CS6300 Speech Technology: Assignment 4 Report

-**Team 7:** Akash Reddy A, EE17B001 and Nikhil Mattapally, EE17B138

1 Dynamic Time Warping-based Recognition

Dynamic Time Warping is a template-based technique to match two time series which have different sizes, a test sequence and a template sequence. We can find similarity of two sequences which may be of different lengths. Here we are using DTW to match the feature vectors of each audio file to check if both are similar (having same word) or not.

Initially we have extracted Cepstrum and deltaCepstrum features using the given executable `ComputeFeatures` and file `mfcc.config`, for each audio file in the train and test data. The DTW distance between any test sample and all train samples is evaluated as follows:

- We first initialise a DTW matrix between test sample and train sample as shown below. As an example, f1, ..., f5 are the feature vectors of the test sample. f1', ..., f7' are feature vectors of the train sample. The DTW distance between f1 and f1' is initialised as the cost between these two vectors $c(f1, f1')$. We have chosen the cost function to be the square of the Euclidean distance or L2-norm between the two feature vectors. All other cells are initialised to ∞.

	f1'	f2'	f3'	f4'	f5'	f6'	f7'
f1	$c(f1, f1')$	∞	∞	∞	∞	∞	∞
f2	∞	∞	∞	∞	∞	∞	∞
f3	∞	∞	∞	∞	∞	∞	∞
f4	∞	∞	∞	∞	∞	∞	∞
f5	∞	∞	∞	∞	∞	∞	∞

- We now iterate through all columns of each row, row by row, setting the value of $D_{i,j} = min(D_{i,j-1}, D_{i-1,j-1}, D_{i-1,j}) + c(fi, fj')$.
  After doing this for the first row, and approaching the second row, we have a table that looks like:

	f1'	f2'	f3'	f4'	f5'	f6'	f7'
f1	$c(f1, f1') \searrow$	$D_{1,2} \downarrow$	$D_{1,3}$	$D_{1,4}$	$D_{1,5}$	$D_{1,6}$	$D_{1,7}$
f2	$D_{2,1} \rightarrow$	$\mathbf{D_{2,2}}$	∞	∞	∞	∞	∞
f3	∞	∞	∞	∞	∞	∞	∞
f4	∞	∞	∞	∞	∞	∞	∞
f5	∞	∞	∞	∞	∞	∞	∞

  where $D_{2,2} = c(f2, f2') + min(D_{2,1}, c(f1, f1'), D_{1,2})$, for instance. We continue to evaluate until $D_{5,7}$, or the bottom-right corner of this table.

After filling up the entire table, we get the DTW distance value between the two samples as $D_{N,M}$, where N is the number of features in the test sample and M is the number of features in the training sample. Here, the value of the DTW distance between the two samples required is $D_{5,7}$.

Steps to check which class a sample belongs to:

1. Match the test sample with each training sample and store the DTW distances.
2. Find the 20 training samples with smallest DTW distances, and note class of each of these training samples.
3. Declare the class of the test sample by majority voting from the best 20 training samples.

After running this test for all the 60 test data samples we obtained 56 correct, which results in an accuracy of 93.33%. Most errors were for words 'five' and 'zero'.

2 Discrete HMM-based Speech Recognition

This method of speech recognition uses statistical modelling to classify the recording into a word. The word with the highest likelihood of generating the observed sequence of feature vectors is returned as the output.

This likelihood for each word W is generated by estimating the Hidden Markov Model from the feature vectors of all the train examples of W. Since speech is a quasi-stationary process, it can be assumed that throughout each window in which any recording can be assumed to be stationary, it remains in the same state of a Markov Model, after which it transitions into the next state. However, these states and transition probabilities are not known to us, and we have to estimate the parameters of the Markov Model from the training examples that we have.

Since the HMM is discrete in our modelling, we need to make sure that the emitted observations from the feature vectors are discrete. By performing MFCC feature extraction, we obtain continuous real-valued (floating-point) vectors as a feature vector. In order to map each vector to a discrete output, we perform K-Means clustering, and then train our HMM models using these discretised outputs.

Steps:

1. The feature vectors of all recordings of all digits are initially taken together (in our assignment, the feature vectors are 38-dimensional), and K-Means clustering is performed on them. This way, each continuous-valued 38-dimensional feature vector get mapped to a single integer, the cluster index. Since each feature vector belongs to a certain phone, we have used the total number of phones to empirically choose the number of clusters that they belong to. We have:

   - two = /t//u/
   - three = /th//r//i/
   - four = /f//o//r/
   - five = /f//ai//v/
   - zero = /z//i//r//o/

   and there are 10 unique phones. A choice of 16 clusters makes sense for isolated phones, However, to accommodate same phones sounding differently due to various co-articulations, some number around 5-6 times this number is chosen (around 80-96 clusters).

   Moreover, a general thumb rule is number of clusters $\approx \frac{\text{number of data points(feature vectors)}}{100}$. We have 8771 feature vectors in total, and therefore we have picked $\mathbf{8771/100 \approx 87}$ **clusters** for the K-Means Clustering (as the size of the codebook). We have found that it performs well. After applying K-Means Clustering and associating each feature vector with the index of the cluster that it belongs to, each digit recording represented initially as a $n \times 38$ sequence of feature vectors is converted to a $n \times 1$ sequence of observations.

2. We store all the length n observation sequences of all recordings of each digit 'two', 'three', 'four', 'five', and 'zero' in separate files in order to train the HMM model for each digit separately. We have chosen the number of states for a certain digit using the following method:

   - Each word has been split up into phones, and almost every phone has been assigned 3 states each, one each for onset, attack, and decay of the phone. Only 'four' is an exception, the last phone /r/ has been given only 2 states based on observation of results. Since it is a shorter phone in the word 'four' and may even be silent in pronunciation sometimes, this choice is somewhat justified intuitively as well.
     - two = /t//u/ = 2 phones = 6 states
     - three = /th//r//i/ = 3 phones = 9 states
     - four = /f//o//r/ = 3 phones = 8 states
     - five = /f//ai//v/ = 3 phones = 9 states
     - zero = /z//i//r//o/ = 4 phones = 12 states

In general, there is no hard and fast rule to select the number of states for the HMMs. If we pick too many states, and hence parameters, we run the risk of training data not being sufficient to learn the parameters well. Nevertheless, we have obtained good results using these numbers of states, so we have stuck with them.

3. Next, we have trained the HMMs with the clustered feature vectors, number of symbols = 87, and number of states as per the digit. These are fed as arguments to the `train_hmm.cc` file, which uses the Baum-Welch Algorithm to train the HMMs.

4. Similarly, the test feature vectors are also mapped to their clusters, and a sequence of observations (cluster indices) is obtained for each text recording. They are all put into a single file and we obtain the log likelihoods (log-alpha values from the Forward method of Testing) of each sequence being generated. We repeat this for each model, and store the sequence likelihoods in a single file.

5. Finally, for each sequence, we give the prediction as the digit corresponding to the model with the highest likelihood for each sequence. Since log is a monotonic function, we can directly compare the log-alpha values to obtain the prediction for each sequence.

The accuracy we have obtained is 98.33%, which is an improvement over DTW-based recognition by 5%.

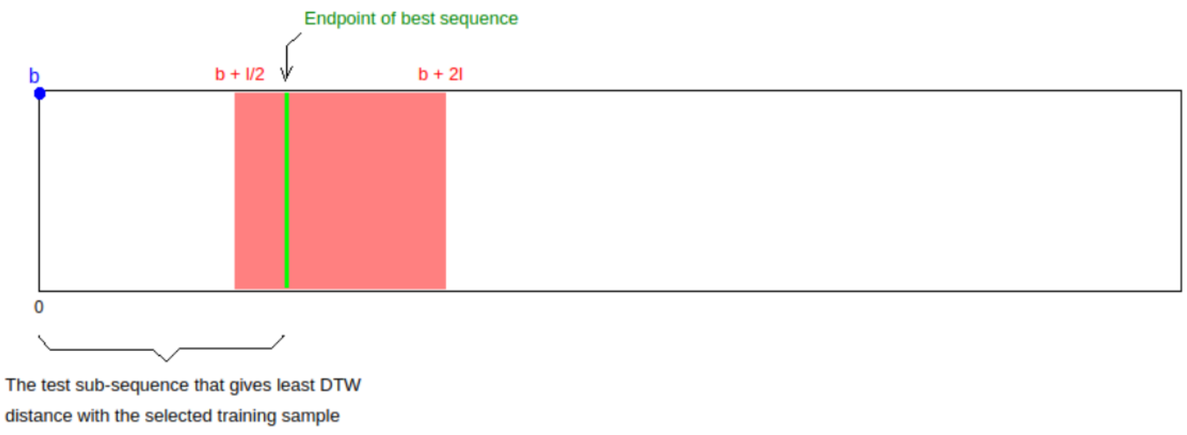
3 Connected Digit Recognition Using 2LDP

Connected Digit Recognition is performed using an extension of the regular dynamic programming used in DTW, to an algorithm called 2-Level Dynamic Programming (2LDP). In this method, we run DTW over ranges of the test sample in which it is most likely that the word lies, predict the best range in which the word may lie, and do this for each word in the vocabulary, picking out the endpoints of the words. We store the values of DTW distance in this process. Then, we assume these to be the new starting points, and repeat the procedure, storing the cumulative DTW distances.

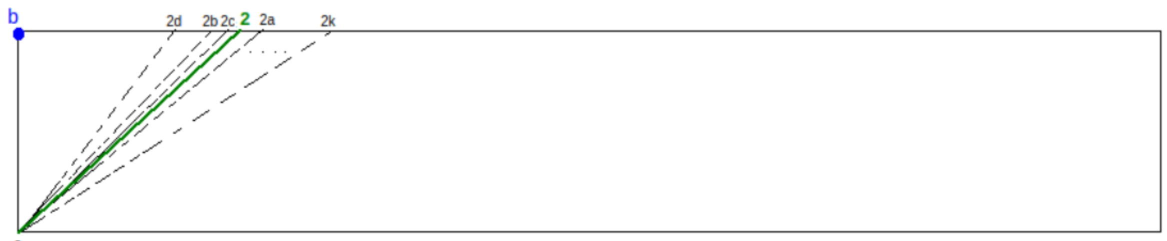
We conclude when we reach the end of the test sample. Finally, the prediction is made by selecting the path that has accumulated the least total DTW distance, and tracing back the path to identify the digits recognised at each stage.

The algorithm is better described below:

1. Let us denote the starting point as b. For each training sample with length l, we set the endpoint e in the range of $[b + \frac{l}{2}, b + 2l]$. We check the DTW distance (as described in Section 1) of the training sample for each e, versus the frames of the test sample in the range $[b : e]$, and select the least DTW distance as the best match of the test sample piece with THAT training sample.

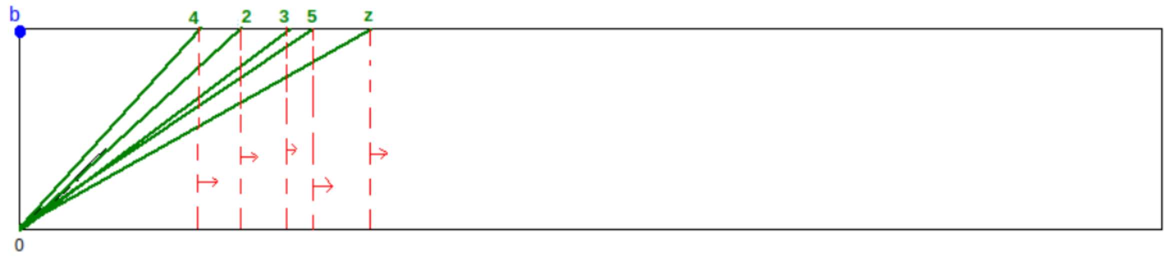

Endpoint of best sequence
b
b + l/2
b + 2l
0
The test sub-sequence that gives least DTW distance with the selected training sample

2. After obtaining the least DTW distance possible for each training sample of a particular digit, we find the minimum distance across all training samples of the digit, to find the least DTW distance for the digit overall (and the corresponding endpoint in the test sample).



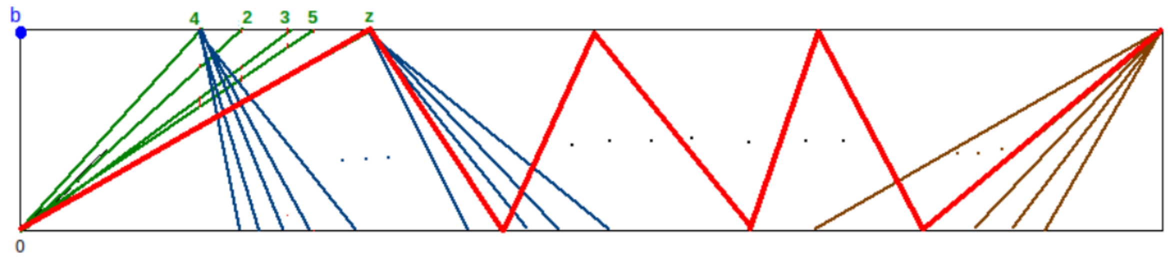
The subsequence with the lowest DTW distance, among all the subsequences that match best with all the training samples of digit '2', is selected as the one best subsequence that represents the digit '2' being uttered first.

3. We obtain this overall minimum DTW distance and the corresponding endpoint for all digits in the vocabulary, after which, we set each of these obtained endpoints as the new starting point and repeat. As we repeat this process, each starting point that we have obtained branches out into a tree - thereby obtaining all possible permutations of digits. We stop a certain tree from branching out when the "average taken across all training samples" of the leftmost endpoint (i.e., average across all training examples of $b + \frac{l}{2}$) crosses the endpoint of the test sample.



The best subsequence representing each digit is found, and their endpoints are set as the starting points for the next iteration of DTW

4. Eventually, all trees stop branching out as their average leftmost endpoints reach the end of the test sample. From all the cumulative DTW distances at the end, we trace back the path that has resulted in the least cumulative DTW distance overall all digit permutations. This is our prediction of the continuous-digit sample.



The sequence with the least cumulative DTW distance is traced back, and the path traced is returned as the connected-digit prediction

Since our implementation of the 2LDP algorithm is taking quite long to run (about 25 min for a sample that's around 80 frames long), it seems too time-consuming to run a test on the entire `dev` data provided and calculating the overall accuracy. However, we have run it for a few examples of connected-digits samples from the `dev` data in order to verify. We have obtained the top 3 predictions for each, corresponding to the 3 least cumulative DTW distances.

Sl.No.	**Original Labels**	**Top 3 Predictions**
1.	(2,3)	**(2,3)**, (2,2), (2,4)
2.	(2,4)	(2,5), (2,3), **(2,4)**
2.	(3,4,4)	(3,3,4), **(3,4,4)**, (3,5,4)

Upon obtaining the above results and consistently obtaining the true labels in the top 3 predictions, we have run the algorithm on the given `blind_test` data, and the corresponding top 3 connected-digit predictions are tabulated below. (The time taken to run all 5 blind tests was around 45-50 minutes. Perhaps, one way to speed this up is to generate a matrix of vector distances of all possible pairs of feature vectors, for each training example - to avoid re-computation whenever necessary in all the DTW iterations.)

Blind Test No.	**Top 3 Predictions**
T1	(4,2), (2,0), (4,4)
T2	(2,2), (0,2), (2,3)
T3	(2,4,2), (2,4,3), (3,4,2)
T4	(3,3,5), (3,3,4), (3,5,5)
T5	(2,5,5), (2,5,4), (2,4,5)

* *