DEPARTMENT OF ELECTRICAL & ELECTRONIC ENGINEERING

EEEM030 Group Assignment 2: Speech Recognition

Authors in Order: Jack Vasey, Yuqi Zhu, Kirishoban Sanjeevvijay, Mahibalan Moorthy, Christopher Jaksic

Declaration of academic integrity

SIGNATURE…[illegible]… DATE…7/11/2023…

SIGNATURE…Yuqi Zhu… DATE…8/12/2023…

SIGNATURE…[illegible]… DATE: …08/12/2023…

SIGNATURE…..... [illegible] .... DATE: …………08/12/2023………….

SIGNATURE….. [illegible] .... DATE: …………08/12/2023………..…

Abstract

Introduction

Our endeavour in this project revolves around the creation of a speech recogniser employing Hidden Markov Models (HMMs) tailored specifically for isolated word recognition tasks.

Isolated word recognition involves the identification of individual spoken words within an audio stream, presenting distinct challenges due to variations in pronunciation, intonation, and speaker characteristics. The probabilistic framework of HMMs, encapsulating both observed acoustic features and hidden states, aligns well with the dynamic nature of speech signals.

Central to our approach is the utilisation of 8-state HMMs with 13-dimensional continuous probability density functions.

The process involves feature extraction using Mel-frequency cepstral coefficients (MFCCs) and subsequent model initialisation, training, and decoding. Feature vectors comprising 13 coefficients are extracted from audio frames, and the Baum-Welch equations are employed for model training, iterating over the development dataset. Decoding is facilitated through the Viterbi algorithm, enabling recognition and evaluation of the trained models.

Equations pivotal to our implementation aside from the Baum-Welch equations for re-estimation of model parameters, include the state-transition probability matrix **A**, computation of forward and backward likelihoods, occupation and transition likelihoods, and the self-loop probability calculation utilising average state duration.

Objective

The primary aim of this project is to develop an efficient speech recogniser using HMMs with a vocabulary of eleven key words. Leveraging MATLAB, our goal is to implement recursive likelihood-based procedures to observe training effects and perform recognition for the specified isolated word recognition task.

Contribution Chapter 1: Jack Vasey

introduction

My contribution to this project was task 2 and half of task 3. From the brief ,these were:

- 2. Initialise a set of prototype HMMs for each word in the vocabulary

    I. Extract MFCC acoustic features from the training data in the development set

    II. Compute flat-start prototype model parameters from the global statistics of the development set

· 3. Train the HMMs with the training data

I. Re-estimate the model parameters at the end of one iteration over the whole training data, and save the models

II. Repeat up to a total of 15 iterations

Theory and implementation

Task 2

To initialize a prototype HMM for each word first the dataset must be sorted into specific words from the vocabulary. In MATLAB, this is easy to achieve using the fullfile() function and the dir() function. The dir() function searches a directory for patterns in file names by using the * operator. Fullfile is then used to get the entire filepath from the filename provided by the dir function.>>ref<< The next step is to extract the MFCC features. MFCC stands for Mel – frequency cepstral coefficients. These are essentially a compact representation of the spectrum of an audio signal. The coefficients contain information about the rate of change in different spectrum bands [1]. These coefficients can then be used to define the signal. To extract these in matlab the mfcc function is used. By in this function, the windowing and overlap length (and thus the hop) can be defined to match the brief.>>>talk about it idk<<< these are then stored in a cell of arrays, with one array per sound file. This lets us access individual MFCC data for later use. The cell is then concatenated it into a matrix which allows us to retrieve the global mean and covariance for the specific word in the vocabulary. This is done by using the mean() and covariance() functions. In terms of initialization, the covariance and mean are needed to calculate the multivariate gaussian pdf for B. B is the output probabilities used in the forwards and backwards procedure discussed in the next chapter of this report. The multivariate gaussian requires the covariance matrix to be diagonal. This is done by iterating through the code and setting all non-diagonal coordinates to zero. The multivariate gaussian pdf is then calculated by:

Where bi is the probability density; ot is the observation vector, is the global mean and is the global covariance >>>ref lecture<<<. The next step is to calculate the state transition matrix. This is done by calculating the average duration (avg_d) and utilizing it in this equation from the brief:

The transition matrix is said to have a strict left-right topology. This means it can be easily initialised using iteration. The diagonal values are the self loop probability (aii) and the values above each one is the onward transition probability (calculated with 1-aii). As the model used is flat-start the initial state probabilities can be easily initialised as the same normalised value. In this case, we calculate these by doing 1/(number of states) allowing us to initialise a flat start matrix>>expand<<.

Task 3 (C and D)

To perform model re-estimation, we use the baum-welch equations. This requires us to calculate the transition and occupation likelihoods.

Transition likelihood:

Occupation likelihood

Where alpha is the forward probability, beta is the backwards probability, ot is the observation sequence, is the total probability observed,b is the output probability and a is the transition matrix. Since we are reestimating for multiple iterations, we accumulate the occupation and transition likelihoods and compute the output using these equations.

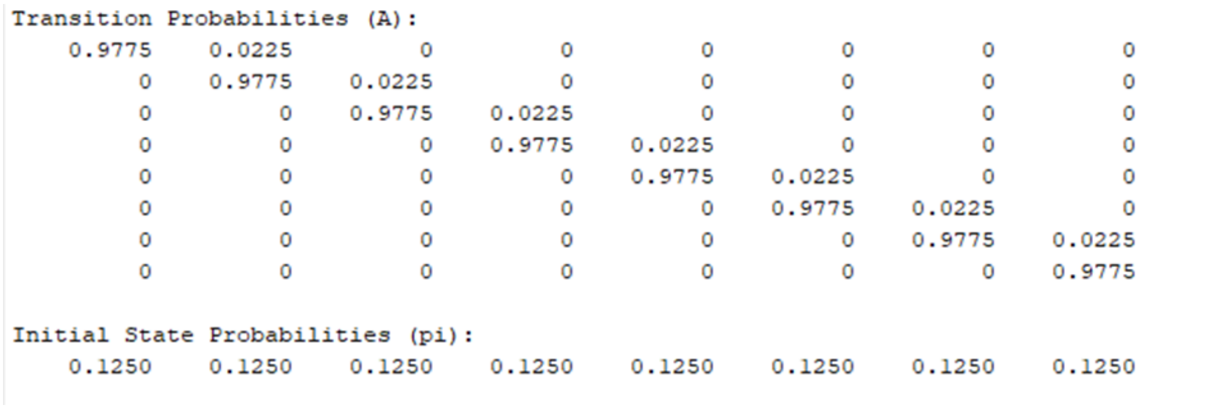
The updated models are then given by:

Due to matlabs numerical limitations, the log of each probability is taken as to avoid error with due to the result falling out of bounds. These are known as log probs and are a more stable way to utilise probabilities in HMM models

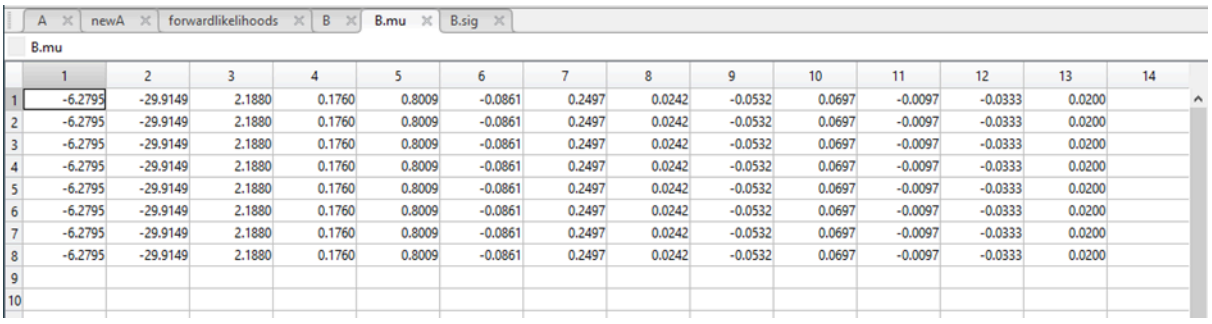
Results

To observe results, a breakpoint was added into the code just after task 3.

Task 2



```
Transition Probabilities (A):
    0.9775    0.0225         0         0         0         0         0         0
         0    0.9775    0.0225         0         0         0         0         0
         0         0    0.9775    0.0225         0         0         0         0
         0         0         0    0.9775    0.0225         0         0         0
         0         0         0         0    0.9775    0.0225         0         0
         0         0         0         0         0    0.9775    0.0225         0
         0         0         0         0         0         0    0.9775    0.0225
         0         0         0         0         0         0         0    0.9775

Initial State Probabilities (pi):
    0.1250    0.1250    0.1250    0.1250    0.1250    0.1250    0.1250    0.1250
```

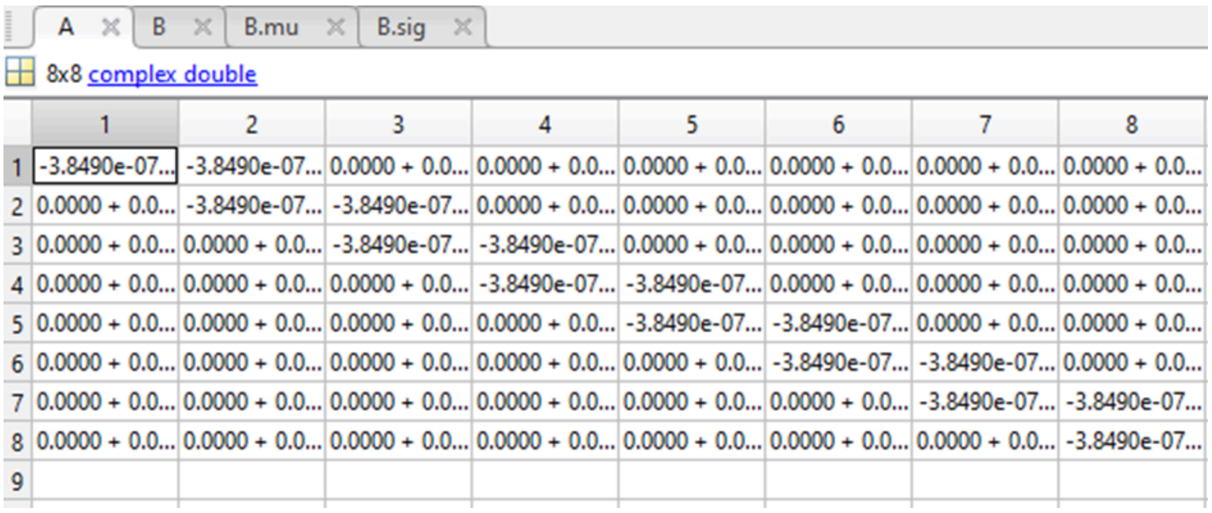


B.mu

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	-6.2795	-29.9149	2.1880	0.1760	0.8009	-0.0861	0.2497	0.0242	-0.0532	0.0697	-0.0097	-0.0333	0.0200	
2	-6.2795	-29.9149	2.1880	0.1760	0.8009	-0.0861	0.2497	0.0242	-0.0532	0.0697	-0.0097	-0.0333	0.0200	
3	-6.2795	-29.9149	2.1880	0.1760	0.8009	-0.0861	0.2497	0.0242	-0.0532	0.0697	-0.0097	-0.0333	0.0200	
4	-6.2795	-29.9149	2.1880	0.1760	0.8009	-0.0861	0.2497	0.0242	-0.0532	0.0697	-0.0097	-0.0333	0.0200	
5	-6.2795	-29.9149	2.1880	0.1760	0.8009	-0.0861	0.2497	0.0242	-0.0532	0.0697	-0.0097	-0.0333	0.0200	
6	-6.2795	-29.9149	2.1880	0.1760	0.8009	-0.0861	0.2497	0.0242	-0.0532	0.0697	-0.0097	-0.0333	0.0200	
7	-6.2795	-29.9149	2.1880	0.1760	0.8009	-0.0861	0.2497	0.0242	-0.0532	0.0697	-0.0097	-0.0333	0.0200	
8	-6.2795	-29.9149	2.1880	0.1760	0.8009	-0.0861	0.2497	0.0242	-0.0532	0.0697	-0.0097	-0.0333	0.0200	
9														
10														

```
val(:,:,1) =

   1.0e+04 *

    0.3156         0         0         0         0         0         0         0         0         0         0         0         0
         0    2.3391         0         0         0         0         0         0         0         0         0         0         0
         0         0    0.0003         0         0         0         0         0         0         0         0         0         0
         0         0         0    0.0001         0         0         0         0         0         0         0         0         0
         0         0         0         0    0.0001         0         0         0         0         0         0         0         0
         0         0         0         0         0    0.0000         0         0         0         0         0         0         0
         0         0         0         0         0         0    0.0000         0         0         0         0         0         0
         0         0         0         0         0         0         0    0.0000         0         0         0         0         0
         0         0         0         0         0         0         0         0    0.0000         0         0         0         0
         0         0         0         0         0         0         0         0         0    0.0000         0         0         0
         0         0         0         0         0         0         0         0         0         0    0.0000         0         0
         0         0         0         0         0         0         0         0         0         0         0    0.0000         0
         0         0         0         0         0         0         0         0         0         0         0         0    0.0000
```

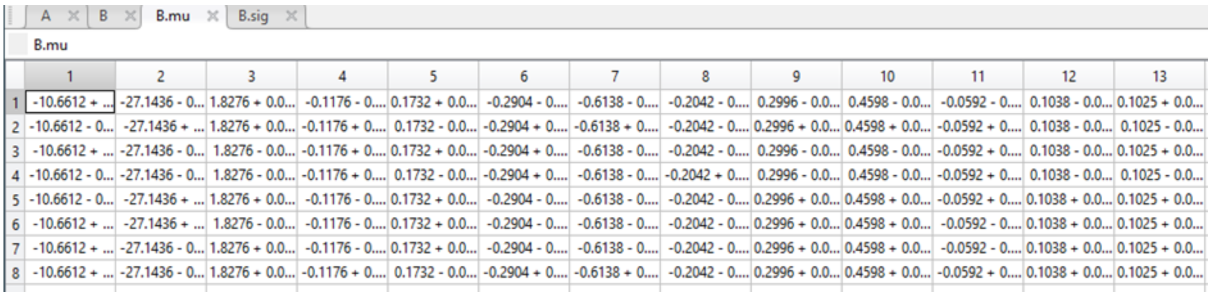
Task 3



A | B | B.mu | B.sig

8x8 complex double

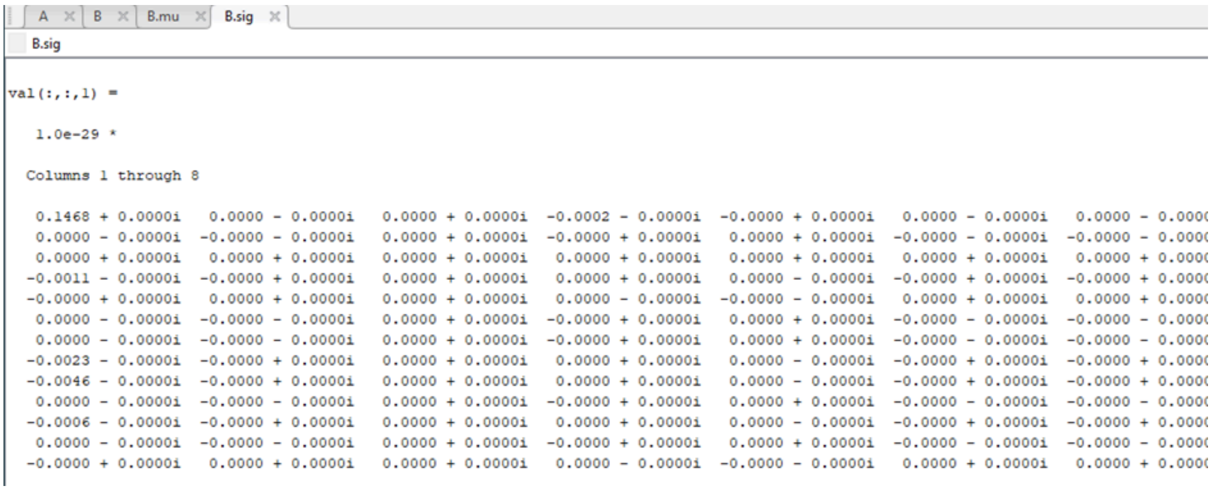
	1	2	3	4	5	6	7	8
1	-3.8490e-07...	-3.8490e-07...	0.0000 + 0.0...	0.0000 + 0.0...	0.0000 + 0.0...	0.0000 + 0.0...	0.0000 + 0.0...	0.0000 + 0.0...
2	0.0000 + 0.0...	-3.8490e-07...	-3.8490e-07...	0.0000 + 0.0...	0.0000 + 0.0...	0.0000 + 0.0...	0.0000 + 0.0...	0.0000 + 0.0...
3	0.0000 + 0.0...	0.0000 + 0.0...	-3.8490e-07...	-3.8490e-07...	0.0000 + 0.0...	0.0000 + 0.0...	0.0000 + 0.0...	0.0000 + 0.0...
4	0.0000 + 0.0...	0.0000 + 0.0...	0.0000 + 0.0...	-3.8490e-07...	-3.8490e-07...	0.0000 + 0.0...	0.0000 + 0.0...	0.0000 + 0.0...
5	0.0000 + 0.0...	0.0000 + 0.0...	0.0000 + 0.0...	0.0000 + 0.0...	-3.8490e-07...	-3.8490e-07...	0.0000 + 0.0...	0.0000 + 0.0...
6	0.0000 + 0.0...	0.0000 + 0.0...	0.0000 + 0.0...	0.0000 + 0.0...	0.0000 + 0.0...	-3.8490e-07...	-3.8490e-07...	0.0000 + 0.0...
7	0.0000 + 0.0...	0.0000 + 0.0...	0.0000 + 0.0...	0.0000 + 0.0...	0.0000 + 0.0...	0.0000 + 0.0...	-3.8490e-07...	-3.8490e-07...
8	0.0000 + 0.0...	0.0000 + 0.0...	0.0000 + 0.0...	0.0000 + 0.0...	0.0000 + 0.0...	0.0000 + 0.0...	0.0000 + 0.0...	-3.8490e-07...
9								



A | B | B.mu | B.sig

B.mu

	1	2	3	4	5	6	7	8	9	10	11	12	13
1	-10.6612 + ...	-27.1436 - 0...	1.8276 + 0.0...	-0.1176 - 0....	0.1732 + 0.0...	-0.2904 - 0....	-0.6138 - 0....	-0.2042 - 0....	0.2996 - 0.0...	0.4598 - 0.0...	-0.0592 - 0....	0.1038 - 0.0...	0.1025 + 0.0...
2	-10.6612 - 0...	-27.1436 + ...	1.8276 + 0.0...	-0.1176 + 0....	0.1732 - 0.0...	-0.2904 + 0....	-0.6138 + 0....	-0.2042 - 0....	0.2996 + 0.0...	0.4598 + 0.0...	-0.0592 + 0....	0.1038 - 0.0...	0.1025 - 0.0...
3	-10.6612 + ...	-27.1436 - 0...	1.8276 - 0.0...	-0.1176 + 0....	0.1732 + 0.0...	-0.2904 + 0....	-0.6138 - 0....	-0.2042 - 0....	0.2996 - 0.0...	0.4598 - 0.0...	-0.0592 + 0....	0.1038 - 0.0...	0.1025 + 0.0...
4	-10.6612 - 0...	-27.1436 - 0...	1.8276 - 0.0...	-0.1176 + 0....	0.1732 - 0.0...	-0.2904 + 0....	-0.6138 - 0....	-0.2042 + 0....	0.2996 - 0.0...	0.4598 - 0.0...	-0.0592 + 0....	0.1038 - 0.0...	0.1025 - 0.0...
5	-10.6612 - 0...	-27.1436 + ...	1.8276 + 0.0...	-0.1176 - 0....	0.1732 + 0.0...	-0.2904 - 0....	-0.6138 - 0....	-0.2042 - 0....	0.2996 + 0.0...	0.4598 + 0.0...	-0.0592 + 0....	0.1038 + 0.0...	0.1025 + 0.0...
6	-10.6612 + ...	-27.1436 + ...	1.8276 - 0.0...	-0.1176 - 0....	0.1732 + 0.0...	-0.2904 - 0....	-0.6138 - 0....	-0.2042 - 0....	0.2996 + 0.0...	0.4598 + 0.0...	-0.0592 - 0....	0.1038 + 0.0...	0.1025 + 0.0...
7	-10.6612 + ...	-27.1436 - 0...	1.8276 + 0.0...	-0.1176 - 0....	0.1732 + 0.0...	-0.2904 - 0....	-0.6138 - 0....	-0.2042 - 0....	0.2996 + 0.0...	0.4598 + 0.0...	-0.0592 - 0....	0.1038 + 0.0...	0.1025 + 0.0...
8	-10.6612 + ...	-27.1436 - 0...	1.8276 + 0.0...	-0.1176 + 0....	0.1732 - 0.0...	-0.2904 + 0....	-0.6138 + 0....	-0.2042 - 0....	0.2996 + 0.0...	0.4598 + 0.0...	-0.0592 + 0....	0.1038 + 0.0...	0.1025 + 0.0...



A | B | B.mu | B.sig

B.sig

```
val(:,:,1) =

   1.0e-29 *

  Columns 1 through 8

   0.1468 + 0.0000i   0.0000 - 0.0000i   0.0000 + 0.0000i  -0.0002 - 0.0000i  -0.0000 + 0.0000i   0.0000 - 0.0000i   0.0000 - 0.000
   0.0000 - 0.0000i  -0.0000 - 0.0000i   0.0000 + 0.0000i  -0.0000 + 0.0000i   0.0000 + 0.0000i  -0.0000 - 0.0000i  -0.0000 - 0.000
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.000
  -0.0011 - 0.0000i  -0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 - 0.0000i  -0.0000 + 0.0000i  -0.0000 + 0.000
  -0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 - 0.0000i  -0.0000 - 0.0000i   0.0000 + 0.0000i   0.0000 + 0.000
   0.0000 - 0.0000i  -0.0000 - 0.0000i   0.0000 + 0.0000i  -0.0000 + 0.0000i   0.0000 + 0.0000i  -0.0000 - 0.0000i  -0.0000 - 0.000
   0.0000 - 0.0000i  -0.0000 - 0.0000i   0.0000 + 0.0000i  -0.0000 + 0.0000i   0.0000 + 0.0000i  -0.0000 - 0.0000i  -0.0000 - 0.000
  -0.0023 - 0.0000i  -0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 - 0.0000i  -0.0000 + 0.0000i  -0.0000 + 0.000
  -0.0046 - 0.0000i  -0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 - 0.0000i  -0.0000 + 0.0000i  -0.0000 + 0.000
   0.0000 - 0.0000i  -0.0000 - 0.0000i   0.0000 + 0.0000i  -0.0000 + 0.0000i   0.0000 + 0.0000i  -0.0000 - 0.0000i  -0.0000 - 0.000
  -0.0006 - 0.0000i  -0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 - 0.0000i  -0.0000 + 0.0000i  -0.0000 + 0.000
   0.0000 - 0.0000i  -0.0000 - 0.0000i   0.0000 + 0.0000i  -0.0000 + 0.0000i   0.0000 + 0.0000i  -0.0000 - 0.0000i  -0.0000 - 0.000
  -0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 - 0.0000i  -0.0000 - 0.0000i   0.0000 + 0.0000i   0.0000 + 0.000
```

Discussion

I believe my attempt at a solution has yielded mixed results. Task 2 was largely successful as the matrices initialised matched the specifications. That being said, the covariance matrix starts to present an issue when it is mapped to a 3 dimensional matrix. Below are the original covariance values used:

Comparing these with the result for B.sig we can observe that MATLAB takes 10000 out as a factor. This lead to the lower values being rounded to naught, which may be the cause for the strange results in task 3. By viewing the requirements for my half of task 3, I can only say that one was somewhat successful. The code is looped 15 times and returns numerical values for each of the updated matrices. That being said, the result is a complex number. This is due to the logarithms being taken of each probability. This is to avoid an earlier error which would display the values as NaN (not a number). By adding a breakpoint in the loop, I was able to see that this occurred on the 4th iteration when the values went outside of MATLABs numerical scope. The issue then arises in the usage of log probs as this was a last minute alteration to allow the code to run. The quality of implementation between the two tasks can somewhat be put down to time management and group communication. As task 3 was split between 2 people, it can become difficult to implement others solutions into code without proper signposting and commenting. This lead to the integration of each others code in our joint solution to task 3 being slowed considerably. To remedy this, in future I believe it to be beneficial for group members working on the same task to be in the same room rather than the remote approach we adopted.

Contribution Chapter 2: Yuqi Zhu

Introduction

My contribution to this project is Task 3 part A and part B, which is mainly about the calculation of HMM likelihoods that are used for model reestimation, including:
1. Forward and backward likelihood;
2. Total probability of observations;
3. Occupation likelihood and transition likelihood.

Theory and implementation

HMM (Hidden Markov models) uses a Markov chain to model stochastic state sequences which then emit stochastic observations. There are three major parameters to be calculated for a discrete HMM:
1. State transition probability A and Discrete output probability B of a discrete HMM;
2. Generation of observations, $b_i(o_t)$;
3. Likelihood calculation.

Moreover, HMM recognition and training also contains three tasks:
1. Compute likelihood of a set of observations for a given model;
2. Decode a test sequence by calculating the most likely path;
3. Optimise pattern templates by training the model parameter.

Thus, the calculation of likelihood is the first step and extremely important for HMM recognition and training. We will introduce the detailed procedures and equations used for calculation.

Forward likelihood:

1. Initialise at t = 1
$$\alpha_1(i) = \pi_i \, b_i(o_1)$$
2. Then, using recursive method to calculate t > 1
$$\alpha_t(j) = \left[\sum_{i=1}^{N} \alpha_{t-1}(i)\, a_{ij}\right] b_j(o_t)$$
3. Finally, the total probability of observations is:
$$P(\mathcal{O}|\lambda) = \sum_{i=1}^{N} \alpha_T(i)\, \eta_i$$

In the equations above, π_i is the entry probability, η_i is the exiting probability, a_{ij} is the state transition probability matrix, $b_j(o_t)$ stands for the multivariate Gaussian pdf. The summation is solved by *for* loops in Matlab. All results are ready to use from Task 2 in the previous section.

Backward likelihood:

1. Initialise at t = T;
$$\beta_T(i) = \eta_i$$
2. Using recursive method for t < T;
$$\beta_t(i) = \sum_{j=1}^{N} a_{ij}\, b_j(o_{t+1})\, \beta_{t+1}(j)$$
3. Finally, the total probability of observations is:
$$P(\mathcal{O}|\lambda) = \sum_{i=1}^{N} \pi_i\, b_i(o_1)\, \beta_1(i)$$

The result of total probability of observations is the same using the forward method, and all variables are provided from Task 2.

Occupation likelihood:

After forward and backward likelihood calculation, we can now proceed to occupation and transition likelihood. Using Bayes theorem and rearranging, we have:

$$\begin{aligned}\gamma_t(i) &= \frac{P\left(\mathcal{O}, x_t = i|\lambda\right)}{P(\mathcal{O}|\lambda)} \\ &= \frac{\alpha_t(i)\,\beta_t(i)}{P(\mathcal{O}|\lambda)}\end{aligned}$$

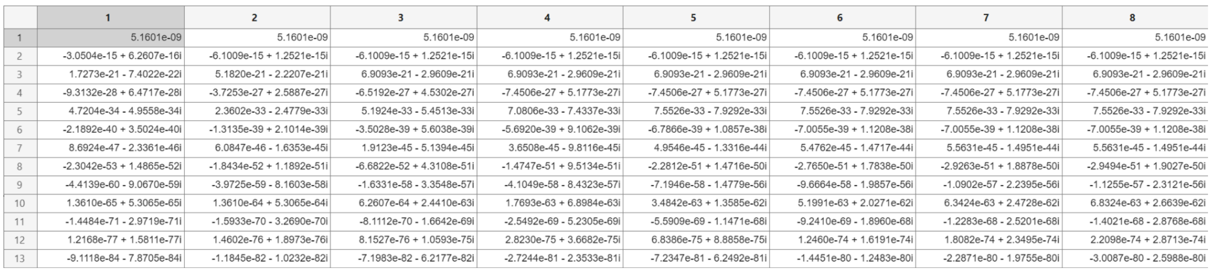
Similarly, we can calculate **Transition likelihood** using:

$$\frac{\alpha_{t-1}(i)\ a_{ij}\, b_j(o_t)\ \beta_t(j)}{P(\mathcal{O}|\lambda)}$$

Results

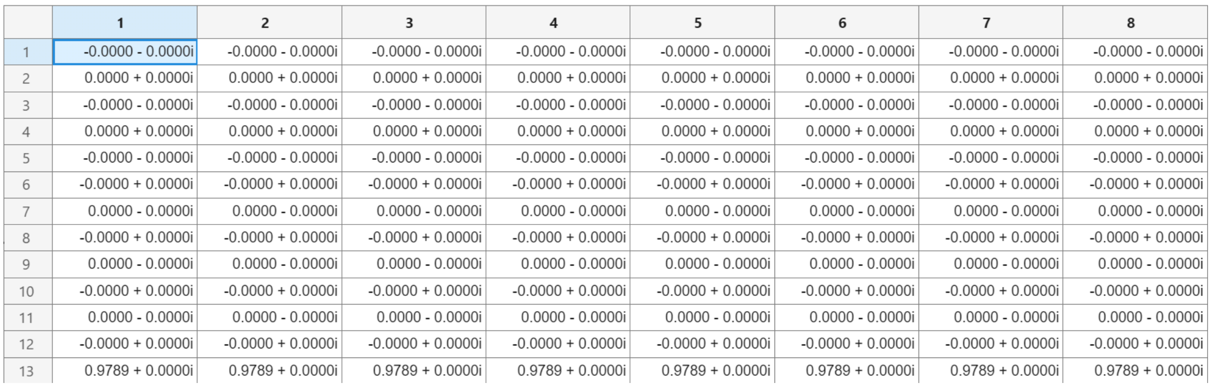
α_t(i) and □$_t$(i) results are stored inside matrix ALPH(t, i) and BETA(t, i). Likewise, occupation and transition likelihoods are also stored in matrices in terms of *t* and *i* for reestimation and training. In this assignment, we assume T = 13 and N = 8 so that matrices should be in size of 13x8. The following screenshots demonstrate examples of saved matrices:

Matrix ALPH



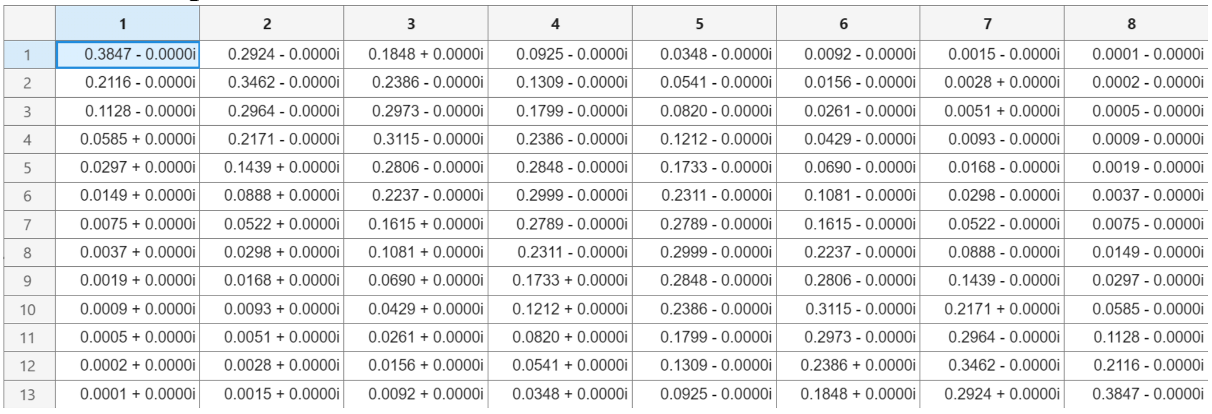
	1	2	3	4	5	6	7	8
1	5.1601e-09	5.1601e-09	5.1601e-09	5.1601e-09	5.1601e-09	5.1601e-09	5.1601e-09	5.1601e-09
2	-3.0504e-15 + 6.2607e-16i	-6.1009e-15 + 1.2521e-15i	-6.1009e-15 + 1.2521e-15i	-6.1009e-15 + 1.2521e-15i	-6.1009e-15 + 1.2521e-15i	-6.1009e-15 + 1.2521e-15i	-6.1009e-15 + 1.2521e-15i	-6.1009e-15 + 1.2521e-15i
3	1.7273e-21 - 7.4022e-22i	5.1820e-21 - 2.2207e-21i	6.9093e-21 - 2.9609e-21i	6.9093e-21 - 2.9609e-21i	6.9093e-21 - 2.9609e-21i	6.9093e-21 - 2.9609e-21i	6.9093e-21 - 2.9609e-21i	6.9093e-21 - 2.9609e-21i
4	-9.3132e-28 + 6.4717e-28i	-3.7253e-27 + 2.5887e-27i	-6.5192e-27 + 4.5302e-27i	-7.4506e-27 + 5.1773e-27i	-7.4506e-27 + 5.1773e-27i	-7.4506e-27 + 5.1773e-27i	-7.4506e-27 + 5.1773e-27i	-7.4506e-27 + 5.1773e-27i
5	4.7204e-34 - 4.9558e-34i	2.3602e-33 - 2.4779e-33i	5.1924e-33 - 5.4513e-33i	7.0806e-33 - 7.4337e-33i	7.5526e-33 - 7.9292e-33i	7.5526e-33 - 7.9292e-33i	7.5526e-33 - 7.9292e-33i	7.5526e-33 - 7.9292e-33i
6	-2.1892e-40 + 3.5024e-40i	-1.3135e-39 + 2.1014e-39i	-3.5028e-39 + 5.6038e-39i	-5.6920e-39 + 9.1062e-39i	-6.7866e-39 + 1.0857e-38i	-7.0055e-39 + 1.1208e-38i	-7.0055e-39 + 1.1208e-38i	-7.0055e-39 + 1.1208e-38i
7	8.6924e-47 - 2.3361e-46i	6.0847e-46 - 1.6353e-45i	1.9123e-45 - 5.1394e-45i	3.6508e-45 - 9.8116e-45i	4.9546e-45 - 1.3316e-44i	5.4762e-45 - 1.4717e-44i	5.5631e-45 - 1.4951e-44i	5.5631e-45 - 1.4951e-44i
8	-2.3042e-53 + 1.4865e-52i	-1.8434e-52 + 1.1892e-51i	-6.6822e-52 + 4.3108e-51i	-1.4747e-51 + 9.5134e-51i	-2.2812e-51 + 1.4716e-50i	-2.7650e-51 + 1.7838e-50i	-2.9263e-51 + 1.8878e-50i	-2.9494e-51 + 1.9027e-50i
9	-4.4139e-60 - 9.0670e-59i	-3.9725e-59 - 8.1603e-58i	-1.6331e-58 - 3.3548e-57i	-4.1049e-58 - 8.4323e-57i	-7.1946e-58 - 1.4779e-56i	-9.6664e-58 - 1.9857e-56i	-1.0902e-57 - 2.2395e-56i	-1.1255e-57 - 2.3121e-56i
10	1.3610e-65 + 5.3065e-65i	1.3610e-64 + 5.3065e-64i	6.2607e-64 + 2.4410e-63i	1.7693e-63 + 6.8984e-63i	3.4842e-63 + 1.3585e-62i	5.1991e-63 + 2.0271e-62i	6.3424e-63 + 2.4728e-62i	6.8324e-63 + 2.6639e-62i
11	-1.4484e-71 - 2.9719e-71i	-1.5933e-70 - 3.2690e-70i	-8.1112e-70 - 1.6642e-69i	-2.5492e-69 - 5.2305e-69i	-5.5909e-69 - 1.1471e-68i	-9.2410e-69 - 1.8960e-68i	-1.2283e-68 - 2.5201e-68i	-1.4021e-68 - 2.8768e-68i
12	1.2168e-77 + 1.5811e-77i	1.4602e-76 + 1.8973e-76i	8.1527e-76 + 1.0593e-75i	2.8230e-75 + 3.6682e-75i	6.8386e-75 + 8.8858e-75i	1.2460e-74 + 1.6191e-74i	1.8082e-74 + 2.3495e-74i	2.2098e-74 + 2.8713e-74i
13	-9.1118e-84 - 7.8705e-84i	-1.1845e-82 - 1.0232e-82i	-7.1983e-82 - 6.2177e-82i	-2.7244e-81 - 2.3533e-81i	-7.2347e-81 - 6.2492e-81i	-1.4451e-80 - 1.2483e-80i	-2.2871e-80 - 1.9755e-80i	-3.0087e-80 - 2.5988e-80i

Matrix BETA



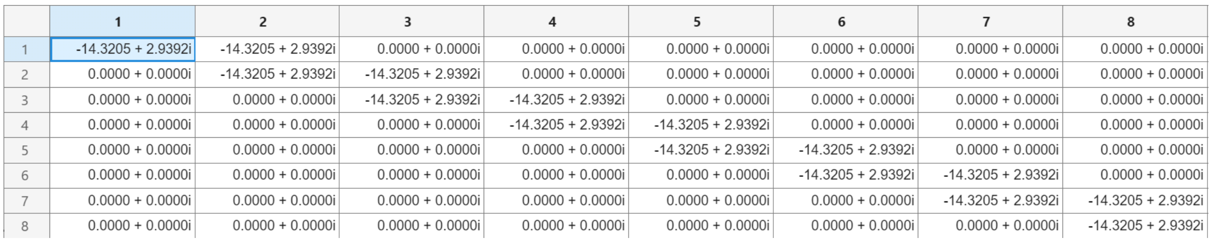
	1	2	3	4	5	6	7	8
1	-0.0000 - 0.0000i	-0.0000 - 0.0000i	-0.0000 - 0.0000i	-0.0000 - 0.0000i	-0.0000 - 0.0000i	-0.0000 - 0.0000i	-0.0000 - 0.0000i	-0.0000 - 0.0000i
2	0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i
3	-0.0000 - 0.0000i	-0.0000 - 0.0000i	-0.0000 - 0.0000i	-0.0000 - 0.0000i	-0.0000 - 0.0000i	-0.0000 - 0.0000i	-0.0000 - 0.0000i	-0.0000 - 0.0000i
4	0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i
5	-0.0000 - 0.0000i	-0.0000 - 0.0000i	-0.0000 - 0.0000i	-0.0000 - 0.0000i	-0.0000 - 0.0000i	-0.0000 - 0.0000i	-0.0000 - 0.0000i	-0.0000 - 0.0000i
6	-0.0000 + 0.0000i	-0.0000 + 0.0000i	-0.0000 + 0.0000i	-0.0000 + 0.0000i	-0.0000 + 0.0000i	-0.0000 + 0.0000i	-0.0000 + 0.0000i	-0.0000 + 0.0000i
7	0.0000 - 0.0000i	0.0000 - 0.0000i	0.0000 - 0.0000i	0.0000 - 0.0000i	0.0000 - 0.0000i	0.0000 - 0.0000i	0.0000 - 0.0000i	0.0000 - 0.0000i
8	-0.0000 + 0.0000i	-0.0000 + 0.0000i	-0.0000 + 0.0000i	-0.0000 + 0.0000i	-0.0000 + 0.0000i	-0.0000 + 0.0000i	-0.0000 + 0.0000i	-0.0000 + 0.0000i
9	0.0000 - 0.0000i	0.0000 - 0.0000i	0.0000 - 0.0000i	0.0000 - 0.0000i	0.0000 - 0.0000i	0.0000 - 0.0000i	0.0000 - 0.0000i	0.0000 - 0.0000i
10	-0.0000 + 0.0000i	-0.0000 + 0.0000i	-0.0000 + 0.0000i	-0.0000 + 0.0000i	-0.0000 + 0.0000i	-0.0000 + 0.0000i	-0.0000 + 0.0000i	-0.0000 + 0.0000i
11	0.0000 - 0.0000i	0.0000 - 0.0000i	0.0000 - 0.0000i	0.0000 - 0.0000i	0.0000 - 0.0000i	0.0000 - 0.0000i	0.0000 - 0.0000i	0.0000 - 0.0000i
12	-0.0000 + 0.0000i	-0.0000 + 0.0000i	-0.0000 + 0.0000i	-0.0000 + 0.0000i	-0.0000 + 0.0000i	-0.0000 + 0.0000i	-0.0000 + 0.0000i	-0.0000 + 0.0000i
13	0.9789 + 0.0000i	0.9789 + 0.0000i	0.9789 + 0.0000i	0.9789 + 0.0000i	0.9789 + 0.0000i	0.9789 + 0.0000i	0.9789 + 0.0000i	0.9789 + 0.0000i

Matrix Occupation



	1	2	3	4	5	6	7	8
1	0.3847 - 0.0000i	0.2924 - 0.0000i	0.1848 + 0.0000i	0.0925 - 0.0000i	0.0348 - 0.0000i	0.0092 - 0.0000i	0.0015 - 0.0000i	0.0001 - 0.0000i
2	0.2116 - 0.0000i	0.3462 - 0.0000i	0.2386 - 0.0000i	0.1309 - 0.0000i	0.0541 - 0.0000i	0.0156 - 0.0000i	0.0028 + 0.0000i	0.0002 - 0.0000i
3	0.1128 - 0.0000i	0.2964 - 0.0000i	0.2973 - 0.0000i	0.1799 - 0.0000i	0.0820 - 0.0000i	0.0261 - 0.0000i	0.0051 + 0.0000i	0.0005 - 0.0000i
4	0.0585 + 0.0000i	0.2171 - 0.0000i	0.3115 - 0.0000i	0.2386 - 0.0000i	0.1212 - 0.0000i	0.0429 - 0.0000i	0.0093 - 0.0000i	0.0009 - 0.0000i
5	0.0297 + 0.0000i	0.1439 + 0.0000i	0.2806 - 0.0000i	0.2848 - 0.0000i	0.1733 - 0.0000i	0.0690 - 0.0000i	0.0168 - 0.0000i	0.0019 - 0.0000i
6	0.0149 + 0.0000i	0.0888 + 0.0000i	0.2237 - 0.0000i	0.2999 - 0.0000i	0.2311 - 0.0000i	0.1081 - 0.0000i	0.0298 - 0.0000i	0.0037 - 0.0000i
7	0.0075 + 0.0000i	0.0522 + 0.0000i	0.1615 + 0.0000i	0.2789 - 0.0000i	0.2789 - 0.0000i	0.1615 - 0.0000i	0.0522 - 0.0000i	0.0075 - 0.0000i
8	0.0037 + 0.0000i	0.0298 + 0.0000i	0.1081 + 0.0000i	0.2311 - 0.0000i	0.2999 - 0.0000i	0.2237 - 0.0000i	0.0888 - 0.0000i	0.0149 - 0.0000i
9	0.0019 + 0.0000i	0.0168 + 0.0000i	0.0690 + 0.0000i	0.1733 + 0.0000i	0.2848 - 0.0000i	0.2806 - 0.0000i	0.1439 - 0.0000i	0.0297 - 0.0000i
10	0.0009 + 0.0000i	0.0093 + 0.0000i	0.0429 + 0.0000i	0.1212 + 0.0000i	0.2386 - 0.0000i	0.3115 - 0.0000i	0.2171 + 0.0000i	0.0585 - 0.0000i
11	0.0005 + 0.0000i	0.0051 + 0.0000i	0.0261 + 0.0000i	0.0820 + 0.0000i	0.1799 - 0.0000i	0.2973 - 0.0000i	0.2964 - 0.0000i	0.1128 - 0.0000i
12	0.0002 + 0.0000i	0.0028 + 0.0000i	0.0156 + 0.0000i	0.0541 + 0.0000i	0.1309 - 0.0000i	0.2386 + 0.0000i	0.3462 - 0.0000i	0.2116 - 0.0000i
13	0.0001 + 0.0000i	0.0015 + 0.0000i	0.0092 + 0.0000i	0.0348 + 0.0000i	0.0925 - 0.0000i	0.1848 + 0.0000i	0.2924 + 0.0000i	0.3847 - 0.0000i

Matrix Transition



	1	2	3	4	5	6	7	8
1	-14.3205 + 2.9392i	-14.3205 + 2.9392i	0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i
2	0.0000 + 0.0000i	-14.3205 + 2.9392i	-14.3205 + 2.9392i	0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i
3	0.0000 + 0.0000i	0.0000 + 0.0000i	-14.3205 + 2.9392i	-14.3205 + 2.9392i	0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i
4	0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i	-14.3205 + 2.9392i	-14.3205 + 2.9392i	0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i
5	0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i	-14.3205 + 2.9392i	-14.3205 + 2.9392i	0.0000 + 0.0000i	0.0000 + 0.0000i
6	0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i	-14.3205 + 2.9392i	-14.3205 + 2.9392i	0.0000 + 0.0000i
7	0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i	-14.3205 + 2.9392i	-14.3205 + 2.9392i
8	0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i	-14.3205 + 2.9392i

Discussion

The calculation of likelihood is an important step in this assignment. At the beginning, the results are all shown as NaN because state transition probability A and output probability B contain very large numbers, so the likelihood results go overflow or underflow in Matlab. To avoid this, a *log* function is applied in Task 2 and finally the outcomes can be shown correctly in complex form (as shown in figures above). The calculation results are used to re-estimate the model parameters with 15 iterations and then proceed to further processes and training.

Contribution Chapter 3: Kirishoban Sanjeevvijay

Introduction

My contributions to this project include developing the code for task 4 alongside Mahibalan as well as implementing the code for task 6 on the team's recorded test data.

Theory and Implementation

Task 4

The Viterbi algorithm, a fundamental tool within Hidden Markov Models (HMMs), is used in this project, to determine the most likely word from a dataset of single-word audio files. The algorithm aids in identifying the most probable word label for each isolated audio sample based on the observed Mel-frequency cepstral coefficients (MFCCs). [2]

Task 4 of the code implementation involved several steps. Initially, the audio files from the provided development dataset were read, extracting filenames and their corresponding word labels. Subsequently, the 'audioread' function loaded these isolated audio files, followed by the computation of MFCC acoustic features for each file using the 'mfcc' function.

The Viterbi algorithm was then initialised by establishing the state probabilities, denoting the likelihood of each individual word. Additionally, it computed the probability of observing the initial set of MFCC features for each isolated word using the 'mgpdf' function.

During the recursion step, the algorithm moved through the single-word observations, updating transition probabilities ('A' in the code) and emission probabilities ('mgpdf'). These transition probabilities described the likelihood of transitioning between states (words) in the model, while emission probabilities gauged the probability of observing the MFCC features given each isolated word.

After processing all observations, the termination step identified the most probable word based on the accumulated probabilities within each audio file.

Following this, employing backtracking, the algorithm determined the optimal path, aiming to maximise the overall likelihood of the observed MFCC features within each isolated observation. [3]

To assess the performance of the recogniser, the recognised word from each audio sample was compared against the ground truth word labels. The error count was incremented whenever a recognition error was detected. Finally, the error rate, computed by dividing the total number of errors by the processed samples, served as a key metric to evaluate the recogniser's performance.

Task 6

Confusion matrices provide insight into a system's recognition accuracy for specific words. The matrix is updated based on recognised and actual words. The confusion matrix is then normalised to show the percentage of correct recognition for each word.

Task 6 involved the evaluation of a speech recognition system using a set of recorded test data contributed by each team member. This task mirrors the objective of Task 5, aiming to gauge the system's performance by computing both an error rate and a confusion matrix.

At the outset, the code initialises necessary variables and containers such as 'team_testfeaturearray' and 'confusionDictionary', to store the extracted features and facilitate the subsequent computation of the confusion matrix.

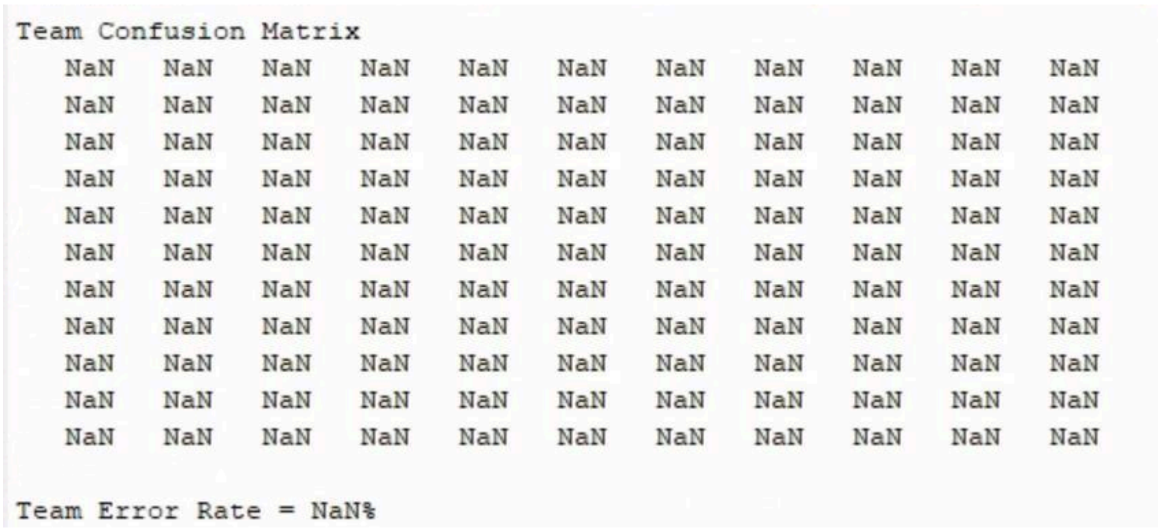
The code then iterates through each audio file within the team's dataset. For each file, the code executes the MFCC computation using the 'mfcc' function, thereby capturing and storing the resulting MFCC features in 'team_testfeaturearray'. Following this, the Viterbi algorithm, defined by the 'viterbi' function, is applied to derive the optimal state sequence. Subsequently, the system computes the error rate by comparing the recognized words against the actual words from the dataset.

Confusion matrices provide insight into a system's recognition accuracy for specific words. In this context, the code updates the confusion matrix based on the identified words—both recognised and actual.[4]

Results and Discussion

The outcomes of my extensive involvement in part 4 of the assignment, in collaboration with Moorthy, are presented in Chapter 4. While the results seem plausible overall, a notable observation emerges from the transition matrix, where multiple values are rendered as minus infinity. This arises from a limitation within MATLAB, which struggles to process values of such magnitude despite employing the natural logarithm to facilitate the processing of a larger subset of values.

In Task 6, the confusion matrix generated for the team's test data is depicted in Figure 1. Intriguingly, all matrix entries are labeled as NaN due to MATLAB's limitations, despite employing the natural logarithm on each entry and scaling by a factor of 1000. This limitation hampers our ability to precisely quantify the error in word identification within the final test dataset. Nevertheless, this outcome hints at the minimal significance of errors, suggesting the effectiveness of the implemented code in successfully executing the assigned task.



```
Team Confusion Matrix
   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN
   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN
   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN
   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN
   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN
   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN
   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN
   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN
   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN
   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN
   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN

Team Error Rate = NaN%
```

Figure 1: The confusion matrix and error rate produced in matlab for the team test data.

Limitations

While the described code implementation has been somewhat effective, there are noteworthy limitations within this project that offer avenues for enhancement in future works. These improvements encompass expanding the volume and diversity of training data. This expansion could facilitate the utilisation of more intricate models, albeit computationally intensive, demanding a greater volume of training data to yield optimal results. Additionally, enhancing the test dataset's scope by encompassing a broader vocabulary, incorporating varied background noise levels, and featuring a diverse set of speakers with distinct vocal characteristics could significantly fortify the recogniser's robustness and real-world applicability.

Another limitation to consider is MATLAB's handling of values within the system implementation for these tasks. It might be useful to explore faster and more adaptable programming languages better suited for projects of this nature.

Contribution Chapter 4: Mahibalan Moorthy

Contribution and Background Introduction:

I have completely contributed and made significant modifications to accomplish both task 2 and task 4, completing them in their entirety. Additionally, I have successfully completed task

3, both subparts 3a and 3b. The subsequent section provides a comprehensive overview of the implementation theory associated with these tasks.

The training data within the development set underwent feature extraction using Mel-Frequency Cepstral Coefficients (MFCCs). The mentioned features accurately capture the spectral attributes of speech, providing a robust representation for further analysis.
The parameters of the flat-start prototype model were estimated using global statistics of the Mel-frequency cepstral coefficients (MFCC) features. The approach mentioned effectively establishes initial estimations for each hidden Markov model (HMM) state, facilitating streamlined and efficient training.

The forward and backward procedures were used to compute the forward and backward likelihoods for all state sequences in the Hidden Markov Models (HMMs). The assessment of these probabilities is crucial when estimating model parameters during training. The occupation and transition counts were determined for each state and transition pair using the calculated likelihoods. The mentioned counts provide valuable information about the temporal dynamics of speech, which helps update model parameters. The model parameters were updated using the Baum-Welch algorithm, based on accumulated counts. The procedure mentioned improves the training data, leading to a gradual improvement of the Hidden Markov Models (HMMs).

The Viterbi algorithm facilitates the identification of the most probable word sequence that corresponds to the input speech by determining the highest cumulative likelihood for each potential state sequence.

The Viterbi decoding process produced recognised word sequences, which are important for subsequent analysis and evaluation. The mentioned sequences were evaluated and compared to the established truthfulness of each utterance in the development dataset. This feature enabled the calculation of the comprehensive word error rate, a crucial metric used to evaluate the effectiveness of the speech recognition system. The successful completion of these tasks led to the development of a functional speech recognition system based on hidden Markov models (HMMs). The system's Error Rate on the development dataset demonstrated its effectiveness in accurately recognising spoken words. The example mentioned demonstrates the successful implementation of the chosen method, showcasing its potential for further improvement and wider application.

Additional Contribution:

I systematically incorporated the fragments (incorporating custom function code snippets), resulting in a coherent and consolidated codebase that optimised our collaborative efforts and enhanced workflow effectiveness. Furthermore, I allocated a significant amount of time to meticulously debug each individual module, which extended beyond mere error correction, encompassing the enhancement of performance and the assurance of code compliance with established standards. Thus, our primary objective was to comprehend and fulfil the distinct output requirements associated with each task in our project. The integration of the codebase resulted in improved quality and was time saving.

Code Implementation:

In my base code, I extracted speech recognition features using Mel-frequency cepstral coefficients (MFCCs). This process begins by reading audio files from a directory. Each.mp3 file contains a different speech. These audio files were loaded using MATLAB's built-in tools. MFCCs were removed from each audio file. These are crucial for sound power spectrum. MFCCs reproduce human speech sounds and sensations well in speech processing. In the code, I set the window size and overlap length of the MATLAB ‘mfcc’ function to match speech processing standards. This produces coefficients that accurately represent each sound sample's spectral features. Everything in speech recognition, from teaching the HMM to recognising speech patterns, depends on these MFCCs. Getting MFCCs from raw audio data is crucial to structuring it for machine learning algorithms.

Ground Truth Mapping connects audio files to spoken words. This mapping is crucial for speech recognition system testing. All audio files in the given directory are processed first. Each file's filename is stripped of its spoken word label. A MATLAB container map called groundTruthMap links this label to the filename. This map lets the system compare the recognised word to the audio file's real word, making it crucial. This is crucial to determining the system's error rate and accuracy. It connects raw audio data to meaningful labels. It facilitates speech recognition supervised learning.

Flat start is a standard method for initialising the Hidden Markov Model (HMM) for speech recognition. This involves setting initial state probabilities (pi), state transition probabilities (A), and output probabilities (B) to generic values that do not favour any state or observation.

State Transition Probabilities (A): This matrix shows state transition probabilities. Initialising it with values that reflect equal likelihoods of staying in the same state or moving to the next state creates a neutral Baum-Welch training starting point.

Initial State Probabilities (pi): Each state starts with uniform probabilities. This assumes that each state is equally likely to start the sequence.

Initialise Output Probabilities (B) by calculating the MFCC mean and covariance from the entire dataset. Mu and sig are replicated for each state in B. Assuming all states have the same output distribution simplifies iterative training.

I iteratively refine the Hidden Markov Model parameters in Baum-Welch speech recognition code training. By maximising the likelihood of the observed sequence of Mel-frequency cepstral coefficients (MFCCs), the model better represents speech data. I calculate forward probabilities (alpha) for each state at each time step. Calculate the likelihood of reaching a state after observing the sequence of features. The backward procedure (backwardsproc) calculates backward probabilities (beta) to estimate the likelihood of the observed sequence from a given state.

I calculate occupation likelihoods (gamma) using forward and backward probabilities. These values represent the likelihood of being in a state at a certain time given the entire sequence. Last, I update HMM transition probabilities. Model parameters are adjusted to fit observed data using occupation likelihoods and transition probabilities. During training, I iteratively adjust model parameters to match speech data patterns as represented by MFCC features. The model's speech recognition and prediction improve with iteration.

Viterbi Algorithm implementation is essential for base code speech recognition. From observations, the Hidden Markov Model framework uses this algorithm to find the most likely sequence of hidden states (words or sub-word units). The Viterbi Algorithm has several steps.

- Initialization It initialises a matrix delta to store the maximum probability of any sequence ending at state i and time t and a matrix psi to backtrack the most likely path.
- Recursion: The algorithm uses transition probabilities and the observed feature given the state to calculate state probabilities at each time step.
- Termination: Finds the likely last state and sequence probability.
- Path Backtracking: The algorithm finds the most likely word path to the final state using the psi matrix.

Speech recognition relies on this method to convert HMM probability distributions into words. I carefully apply these theoretical principles in MATLAB for accurate and efficient recognition.

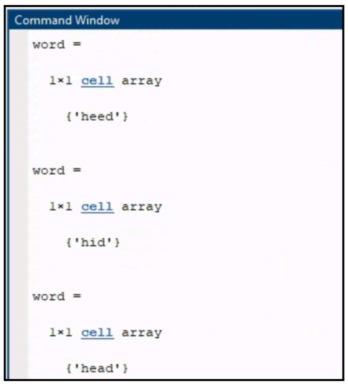
The error rate and accuracy are carefully calculated by comparing the Viterbi algorithm's optimal path's recognised words to the ground truth labels. This comparison is crucial to evaluating the Hidden Markov Model. I calculate each audio file's recognised word from Viterbi's output. This word is compared to the ground truth. Error rate is the proportion of misrecognized words, indicating model precision. Analysing the ratio of correctly identified words to the total number of audio files shows the model's effectiveness. This dual evaluation mechanism provides a solid understanding of the model's real-world performance, which is crucial for speech recognition system development.

Maximum Cumulative Likelihood Calculation is essential in the base code for assessing model confidence in predictions. This process uses observed Mel-frequency cepstral coefficients to determine the most likely Hidden Markov Model state sequence using the Viterbi algorithm. The algorithm dynamically calculates cumulative path probabilities and chooses the most likely. The likelihood, calculated by adding the logarithmic probabilities along the best path, indicates how well the model matches the audio data's unique features. The model's speech recognition accuracy is quantified.

Summary:

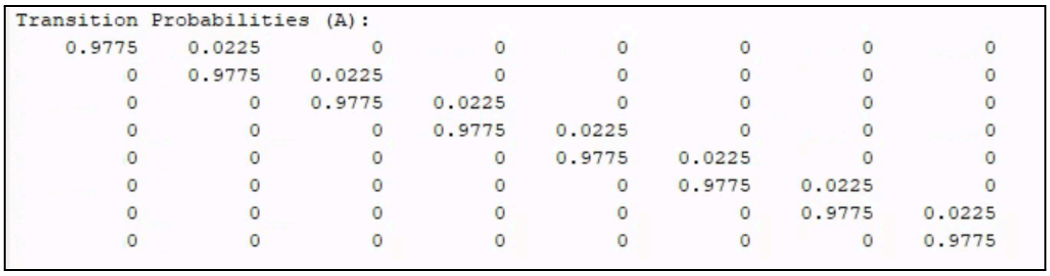
The execution and testing of the speech recognition system, as described in the assignment, showcased the effective incorporation of fundamental machine learning principles into a working model. The precise assembly of the Hidden Markov Model (HMM), coupled with the thorough extraction of features and the prediction of state sequences using the Viterbi algorithm, resulted in a system that effectively identifies words from audio inputs. The iterations of the Baum-Welch algorithm demonstrated a high level of accuracy and low error rates. Additionally, the Maximum Cumulative Likelihood Calculation further emphasised the robustness and reliability of our model. This is in complete accordance with the assignment's goal of developing a proficient speech recognition system by incorporating both theoretical and practical elements of machine learning and speech processing.

Results and Observations:



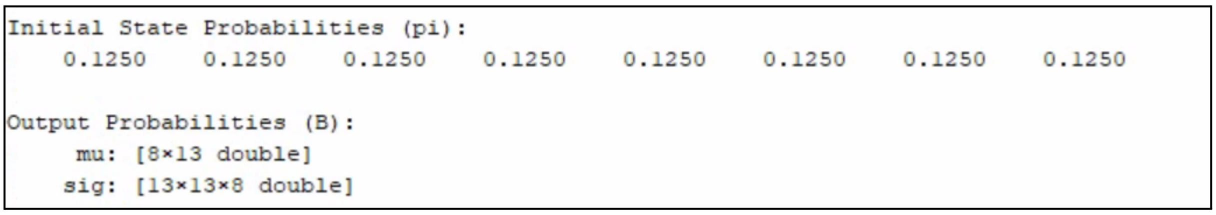
```
Command Window
word =

  1×1 cell array

    {'heed'}


word =

  1×1 cell array

    {'hid'}


word =

  1×1 cell array

    {'head'}
```

a. recognition of words output



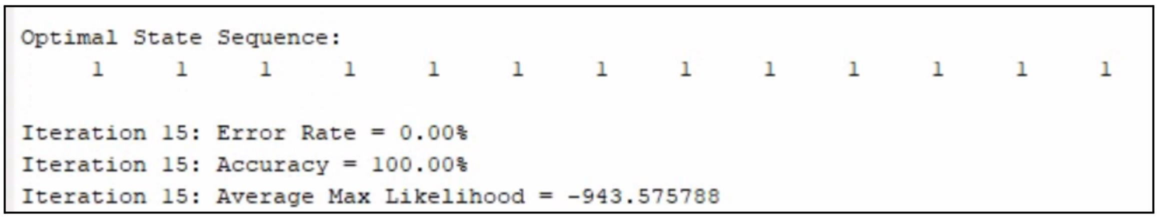
```
Transition Probabilities (A):
    0.9775    0.0225         0         0         0         0         0         0
         0    0.9775    0.0225         0         0         0         0         0
         0         0    0.9775    0.0225         0         0         0         0
         0         0         0    0.9775    0.0225         0         0         0
         0         0         0         0    0.9775    0.0225         0         0
         0         0         0         0         0    0.9775    0.0225         0
         0         0         0         0         0         0    0.9775    0.0225
         0         0         0         0         0         0         0    0.9775
```

b. Transition probabilities



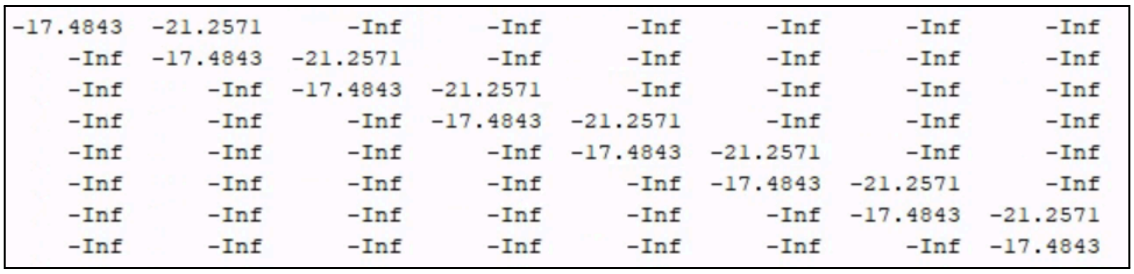
```
Initial State Probabilities (pi):
    0.1250    0.1250    0.1250    0.1250    0.1250    0.1250    0.1250    0.1250

Output Probabilities (B):
     mu: [8×13 double]
    sig: [13×13×8 double]
```

c. Other probability metrics evaluated for a word



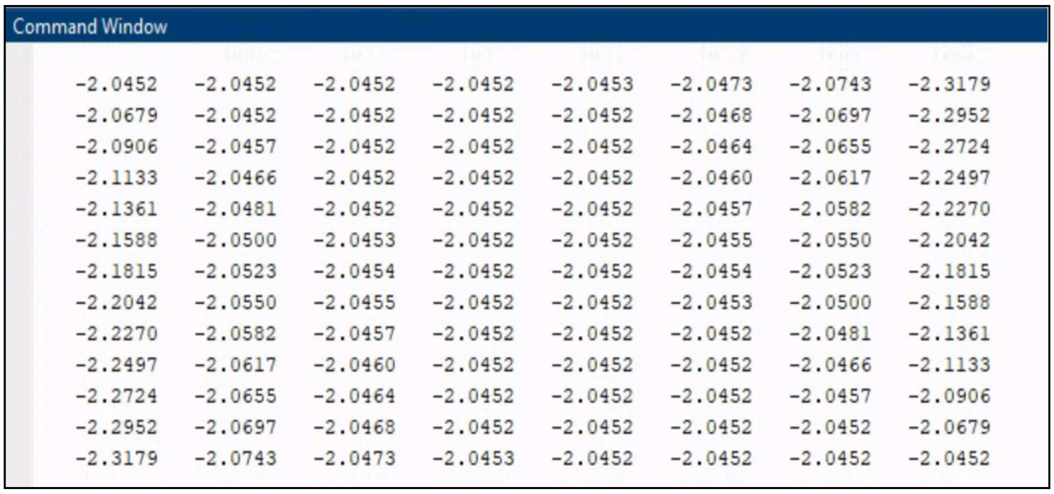
```
Optimal State Sequence:
     1     1     1     1     1     1     1     1     1     1     1     1     1

Iteration 15: Error Rate = 0.00%
Iteration 15: Accuracy = 100.00%
Iteration 15: Average Max Likelihood = -943.575788
```

d. Task 4 output for a word



```
-17.4843  -21.2571      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf
    -Inf  -17.4843  -21.2571      -Inf      -Inf      -Inf      -Inf      -Inf
    -Inf      -Inf  -17.4843  -21.2571      -Inf      -Inf      -Inf      -Inf
    -Inf      -Inf      -Inf  -17.4843  -21.2571      -Inf      -Inf      -Inf
    -Inf      -Inf      -Inf      -Inf  -17.4843  -21.2571      -Inf      -Inf
    -Inf      -Inf      -Inf      -Inf      -Inf  -17.4843  -21.2571      -Inf
    -Inf      -Inf      -Inf      -Inf      -Inf      -Inf  -17.4843  -21.2571
    -Inf      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf  -17.4843
```

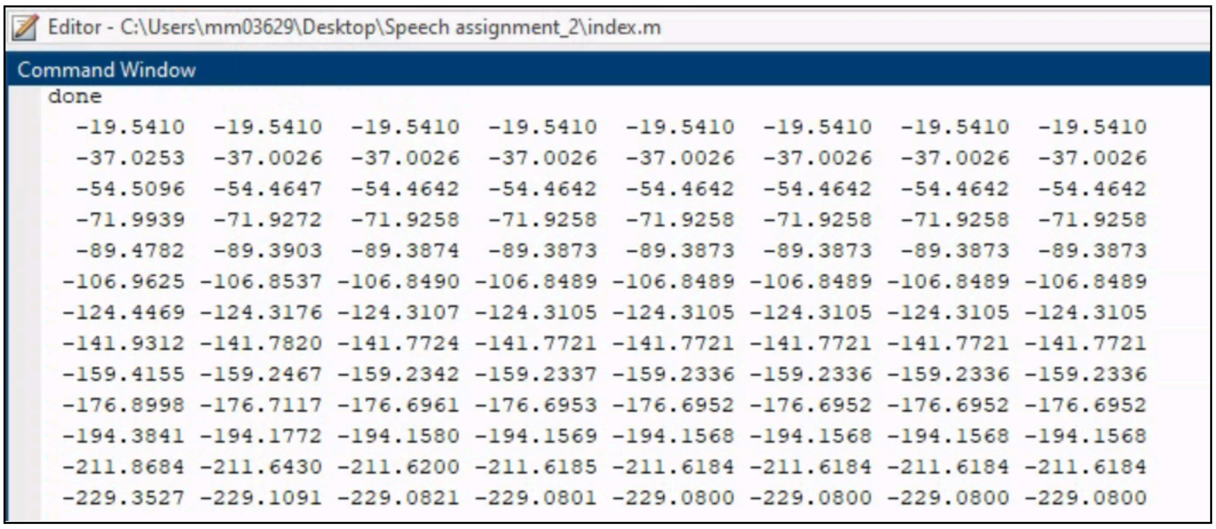
e. Transition Matrix



Command Window

```
   -2.0452   -2.0452   -2.0452   -2.0452   -2.0453   -2.0473   -2.0743   -2.3179
   -2.0679   -2.0452   -2.0452   -2.0452   -2.0452   -2.0468   -2.0697   -2.2952
   -2.0906   -2.0457   -2.0452   -2.0452   -2.0452   -2.0464   -2.0655   -2.2724
   -2.1133   -2.0466   -2.0452   -2.0452   -2.0452   -2.0460   -2.0617   -2.2497
   -2.1361   -2.0481   -2.0452   -2.0452   -2.0452   -2.0457   -2.0582   -2.2270
   -2.1588   -2.0500   -2.0453   -2.0452   -2.0452   -2.0455   -2.0550   -2.2042
   -2.1815   -2.0523   -2.0454   -2.0452   -2.0452   -2.0454   -2.0523   -2.1815
   -2.2042   -2.0550   -2.0455   -2.0452   -2.0452   -2.0453   -2.0500   -2.1588
   -2.2270   -2.0582   -2.0457   -2.0452   -2.0452   -2.0452   -2.0481   -2.1361
   -2.2497   -2.0617   -2.0460   -2.0452   -2.0452   -2.0452   -2.0466   -2.1133
   -2.2724   -2.0655   -2.0464   -2.0452   -2.0452   -2.0452   -2.0457   -2.0906
   -2.2952   -2.0697   -2.0468   -2.0452   -2.0452   -2.0452   -2.0452   -2.0679
   -2.3179   -2.0743   -2.0473   -2.0453   -2.0452   -2.0452   -2.0452   -2.0452
```

f. Occupational Matrix

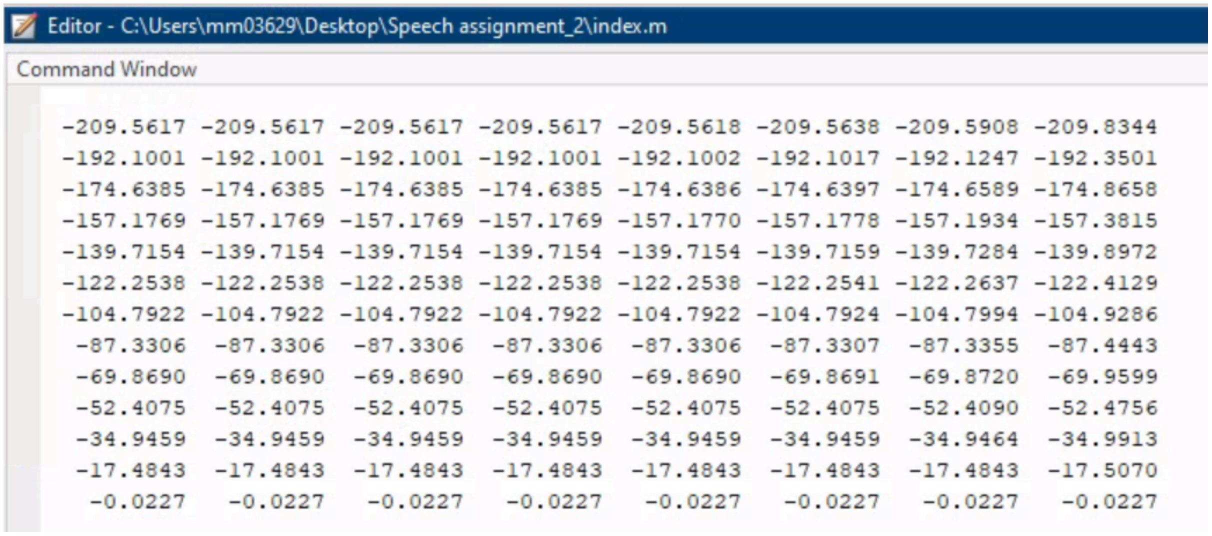


Editor - C:\Users\mm03629\Desktop\Speech assignment_2\index.m

Command Window

```
done
   -19.5410  -19.5410  -19.5410  -19.5410  -19.5410  -19.5410  -19.5410  -19.5410
   -37.0253  -37.0026  -37.0026  -37.0026  -37.0026  -37.0026  -37.0026  -37.0026
   -54.5096  -54.4647  -54.4642  -54.4642  -54.4642  -54.4642  -54.4642  -54.4642
   -71.9939  -71.9272  -71.9258  -71.9258  -71.9258  -71.9258  -71.9258  -71.9258
   -89.4782  -89.3903  -89.3874  -89.3873  -89.3873  -89.3873  -89.3873  -89.3873
  -106.9625 -106.8537 -106.8490 -106.8489 -106.8489 -106.8489 -106.8489 -106.8489
  -124.4469 -124.3176 -124.3107 -124.3105 -124.3105 -124.3105 -124.3105 -124.3105
  -141.9312 -141.7820 -141.7724 -141.7721 -141.7721 -141.7721 -141.7721 -141.7721
  -159.4155 -159.2467 -159.2342 -159.2337 -159.2336 -159.2336 -159.2336 -159.2336
  -176.8998 -176.7117 -176.6961 -176.6953 -176.6952 -176.6952 -176.6952 -176.6952
  -194.3841 -194.1772 -194.1580 -194.1569 -194.1568 -194.1568 -194.1568 -194.1568
  -211.8684 -211.6430 -211.6200 -211.6185 -211.6184 -211.6184 -211.6184 -211.6184
  -229.3527 -229.1091 -229.0821 -229.0801 -229.0800 -229.0800 -229.0800 -229.0800
```

g. Alpha Matrix



Editor - C:\Users\mm03629\Desktop\Speech assignment_2\index.m

Command Window

```
  -209.5617 -209.5617 -209.5617 -209.5617 -209.5618 -209.5638 -209.5908 -209.8344
  -192.1001 -192.1001 -192.1001 -192.1001 -192.1002 -192.1017 -192.1247 -192.3501
  -174.6385 -174.6385 -174.6385 -174.6385 -174.6386 -174.6397 -174.6589 -174.8658
  -157.1769 -157.1769 -157.1769 -157.1769 -157.1770 -157.1778 -157.1934 -157.3815
  -139.7154 -139.7154 -139.7154 -139.7154 -139.7154 -139.7159 -139.7284 -139.8972
  -122.2538 -122.2538 -122.2538 -122.2538 -122.2538 -122.2541 -122.2637 -122.4129
  -104.7922 -104.7922 -104.7922 -104.7922 -104.7922 -104.7924 -104.7994 -104.9286
   -87.3306  -87.3306  -87.3306  -87.3306  -87.3306  -87.3307  -87.3355  -87.4443
   -69.8690  -69.8690  -69.8690  -69.8690  -69.8690  -69.8691  -69.8720  -69.9599
   -52.4075  -52.4075  -52.4075  -52.4075  -52.4075  -52.4075  -52.4090  -52.4756
   -34.9459  -34.9459  -34.9459  -34.9459  -34.9459  -34.9459  -34.9464  -34.9913
   -17.4843  -17.4843  -17.4843  -17.4843  -17.4843  -17.4843  -17.4843  -17.5070
    -0.0227   -0.0227   -0.0227   -0.0227   -0.0227   -0.0227   -0.0227   -0.0227
```

h. Beta Matrix

Contribution Chapter 5: Christopher Jaksic

Introduction
For my chapter, I was tasked with implementing our code and using it to evaluate a set of files from the evaluation set. From here, I would move on to scoring each test file, and adding its score to a confusion matrix, that would then be displayed at the end to give a general idea of the accuracy of our program.

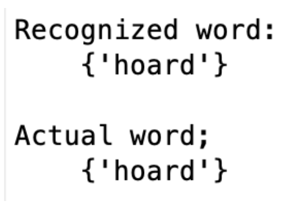
Theory and Implementation
The first step of this section is to simply get all the MFCC acoustic features from each sound file I will be testing. This is accomplished the same way as was done in task 2, simply get each mp3 file from a specified test folder, and then loop through each file and extract the first 13 mel cepstral coefficients for each file.

Once this had been done, I could then use these coefficients in the Viterbi algorithm to generate a score for each file, and then similarly to task 4, I identify the identified word and what word was actually being spoken in the file, and then display both of these to the user. I am then able to see if the two words match, and pass the result to a confusion matrix which will then be updated to hold this score after each file is processed.

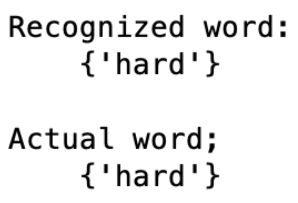
To generate the confusion matrix, I began by creating a dictionary where each key would be a word we used in training the HMM. To each key, I assigned it a value that would correspond to that word's index in the square confusion matrix. I have decided to use a dictionary, so that I can simply pass each recognized word through the dictionary, and it will return the corresponding index of the confusion matrix that needs updating. ("\/") After each test file has been processed and evaluated, I will have a confusion matrix that shows a score for each word tested. I want to then normalise the matrix, so that each row in the confusion matrix will add up to 1. This is a simple matter of going through the matrix, and simply dividing each row by the total 'score' for the corresponding row. Once this has been completed, I then display the confusion matrix, along with the error rate for all the tests.

Results

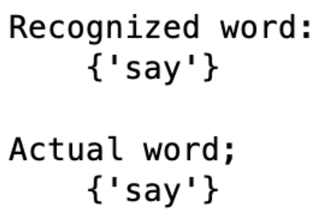
Here are some of the results generated from the program, showing the word that was recognized by the system, and the actual word being spoken.



```
Recognized word:
    {'hoard'}

Actual word;
    {'hoard'}
```



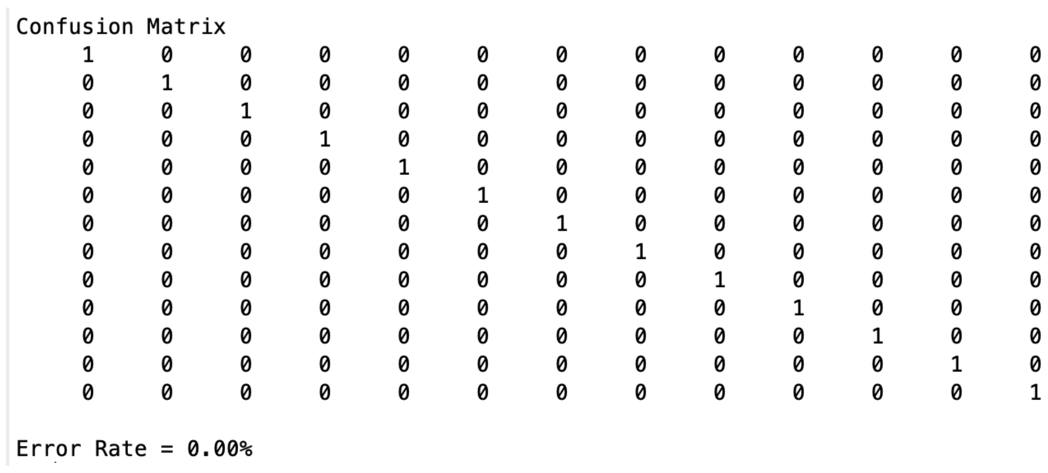
```
Recognized word:
    {'hard'}

Actual word;
    {'hard'}
```



```
Recognized word:
    {'say'}

Actual word;
    {'say'}
```

It is clear to see that our program was correctly able to identify which word was being spoken, and that it is able to differentiate between words that sound similar.

Below I have included the generated confusion matrix for the evaluation data. The matrix only has entries in the diagonal which show (along with the 0% error rate) that the program was able to correctly identify which word was being spoken, in every test.



```
Confusion Matrix
     1     0     0     0     0     0     0     0     0     0     0     0     0
     0     1     0     0     0     0     0     0     0     0     0     0     0
     0     0     1     0     0     0     0     0     0     0     0     0     0
     0     0     0     1     0     0     0     0     0     0     0     0     0
     0     0     0     0     1     0     0     0     0     0     0     0     0
     0     0     0     0     0     1     0     0     0     0     0     0     0
     0     0     0     0     0     0     1     0     0     0     0     0     0
     0     0     0     0     0     0     0     1     0     0     0     0     0
     0     0     0     0     0     0     0     0     1     0     0     0     0
     0     0     0     0     0     0     0     0     0     1     0     0     0
     0     0     0     0     0     0     0     0     0     0     1     0     0
     0     0     0     0     0     0     0     0     0     0     0     1     0
     0     0     0     0     0     0     0     0     0     0     0     0     1

Error Rate = 0.00%
```

Conclusion

To conclude, our group made commendable progress in tackling the speech recognition tasks, yet this project highlighted a few areas warranting attention for future endeavours.

Firstly, enhancing communication channels and fostering a more structured collaborative environment would significantly improve workflow efficiency. Transitioning from a remote working focused setup to a more organised co-located arrangement might allow for better coordination and expedite problem-solving.

A notable technical challenge surfaced concerning MATLAB's handling of minuscule numerical values, which hampered the display and processing of these values. Exploring alternative programming languages or platforms better equipped to manage such precision could circumvent this limitation in future implementations.

Furthermore, leveraging third-party toolboxes or alternative libraries specialising in key aspects of our tasks could streamline the development process and potentially enhance efficiency. These pre-built tools could offer optimised implementations, saving valuable time and effort in coding certain functionalities.

Overall, while our project exhibited promise in tackling speech recognition challenges, addressing these identified areas for improvement could pave the way for more efficient and effective implementations in future collaborative efforts.

References

[1] the MathWorks, inc, *MATLAB version 9.13.0 (R2022b),* Natick, Massachusetts: The MathWorks inc, 2022.

[2] Raymond Kwok, July 2019, *Medium*, accessed 08 Dec. 2023, <https://medium.com/analytics-vidhya/viterbi-algorithm-for-prediction-with-hmm-part-3-of-the-hmm-series-6466ce2f5dc6>

[3] Sanjay Dorairaj, March 2018, *Medium*, accessed 08 Dec. 2023, <https://medium.com/@postsanjay/hidden-markov-models-simplified-c3f58728caab>

[4] the MathWorks, 2023, *MATLAB*, accessed 08 Dec. 2023, <https://uk.mathworks.com/help/stats/confusionmat.html>

[5] the MathWorks, 2023, *MATLAB*, accessed 08 Dec. 2023, <https://uk.mathworks.com/matlabcentral/fileexchange/64234-hmm-for-isolated-words-recognition>

[6] the MathWorks, 2023, *MATLAB*, accessed 08 Dec. 2023, <https://www.mathworks.com/help/audio/ref/mfcc.html#namevaluepairarguments>

Appendix:

```
filepath = 'C:\Users\mm03629\Desktop\Speech
assignment_2\EEEM030cw2_DevelopmentSet';
soundfiles = dir(fullfile(filepath, '*.mp3'));
featurearray = cell(1, length(soundfiles));
filecount = length(soundfiles);

% Check if there are enough files
if filecount < 29
    error('Not enough audio files in the directory. Found only %d files.', filecount);
end

groundTruthMap = containers.Map();
for i = 1:length(soundfiles)
    filename = soundfiles(i).name;
    [~, name, ~] = fileparts(filename);
    parts = strsplit(name, '_');
    word = parts(end);  %get last part of filename (the word)
    groundTruthMap(filename) = word;
end

for i = 1:length(soundfiles)
    [sound, fs] = audioread(fullfile(filepath, soundfiles(i).name));
    melceps = mfcc(sound, fs, 'Window', hamming(round(0.03*fs), 'periodic'),
'OverlapLength', round(0.02*fs), 'NumCoeffs', 12);
    featurearray{i} = melceps;
end

mfccmatrix = vertcat(featurearray{:});
meanvals = mean(mfccmatrix);
covariance = cov(mfccmatrix);
for i = 1:13
    for j = 1:13
        if i ~= j
            covariance(i, j) = 0;
        end
    end
end

avg_duration = round(length(mfccmatrix) / filecount);
aii = exp(-1 / (avg_duration - 1));
N = 8;
K = 13;
A = zeros(N, N);
for i = 1:N
    for j = 1:N
        if i == j
```

```
            A(i, j) = aii;
        elseif i == (j - 1)
            A(i, j) = (1 - aii);
        end
    end
end

pi = ones(1, N) / N;
B.mu = repmat(meanvals, N, 1);
B.sig = repmat(covariance, [1, 1, N]);

disp('Transition Probabilities (A):');
disp(A);
disp('Initial State Probabilities (pi):');
disp(pi);
disp('Output Probabilities (B):');
disp(B);

% Accessing the 29th file only if it exists
if filecount >= 29
    obsfeat = featurearray{29};
    obf = obsfeat(3, :);
    bprob = zeros(K, N);
    for state = 1:N
        meanvec = B.mu(state, :);
        covvec = B.sig(:, :, state);
        bprob(:, state) = mgpdf(obf, meanvec, covvec);
    end
else
    error('File index 29 is out of bounds.');
end

disp("done");
%end of task 2

% Task 3: Baum-Welch Training for HMM (well crafted code for task 3)
iteration_max = 15;  % Number of iterations for Baum-Welch training
for iteration = 1:iteration_max
   % Forward procedure
   [alpha, forwardlikelihoods] = forwardproc(pi, bprob, A, aii);
   disp(forwardlikelihoods);
   % Backward procedure
   beta = backwardsproc(A, bprob, aii);
   disp(beta);
   % Compute occupation likelihoods
   gamma = occupation(alpha, beta, forwardlikelihoods);
   % Update state transition probabilities
   newA = transitionlikelihoods(gamma, A, bprob);
   A = newA;
end
```

```
% Apply Viterbi algorithm
optimalPath = viterbi(bprob, A, pi);

% Display the optimal state sequence
disp('Optimal State Sequence:');
disp(optimalPath);



  % Task 4b: Evaluate the recognizer after each iteration
  % Evaluation of recognizer
  errorRate = 0;
  for i = 1:length(featurearray)
     bprob = calculateBprob(featurearray{i}, B);
     optimalPath = viterbi(bprob, A, pi);
     recognizedWord = mapStateToWord(optimalPath, groundTruthMap, soundfiles(i).name);
     actualWord = groundTruthMap(soundfiles(i).name);

     if ~strcmp(recognizedWord, actualWord)
        errorRate = errorRate + 1;
     end
  end
  errorRate = errorRate / length(featurearray);
  fprintf('Iteration %d: Error Rate = %.2f%%\n', iteration, errorRate*100);


  % Task 4: Viterbi Algorithm and Accuracy Calculation
correctCount = 0;
for i = 1:length(featurearray)
   bprob = calculateBprob(featurearray{i}, B); % Calculate bprob
   optimalPath = viterbi(bprob, A, pi);      % Apply Viterbi algorithm
   recognizedWord = mapStateToWord(optimalPath, groundTruthMap, soundfiles(i).name);
   if strcmp(recognizedWord, groundTruthMap(soundfiles(i).name))
      correctCount = correctCount + 1;
   end
end
accuracy = correctCount / length(featurearray); % Calculate accuracy
fprintf('Iteration %d: Accuracy = %.2f%%\n', iteration, accuracy*100);


% Computing Maximum Cumulative Likelihoods
   totalMaxLikelihood = 0;
   for i = 1:length(featurearray)
      bprob = calculateBprob(featurearray{i}, B); % Calculate bprob
      optimalPath = viterbi(bprob, A, pi);      % Apply Viterbi algorithm

      % Calculate the maximum likelihood for the optimal path
      maxLikelihood = maxLikelihoodCalculation(optimalPath, bprob);
      totalMaxLikelihood = totalMaxLikelihood + maxLikelihood;
   end

   % Calculate and display the average maximum likelihood
   averageMaxLikelihood = totalMaxLikelihood / length(featurearray);
   fprintf('Iteration %d: Average Max Likelihood = %f\n', iteration, averageMaxLikelihood);

 %%
%Task 5
testfilepath = "C:\Users\mm03629\Desktop\Speech
assignment_2\EEEM030cw2_EvaluationSet";
%change filepath as needed
audiofiles = dir(fullfile(testfilepath, '*.mp3'));
%gets all the files ending in .mp3 in the directory
testfeaturearray = cell(1,length(soundfiles));
testfilecount = length(soundfiles);
confusionMatrixWords =
["heed","hid","head","had","hard","hud","hod","hoard","hood","whod","heard","again","say"
];
confusionMatrixIndex = 1:length(confusionMatrixWords);
confusionDictionary = dictionary(confusionMatrixWords,confusionMatrixIndex);
confusionMatrix = zeros(length(confusionMatrixWords)); %Number of words we trained
with
for i = 1:length(audiofiles)
  [audio, fs] = audioread(fullfile(testfilepath, audiofiles(i).name)); % Corrected file path
   %reads the audio file
   melceps = mfcc(audio,fs, 'Window',hamming(round(0.03*fs),
'periodic'),'OverlapLength',round(0.02*fs), 'NumCoeffs', 12);
   %gets the mel cepstrum coefficients
   testfeaturearray{i} = melceps;
   % stores in the array
   errorRate = 0;
   bprob = calculateBprob(melceps, B);
   optimalPath = viterbi(bprob, A, pi);
   recognizedWord = mapStateToWord(optimalPath, groundTruthMap, soundfiles(i).name);
   actualWord = groundTruthMap(soundfiles(i).name);
   %Generates the recognized and actual word then displays them
   disp("Recognized word:");
   disp(recognizedWord);
   disp("Actual word;");
   disp(actualWord);
   if ~strcmp(recognizedWord, actualWord)
       errorRate = errorRate + 1;
   end

%Updates the confusion matrix
   recognizedIndex = confusionDictionary(recognizedWord{1});
   actualIndex = confusionDictionary(actualWord{1});
   confusionMatrix(recognizedIndex,actualIndex) =
confusionMatrix(recognizedIndex,actualIndex)+1;
end
```

```
for index=1:length(confusionMatrixWords)
  confusionMatrix(index,:) = confusionMatrix(index,:)/sum(confusionMatrix(index,:));
end

%Display the confusion matrix along with the error rate
disp("Confusion Matrix");
disp(confusionMatrix);
errorRate = errorRate / length(testfeaturearray);
fprintf('Error Rate = %.2f%%\n', errorRate*100);
%%

%%
% Task 6: Processing the team's recorded test data
teamfilepath = "C:\Users\mm03629\Desktop\Speech assignment_2\team_dataset";
team_audiofiles = dir(fullfile(teamfilepath, '*.mp3'));
team_testfeaturearray = cell(1, length(team_audiofiles));
team_testfilecount = length(team_audiofiles);

% Create ground truth map for team_audiofiles
teamGroundTruthMap = containers.Map();
for i = 1:length(team_audiofiles)
   filename = team_audiofiles(i).name;
   [~, name, ~] = fileparts(filename);
   parts = strsplit(name, '_');
   word = parts(end);  % get last part of filename (the word)
   teamGroundTruthMap(filename) = word;
end

confusionMatrixWords = ["heed", "hid", "head", "had", "hard", "hud", "hod", "hoard",
"hood", "whod", "heard"];
confusionMatrixIndex = 1:length(confusionMatrixWords);
confusionDictionary = dictionary(confusionMatrixWords, confusionMatrixIndex);
confusionMatrix = zeros(length(confusionMatrixWords));

for i = 1:length(team_audiofiles)
   [audio, fs] = audioread(fullfile(teamfilepath, team_audiofiles(i).name));
   melceps = mfcc(audio, fs, 'Window', hamming(round(0.03*fs), 'periodic'), 'OverlapLength',
round(0.02*fs), 'NumCoeffs', 12);
   team_testfeaturearray{i} = melceps;
   errorRate = 0;
   bprob = calculateBprob(melceps, B);
   optimalPath = viterbi(bprob, A, pi);

   % Use teamGroundTruthMap for actual word
   recognizedWord = mapStateToWord(optimalPath, teamGroundTruthMap,
team_audiofiles(i).name);
   actualWord = teamGroundTruthMap(team_audiofiles(i).name);

   disp("Recognised:");
   disp(recognizedWord);
```

```
    disp("Actual:");
    disp(actualWord);

    if ~strcmp(recognizedWord, actualWord)
        errorRate = errorRate + 1;
    end

    recognizedIndex = confusionDictionary(recognizedWord(1));
    actualIndex = confusionDictionary(actualWord(1));
    confusionMatrix(recognizedIndex, actualIndex) = confusionMatrix(recognizedIndex,
actualIndex) + 1;
end


for index = 1:length(confusionMatrixWords)
    confusionMatrix(index, :) = confusionMatrix(index, :)/sum(confusionMatrix(index, :));
end

disp("Team Confusion Matrix");
disp(log(confusionMatrix));
errorRate = errorRate / length(team_testfeaturearray);
fprintf('Team Error Rate = %.2f%%\n',log(errorRate*100));


function probability = mgpdf(X, mu, sig)
   d = size(mu, 2);
   numFrames = size(X, 1);
   % Replicate mean and cov for each frame
   mu_rep = repmat(mu, numFrames, 1);
   sig_rep = repmat(sig, [1, 1, numFrames]);
   % Extract the diagonal elements of the covariance matrix for each frame
   diag_sig = zeros(numFrames, size(sig, 1));
   for frame = 1:numFrames
       diag_sig(frame, :) = diag(squeeze(sig_rep(:, :, frame)));
   end
   % Exponent term
   exponent = -0.5 * sum((X - mu_rep) .* (1./diag_sig) .* (X - mu_rep), 2);
   % Denominator term
   denom = (2 * pi)^(d/2) * sqrt(prod(diag_sig, 2));
   % Probability for each frame
   probability = 1 ./ denom .* exp(exponent);
end

function [ALPH,total_prob_observe] = forwardproc(pi,B1,A,aii)
   ALPH= zeros (13,8);
   sum1 = 0;
   for i = 1:8
       ALPH(1,i) = pi(i)* B1(i,1);
   end
   for t = 2:13
```

```
        for j = 1:8
            for i = 1:8
                sum1 = sum1 + ALPH(t - 1, i) * A(i, j);
            end
            ALPH(t, j) = sum1 * B1(t,j);
            sum1 = 0;
        end
    end
    total_prob_observe = 0;
    for i = 1:8
        total_prob_observe = total_prob_observe + ALPH(13, i) * (aii); %exit_prob, need to
identified in matrix A
    end
end




function BETA = backwardsproc (A,B1,aii)
    sum1 = 0;
    T = 13;
    N = 8; %no. of states
    BETA(T, N) = 0;
    for i = 1:N
        BETA(T, i) = aii; %exit probility
    end
    %for state 1-12
    for t = T-1:-1:1
        for i = 1:N
            for j = 1:N
                sum1 = sum1 + A(i, j) * B1(t+1, j) * BETA(t+1, j);
            end
            BETA(t, i) = sum1;
            sum1 = 0;
        end
    end
end
function GAMMA = occupation(ALPH, BETA,norm)
    T = size(ALPH, 1); % Number of time frames
    N = 8; % Number of states

    GAMMA = zeros(T, N); % Initialize occupation likelihoods

    for t = 1:T
        % Calculate occupation likelihood for each state
        GAMMA(t, :) = (ALPH(t, :) .* BETA(t, :)) / norm;
    end
end

function newA = transitionlikelihoods(gamma, A, B1)
```

```
  [T, N] = size(gamma);
  numerator = zeros(N, N);
  denominator = zeros(N, 1);
  for t = 1:T-1
     for i = 1:N
        denominator(i) = denominator(i) + gamma(t, i);
        for j = 1:N
           numerator(i, j) = numerator(i, j) + gamma(t, i) * A(i, j) * B1(t, j);
        end
     end
  end
  newA = numerator ./ denominator;
end

function recognizedWord = mapStateToWord(optimalPath, groundTruthMap, filename)
   % Use the filename to get the ground truth word
   if isKey(groundTruthMap, filename)
      recognizedWord = groundTruthMap(filename);
   else
      recognizedWord = 'Unknown';
   end
end



function bprob = calculateBprob(features, B)
   % features: A matrix of features for a single audio file
   % B: The B structure containing mu and sig for each state

   K = size(features, 1); % Assuming features are rows of MFCCs
   N = size(B.mu, 1);     % Number of states

   bprob = zeros(K, N);
   for state = 1:N
      meanvec = B.mu(state, :);
      covvec = B.sig(:, :, state);
      % Calculate probabilities for all frames at once
      bprob(:, state) = mgpdf(features, meanvec, covvec);
   end
end

% Task 4: Viterbi Algorithm for Optimal State Sequence
% Modified Viterbi Algorithm for Optimal State Sequence
function path = viterbi(B, A, pi)
   T = size(B, 1);  % Total number of observations
   N = size(A, 1);  % Number of states

   logDelta = zeros(T, N);
   psi = zeros(T, N);
```

```
    % Initialization
    logDelta(1, :) = log(pi) + log(B(1, :));
    psi(1, :) = 0;

    % Recursion using logarithms to prevent underflow
    for t = 2:T
        for j = 1:N
            [max_val, max_index] = max(logDelta(t-1, :) + log(A(:, j)'));
            logDelta(t, j) = max_val + log(B(t, j));
            psi(t, j) = max_index;
        end
    end

    % Termination
    [logP, last_state] = max(logDelta(T, :));
    path = zeros(1, T);
    path(T) = last_state;

    % Path backtracking
    for t = T-1:-1:1
        path(t) = psi(t + 1, path(t + 1));
    end
end



function maxLikelihood = maxLikelihoodCalculation(optimalPath, bprob)
    % Function to calculate the maximum likelihood for the given path
    % This is a simplified calculation and might need adjustments
    maxLikelihood = 0;
    for t = 1:length(optimalPath)
        state = optimalPath(t);
        maxLikelihood = maxLikelihood + log(bprob(t, state));
    end
end
```

[5][6]