20,10)	(93,69)

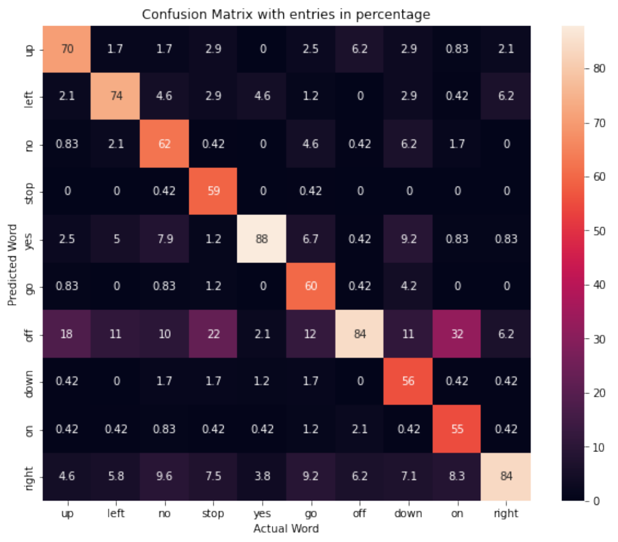
From the above table, we can observe:

- The accuracy has increased significantly as compared to the VQ method, where accuracy was well below 85% for clean audio and well below 60% for noisy audio
- Increasing the N_s or N_m is improving the performance of the model. Increasing N_s above 15 gives quite good accuracy(above 90%) on the clean test data
- Despite the good performance on the clean data, the performance on noisy data is poor(maximum of 69 %). This can be improved by doing data augmentation, such that our model will be more generalized but the cost will be drop in clean test accuracy

The confusion matrix for the best accuracy model of the above, i.e. for (N_s=20,N_m=10) is shown below for both clean test data and noisy test data:



(a) Confusion matrix for testing on clean data



(b) Confusion matrix for testing on noisy data

Figure 6: Results for GMM-HMM with N_s=20 and N_m=10

From the confusion matrix above, we observe:

- The words "yes","stop","right" have the best accuracy. Whereas words "down" and "off" poor accuracy
- In the clean data, we can observe that most of the classification error is occurring due to misclassification of "off" into "up" and "up" and "on"to "off". It is hard for the model to distinguish up and off
- For the noisy test data, many of the words are wrongly predicted as "off". This can be explained due to presence of fricative at end of "off". Hence the noise is being mistaken as fricative "f" by the trained model.

5.2 Training With Data Augmentation

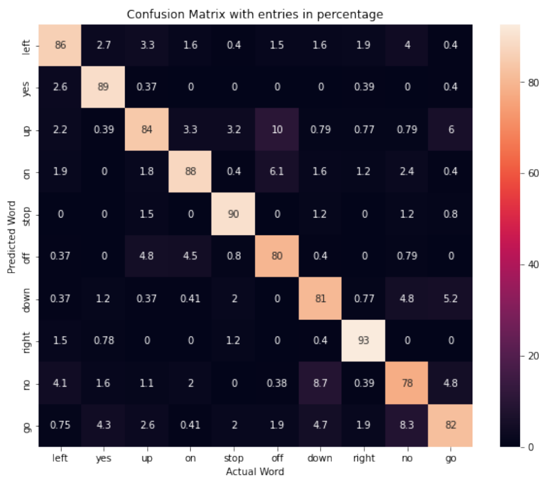
We now train GMM-HMMs using augmented data in hope that the accuracy on the noisy test data will increase at the expense of little drop in clean test data accuracy. The results for different N_s and N_m are tabulated below:

N_s,N_m	Accuracy(clean,noisy)
(8,4)	(85,68)
(8,12)	(89,71)
(16,12)	(85,73)

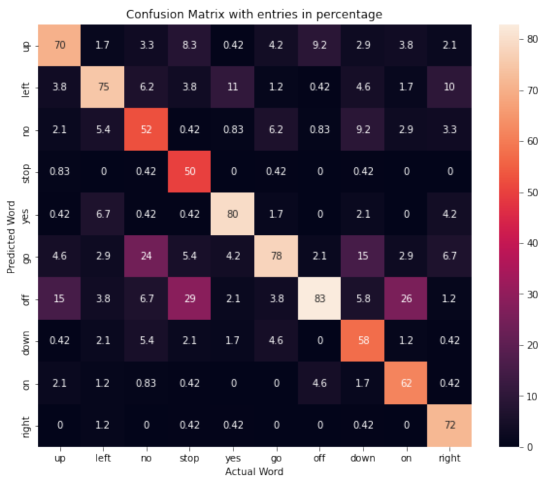
From the table given for the 3 cases in which the augmented data(noise added and speed of utterance changed a little) are used, we can observe the following:

- Even for a lightweight model of $N_s = 8$ and $N_m = 4$, we see that the noisy accuracy has increased quite a lot as compared to previous case(by 4%) via using noise injection and utterance speed changing.
- If we use a model with more parameters as in $N_s = 8$ and $N_m = 12$, we get much better clean accuracy than $N_m = 4$. This is because the mixtures of Gaussians are now able to model the actual distribution quite well.
- **Note:** For the $N_s = 16$ and $N_m = 12$ case, the data chosen is only from the modified one and the actual data is not chosen. So the training data size is effectively half. This is done as the training time was too high otherwise. So we see that the noisy test performance is better but there is a drop in the clean performance.
- If more training time is available, more training data modifications can be made and the model can be trained to be a robust one both for noisy and clean test.

The confusion matrix for the three cases of augmented data are shown below:

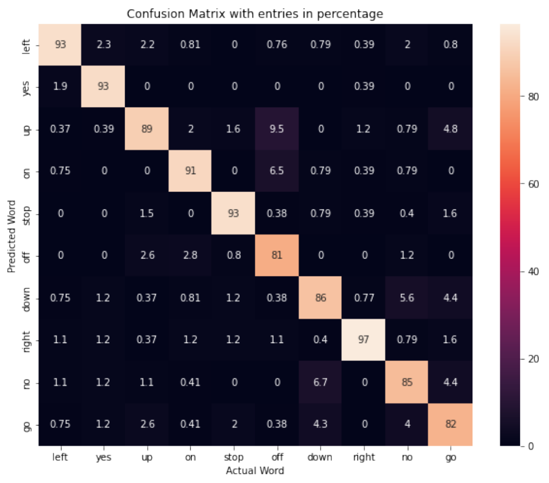


(a) Confusion matrix for testing on clean data

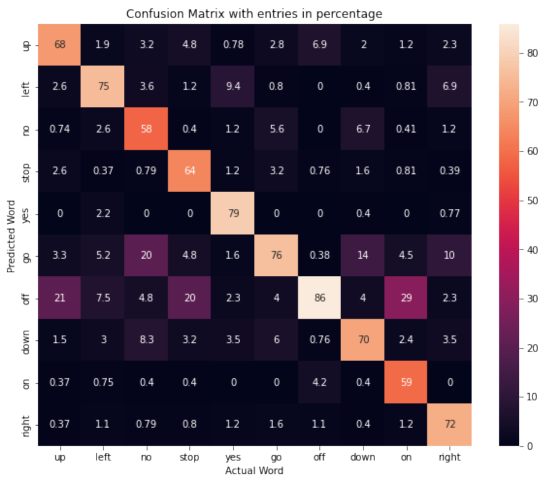


(b) Confusion matrix for testing on noisy data

Figure 7: Results for augmented data GMM-HMM with $N_s = 8$ and $N_m = 4$

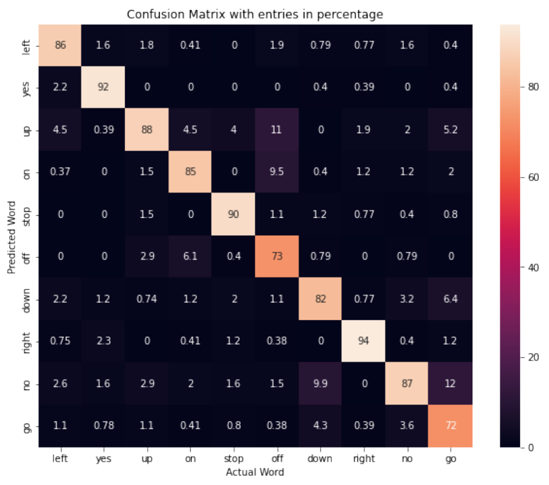


(a) Confusion matrix for testing on clean data

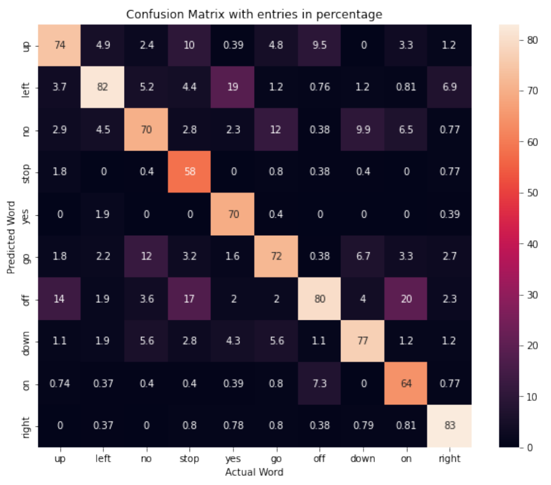


(b) Confusion matrix for testing on noisy data

Figure 8: Results for augmented data GMM-HMM with $N_s = 8$ and $N_m = 12$



(a) Confusion matrix for testing on clean data



(b) Confusion matrix for testing on noisy data

Figure 9: Results for augmented data GMM-HMM with $N_s = 16$ and $N_m = 12$

From the confusion matrices above, we observe the following:

- From the clean audio experiments of all the 3 above cases, we can see that the models struggles a bit to differentiate between the triplets: ”off”,”on”,”up”. The reason is the very similar vowel part.

- Another such triplet is ”no”,”go”,”down”. Again this is mainly due to the similarity in the vowel part. Most of the utterance of these three words is focused on the vowel part leading to this confusion.

- For the noisy cases, we see that the confusions of the clean cases are just getting magnified. This can be seen both in the case of triplet ”off”,”on”,”up” and triplet ”no”,”go”,”down”

- Also, new confusions are arising in noisy case, like in pair ”yes”,”left” and pair ”stop”,”off”

- The words with high ease of prediction are ”right”, ”stop”, ”yes” for clean data and ”left”, ”off”, ”right” for noisy data.

6 Conclusion

We have explored two methods namely: **Vector Quantization** and **GMM-HMM** for the problem of automatic command recognition. We have seen that for reasonable training time(around an hour, less than 2 hour for sure) the maximum accuracy in the case of VQ method is 81 % on clean data and 58 % on noisy data. For GMM-HMM it is much better, around 93 % on clean data and 73 % on noisy data. Thus, we can conclude that GMM-HMMs are much superior compared to VQ method. More heavyweight(more parameters) models required much greater time to train and also will required more memory in storage. The confusion matrix also gave us an idea about which command words are better to use, i.e. have lower confusion even in noisy environment. These matrices can be used to decide which words to choose during deployment of real world command recognition systems.