EE 679 computing assignment 5: Digit recognition

Khushal Kharade 10d070023

A template based digit recognition is developed with the help of k-means algorithm. I extracted all the utterances of a digit using PRAAT and saved them in a folder. The utterances are extracted such that there as less silence as possible. Silence removal is not done.

The data organization is done as follows. Folder 'D:\…\0' contains all the 16 utterances of `zero` by 4 speakers, folder 'D:\…\1' contains that of `one` and so on. The utterances are numbered from 1-16 in the same order as that of speakers i.e. 1-4 are of speaker1, 5-8 are of speaker2 and so on.

Six functions are written, namely *'feature_extraction.m'*, *'create_codevector.m'*, *'create_codebook.m'*, *'digit_recognizer.m'*, *'distance_Ndim'*, *'get_accuracy'* to implement the problem. Each of the functions description is given below.

Steps in *'feature_extraction'*

- It takes a sound/utterance vector `sound_in` as input.
- pre-emphasis is done by applying a filter `P(z)= 1-0.95*z^(-1).`
- Hamming window of length 20ms corresponding to 161 samples is chosen for windowing the `sound_in` vector in frames. `frame_size = 161`.
- No. of frames are computed as `length(sound_in)/frame_size.`
- Hamming window is applied over all the frames and 1024 point FFT is computed for each. Then for each frame cepstral coefficients are computed as `real_cepstrum_WinPre = real(ifft(log10(abs(FFT_WinPre))));` here `WinPre` is windowed and pre-emphasized signal vector. Only first N coefficients are taken as representative to each frame.
- It returns matrix `cepst_coeff` of size `N x No_of_frames;`

Steps in *'create_codevector'*

- It takes an input digit `num` and returns its codevector.
- It computes matrix `cepst_coeff` for each of the training data i.e. '.wav' file of selected utterances of digit `num` out of all 16, by iteratively calling the function *'feature_extraction'* and appends them all to form a data matrix `w` of size N x total_no_frames.
- A built-in function for k-means is used as follows: `[idx, C] = kmeans(w',clusters);` `clusters =` number of clusters to be formed from all data points in data matrix `w'`.
- It returns a *matrix* `C` (code_vector) of size `clusters x N` corresponding to digit `num.`

Steps in '*create_codebook'*

- This functions just passes a digit from 0-9 to *create_codevector* and saves each created matrix (code_vector) into same directory with name CB0.mat, CB1.mat and so on.

Now the codebook is ready but the actual recognition step is yet to be performed.
So far, we have N dimensional acoustic space and there are `clusters` number of points for each digit in this space. The same is done to compute `clusters` number of points for the test/recorded utterance.
And the '`distance_Ndim`' is defined to compute the distance of cluster points in test utterance from the points in cluster of each digit.

Steps in '*digit_recognizer*'

- The utterance which is to be recognized is given as the input. All the codebooks are imported from the same directory which will be used in recognition process.
- Then its data matrix is formed using *feature_extraction.* And k-means is applied to form its clusters in `test_C`.
- Since the cluster points formed by k-means are not in any specific order, we have to find distance of each point in one cluster with every other point in another cluster. Distance is computed as `d0(i,j) = distance_Ndim(test_C(i,:),ref0(j,:));`
  Here d0(i,j) will contain distance of ith point in cluster of test_C from jth point in cluster of digit zero.
- Now the minima `dist0,dist1 etc.` of each matrices `d0, d1, d2 etc.` is stored in a vector `dist.` And now the minimum of this vector will represent the nearest reference vector from the test vector.
  ```
  [~,digit] = min(dist);
  digit = digit-1;
  ```
  The `digit` is then returned as the recognized digit.

Description of '*get_accuracy'*
Now, accuracy of the developed application is computed as follows. we have total of 160 utterances of 10 digits, out of which half(80) are used for training and half(80) will be used for testing. Even numbered utterances are used for training and odd are for testing. Accuracy is computed by comparing the result of the digit_recognizer with that of original digit. This is done for all 80 test utterances. E.g. if all the utterances are correctly recognized count will be 80 and hence accuracy will be 100%.

Observations and Results:
Initially, just to play safe, I tested with the same data I used for training. That gave me 100% accuracy. But later when I was testing with my recorded voice, I found that the method gives wrong results. Then I tried using first 8 utterances for training and later 8 for testing. Again I found that the method fails. But when I used 2 utterances of each speaker for training and other 2 for testing the method works great. So, I concluded that this is all speaker dependent. And the reason for this is that we are using very few (only 8) utterances for training and those of only 4 speakers. So, if we increase the number of speakers for creating codebook and then we will be able to extract many different characteristics/ways of pronouncing that particular digit.
Next I tried varying N and computed the accuracy N =13, 20, 26, 39. The result is shown in table below for 12 `clusters.`

Acoustic space dimensionality N	Accuracy (in %)
13	92.5
20	93.75
26	93.75
39	95

We can see that there is not much variance in accuracy from 13 to 39. For 39 dimensionality of acoustic space we are getting highest accuracy, so I have chosen N=39.
Later, I tried changing the number of clusters formed by k-means from `clusters=` 4, 6, 8, 10, 12. I found that as the number of clusters are increased the method gives more accurate results shown in table below for N = 39.

Number of clusters	Accuracy (in %)
4	70
6	83.75
8	86.25
10	92.5
12	95

Also, it seems reasonable because what k-means does is randomly clustering the data points by iteratively combining few points, to minimize the mutual distance and finding centroids of each cluster. So, as there are more number of clusters we don't lose on some features. Meaning, by falsely putting a point in certain cluster which instead could belong to another if there had been more clusters. This would give us more cluster points creating varieties in distance measures among digits to improve accuracy.