EE679 | Computing Assignment 5

Ashwin Kachhara | 10d070048

Method

1. **Segmentation**: (`seg.m`)
   First we extract the digits from the given data. So we have 16 instances of each digit as per the training data. The segmentation is done through [1].
   We could choose to use this completely as training data or choose a portion of this as test data.
2. **Extracting the Features**: (`wincepstrum.m`)
   Now that we have segmented the digits, we need to extract data useful for the recognition purpose. We will use cepstral coefficients of windowed segments of the digit. Thus we have as many sets of coefficients as the num of windows that fall into the segment of speech.
   a. Before anything, we **pre-emphasize** the speech signal with a factor of 0.975 to emphasize the higher frequency energy in speech
   b. Following that, we window the entire signal with **20ms Hamming** windows (non-overlapping). If k such windows cover the length of the signal, then we have k 'frames'.
   c. For each frame, we compute the real cepstrum C and use it to construct a feature vector of the frame. The feature vector is **39 dimensional**. The first 13 elements are the first 13 cepstral coefficients. The next thirteen are **Δ(i)** = C(i+1)-C(i) and the last thirteen elements of the feature vector are **ΔΔ(i)** = Δ(i+1)-Δ(i).
      Now we have 39 dimensional feature vectors for each digit utterance.
3. **Constructing the Codebook (Training)**: (`traincodebook.m`)
   We will construct codebooks for each digit, which will contain representative feature vectors called centroids for that digit.
   a. Let our base codebook $\mathbf{CB_{-1}}$ be the feature vectors of the first digit utterance (or could be any, chosen at random).
   b. Iteratively, for all other training utterances, for each frame, we find the nearest centroid in CB0 and group that frame with it. By 'nearest', we mean in a **Euclidean distance** sense, i.e. the least Euclidean distance between the feature vectors. This is called **clustering**.
   c. Now that we have clustered the nearest feature vectors, we compute the **mean feature vector** of each cluster and name that as the new centroid.
      These new centroids constitute the Codebook $\mathbf{CB_0}$ that we will hereafter use. We have now constructed one such codebook for each of the digits in the vocabulary (0-9).
4. **Testing**: (`test.m`)
   Testing is much like what we have done till now.
   a. We extract the 39dim feature vectors for each frame in the test utterance
   b. For each codebook, we find the nearest centroid in the particular codebook and compute the **average distance** of all frames with nearest centroids in the codebook.

c. We have 10 such avg distances for each codebook. Logically, the codebook **minimum avg distance** is the digit our test utterance is predicted to be.

Observations

I tried two different combinations of test-train data

1. Training – first 15 utterances of each digit. Test – 16th utterance of each digit.
   Detecting 'seven' instead of 'five'; 'nine' instead of 'three'; 'one' instead of 'two'.
   So, that turns out to be 70% accuracy for test data.
   And 100% accuracy if we test with any utterance from the training set.
2. Training – All 16 utterances of each digit. Test – A recoding of my own voice.
   Correct detection of digit is seen only for 'four', 'eight'. However, the distance of the correct digit is 2nd or 3rd lowest in each case. So, finally an accuracy of 20% when using my own voice signals.

Results

So, for one case (using the 16th utterance as test) we are getting very promising accuracy of 70% considering the relative simplicity of the model, while in the second case (using my voice), we are getting very poor accuracy of 20%.

Quite obviously, with a larger and more exhaustive training set, the accuracies could have been much better. Within limits, the model is as good as the way it is trained.

Apart from that, the reason of poor accuracy in the second case is likely due to the following factors:

- Different recording equipment
- Environmental effects (noise etc)
- Any other processing that may have been done to the given signals

So, I think proper noise cancellation/reduction is extremely necessary before applying any kind of speech recognition algorithm onto a speech signal. Plus, more exhaustive training sets i.e. covering all the ways a person may speak a word, is necessary to improve the accuracy of any model.

References

[1]: *Theodoros Giannakopoulos*; A method for silence removal and segmentation of speech signals, implemented in Matlab

A brief note on directories/files

- Directories ‘zero’, ‘one’. . . ‘nine’ contain the extracted utterances from the given data.
- Directory ‘test’ contains the segments extracted from my own voice ‘digit.wav’
- centroidzero.mat, centroidone.mat, ... centroidnine.mat are the codebooks saved from traincodebook.m
- Codes: seg.m, test.m, traincodebook.m, wincepstrum.m