Report for Speech Recognition

EEEM030 - Speech Recognition - Group Assignment 2

Group 1: Xiaoguang Liang, Tofarati Onatunde, Zeyad Adelmonem and Jaison Baptist Sequira

December 6, 2024

Contents

1 Introduction **2**

2 Feature Extraction **3**
2.1 Significance of Feature Extraction in Speech Recognition . . . 3
2.2 Audio Data Setup . . . 3
2.3 MFCC Extraction Process . . . 3
2.3.1 Frame and Hop Size Parameters . . . 4
2.3.2 Hamming Window for Spectral Analysis . . . 4
2.3.3 MFCC Calculation . . . 4
2.4 Visualisation of Results . . . 4
2.4.1 Audio Waveform to MFCC Spectrogram . . . 4
2.4.2 Spectrogram of the Audio Signal . . . 4
2.4.3 Evolution of MFCC Coefficients . . . 4
2.4.4 Histogram of MFCC Coefficients . . . 5
2.4.5 3D Plot of MFCC Features . . . 5

3 Model initialization **7**
3.1 Paremeters of HMM initialization . . . 7
3.2 Initializaton for Multivariate Gaussian PDF . . . 8
3.2.1 Segments splitting for data . . . 8
3.2.2 Calculation for mean . . . 8
3.2.3 Calculation for variance . . . 9
3.3 Normalization and regularization . . . 9

4 Model training with HMMs **9**
4.1 Forward algorithm . . . 9
4.2 Backward algorithm . . . 10
4.3 Underflow problem . . . 11
4.4 Occupation likelihoods . . . 11
4.5 Transition likelihoods . . . 11
4.6 Re-estimation . . . 12

5 Evaluation **12**
5.1 Viterbi Algorithm . . . 12
5.2 Evaluation on the development set . . . 12
5.3 Evaluation on the evaluation set . . . 14

6 Data augmentation **14**

7 Conclusion **15**

A Appendix: Structure of the codes **18**

B Appendix: Codes 19

Abstract

Abstract This report presents the implementation of a speech recognition system based on a Hidden Markov Model (HMM) for isolated word recognition. Utilizing precomputed Mel-Frequency Cepstral Coefficients (MFCCs) as features, the system incorporates feature normalization, parameter initialization for the HMM, sequence decoding via the Viterbi algorithm, and comprehensive evaluation metrics. The system's performance is assessed through a recognition error rate and a confusion matrix, providing insights into its effectiveness in identifying words from a limited vocabulary. The implementation demonstrates the application of HMMs in speech recognition and highlights the impact of key steps such as initialization and decoding on system accuracy.

1 Introduction

This assignment involves designing and implementing a basic speech recognition system using fundamental speech and audio processing techniques. The system, developed in MATLAB, performs isolated-word recognition for a vocabulary of eleven words, structured around five key tasks: feature extraction, model initialisation, training, evaluation, and testing. HMMs form the core methodology, providing a framework to explore the impact of maximum likelihood training on system performance. The recogniser uses the Baum-Welch algorithm and the Viterbi algorithm for decoding and recognising these words. The activities include within this coursework include:

- **MFCC Feature Extraction:** for extracting acoustic features for training, evaluation, and additional test datasets.
- **HMM Initialisation and Training:** For initialising the prototype HMMs using global statistics of the training data. In addition to iteratively refining model parameters using forward-backward algorithms to compute likelihoods and re-estimate transitions and emissions.
- **Recognition and Evaluation:** for calculating the Viterbi likelihoods and decoding the recognition outputs. In turn scoring the system performance via confusion matrices and error rate calculations.

The contribution of members of our group is shown in Table 1 and Table 2.

Table 1: Contribution list of the written report

Section	Name
Abstract	Tofarati Onatunde
1 Introduction	Tofarati Onatunde
2	Tofarati Onatunde
3	Xiaoguang Liang
4	Zeyad Adelmonem
5	Jaison Baptist Sequira
6	Xiaoguang Liang
7 Conclusion	Tofarati Onatunde

Table 2: Contribution list of codes

Task	Name
1	Tofarati Onatunde
2	Xiaoguang Liang
3	Zeyad Adelmonem & Xiaoguang Liang
4	Jaison Baptist Sequira & Zeyad Adelmonem & Xiaoguang Liang
5	Xiaoguang Liang

2 Feature Extraction

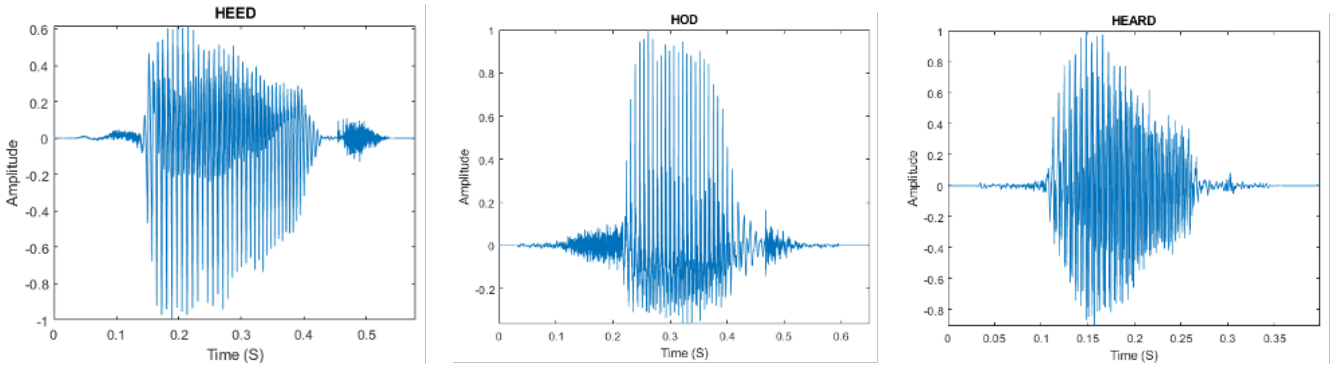
2.1 Significance of Feature Extraction in Speech Recognition

Feature extraction is an essential step in developing a speech recogniser system, as it transforms raw audio signals into a meaningful and compact interpretations that can be effectively processed by the speech recognition model [1].

Therefore, this section of the report will provide an overview of the underlying principles, processes and parameters used to extract 13 MFCCs including the zeroth coefficient, from the development and evaluation audio files sets, while supplementing relevant images of key results [1].

Before, explaining how the 13 MFCCs were extracted, it is vital to note that MFCC's are widely regarded as the standard interpretation for speech recognition. This is because MFCC's closely mimic the way the human ear process sound, by capturing the frequency spectrum of the audio signals on the Mel scale that closely aligns with the human auditory sensitivity to sound frequencies, while actively discarding irrelevant information such as pitch variations. Which works to retain essential phonetic features needed for distinguishing between speech sounds [1].

2.2 Audio Data Setup



(a) sp15a_w02_heed.mp3 (b) sp01a_w02_hod.mp3 (c) sp30a_w02_heard.mp3

Figure 1: Audio Signals of sound samples from the Developmet data set.

As shown in Appendix B Code 6, the MATLAB code begins by defining 30 speakers used in the development data set and listing their repeated eleven-word vocabulary collections: "heed, hid, head, had, hard, hud, hod, hoard, hood, whod, and heard". Resulting in the 330 audio files showcasing variations in vowel sounds, constants and language articulations, ideal for studying phonetic features, visually represented by figure 1.

Note: As shown in Appendix B, two `for` loops were used which generated filenames for all speaker-word combinations. Resulting in paths to all audio files, ensuring an organised dataset for subsequent processing. Each file followed the naming convention `spXXa_wYY_word.mp3`, where: "XX" is the speaker index, "YY" is the word index, and "word" is the spoken terminology.

2.3 MFCC Extraction Process

Therefore, to initialise the feature extraction of the 13 MFCCs, including the zeroth coefficient, from the development set, and later the evaluation set, a few steps had to be taken, and are described below.

2.3.1 Frame and Hop Size Parameters

The process begins with segmenting the audio signals into overlapping frames, each 30ms in duration. This temporal resolution corresponds to the typical length of phonemes, ensuring that each frame captures meaningful variations in speech without losing important details. Additionally, a hop size of 10ms with a 20ms overlap between consecutive frames was implemented. As this provided smooth transitions and comprehensive temporal coverage between frames. Overlapping frames are crucial in speech processing as they capture the continuity of spoken words, avoiding information loss between frames.

2.3.2 Hamming Window for Spectral Analysis

Note: A Hamming window was also applied to each frame to reduce spectral leakage caused by transitions and abrupt frame boundaries. This step was utilised to improve the accuracy of frequency domain analysis, ensuring that the extracted features are reliable and precise.

2.3.3 MFCC Calculation

Finally, a built-in MATLAB function was implemented to extract the final MFCC values from each frame. Providing a compact yet detailed representation of the speech signal that focusing on the spectral content most relevant to human perception. It utilised the previously specified frame size, hop size, and the Hamming window to ensure high-quality feature extraction. The resulting 13-dimensional feature vector for each frame captures the static spectral characteristics of the speech signal. Then process was repeated for the evaluation set of audio samples.

Take note that delta and delta-delta features (velocities and accelerations) were excluded from the overall calculation of the MFCC's. This simplification allows the code to focus on the static spectral characteristics of the audio signals. Which in turn improves computational efficiency while also maintaining sufficient information for effective speech recognition.

2.4 Visualisation of Results

After extracting the MFCC features for development and evaluation audio sets, several visualisations were created to illustrate the properties of the extracted features. Note: These graphs shown in this subsection were produced for all audio samples. However, only images pertaining to audio `sp01a_w01_heed.mp3` will be shown for simplification purposes.

2.4.1 Audio Waveform to MFCC Spectrogram

By sourcing the length of the audio signal and plotting it against its linear amplitude, a graph (Figure 2) providing a time-domain representation of the heed speech signal showcasing its temporal variations and intensity was made.

After putting the original audio signal, through the MFCC process described in *Section 2.3*, the corresponding MFCCs can be displayed in a spectrogram-like image (Figure 3). Where the x-axis represents the frame index (time), and the y-axis illustrates the MFCC coefficients and the colour intensity indicates the magnitude. This visualisation provides a view of how the audios spectral content evolves over time.

2.4.2 Spectrogram of the Audio Signal

Additionally, another spectrogram (Figure 4) can be made to visualises the frequency domain of the signal over time. In turn, offering insights into the full frequency spectrum of the audio data.

2.4.3 Evolution of MFCC Coefficients

Moreover, the evolution (Figure 5) of the first MFCC coefficient can be plotted over time to highlight its dynamic behaviour across frames. This helps to emphasis the temporal variations in spectral content, which can be utilised when distinguishing between different speech sounds.

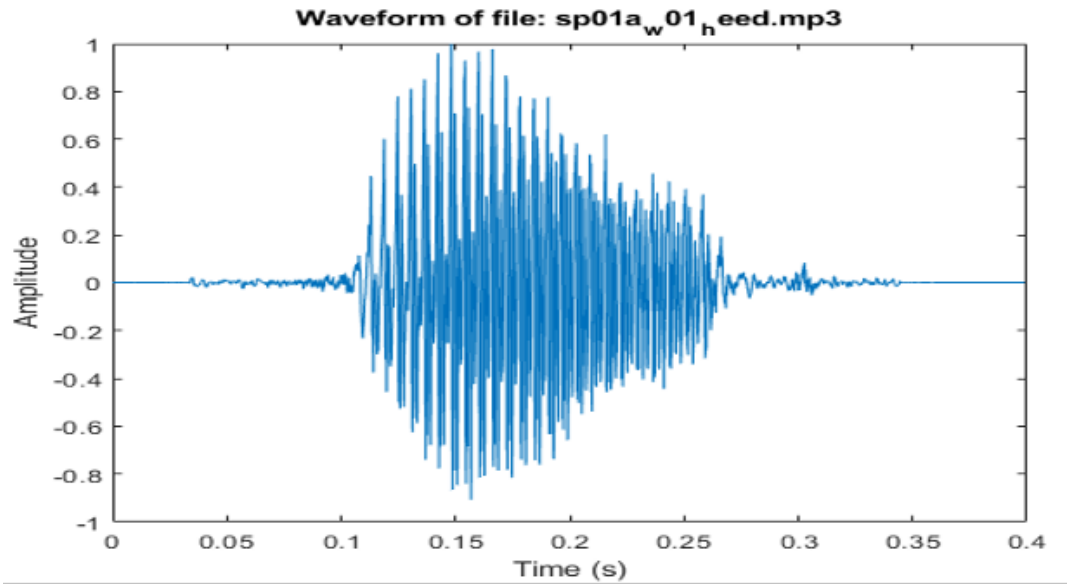


Figure 2: Audio Waveform of sp01a_w01_heed.mp3.

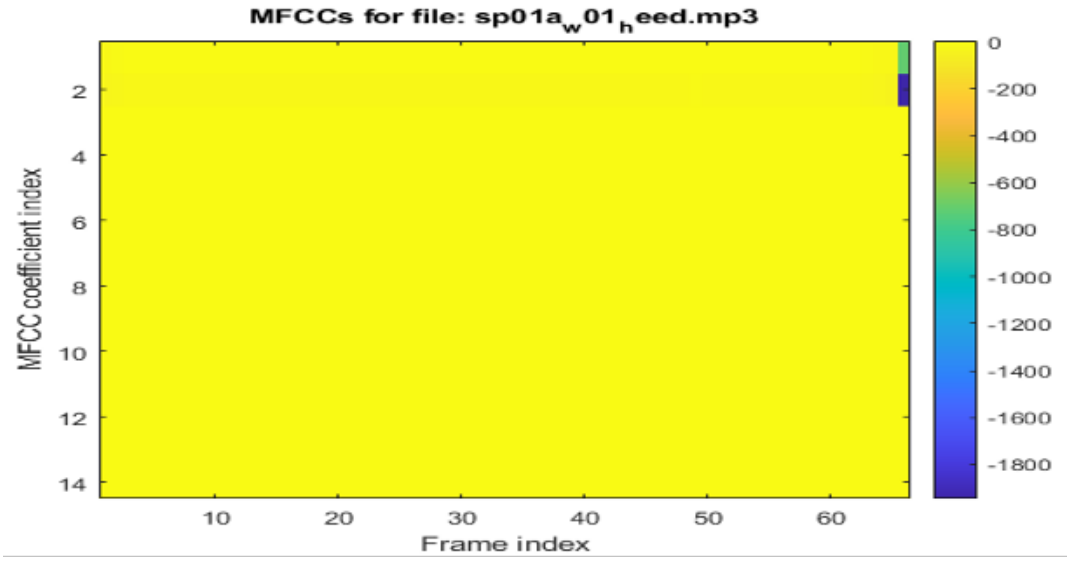


Figure 3: MFCC spectrogram of sp01a_w01_heed.mp3.

2.4.4 Histogram of MFCC Coefficients

The histogram (Figure 6) of the first MFCC coefficient shows its distribution across frames. This statistical representation reveals the range and frequency of phonetic feature values, thus providing insight into the acoustic properties of each speech signal.

2.4.5 3D Plot of MFCC Features

Lastly, a 3D mesh plot (Figure 7) can be used to visualise the first three MFCC coefficients across time. Thus, illustrating the relationships between coefficients and their temporal evolution, offering a more comprehensive view of the feature space.

These images are useful in analysing and refining the extracted features, because:

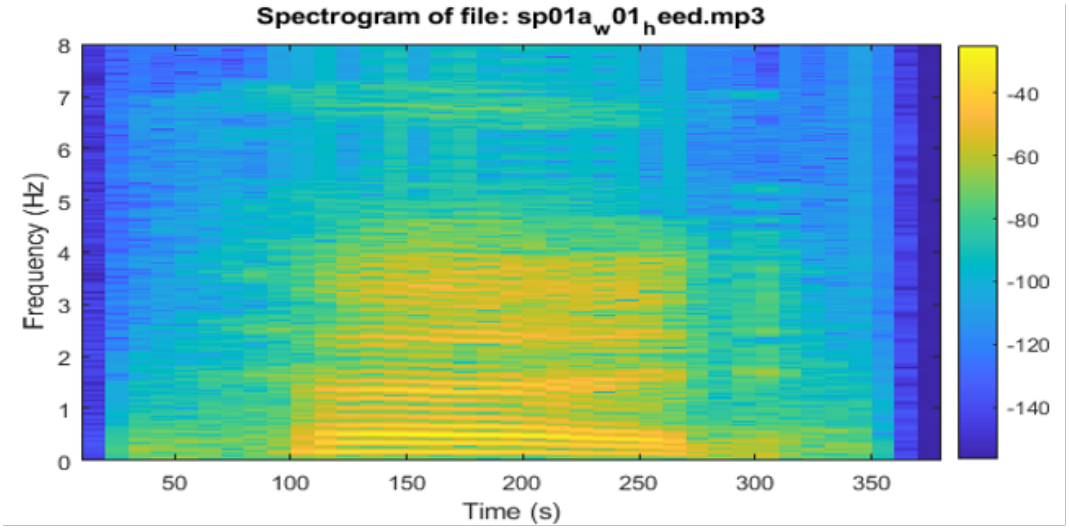


Figure 4: MFCC spectrogram of sp01a_w01_heed.mp3.

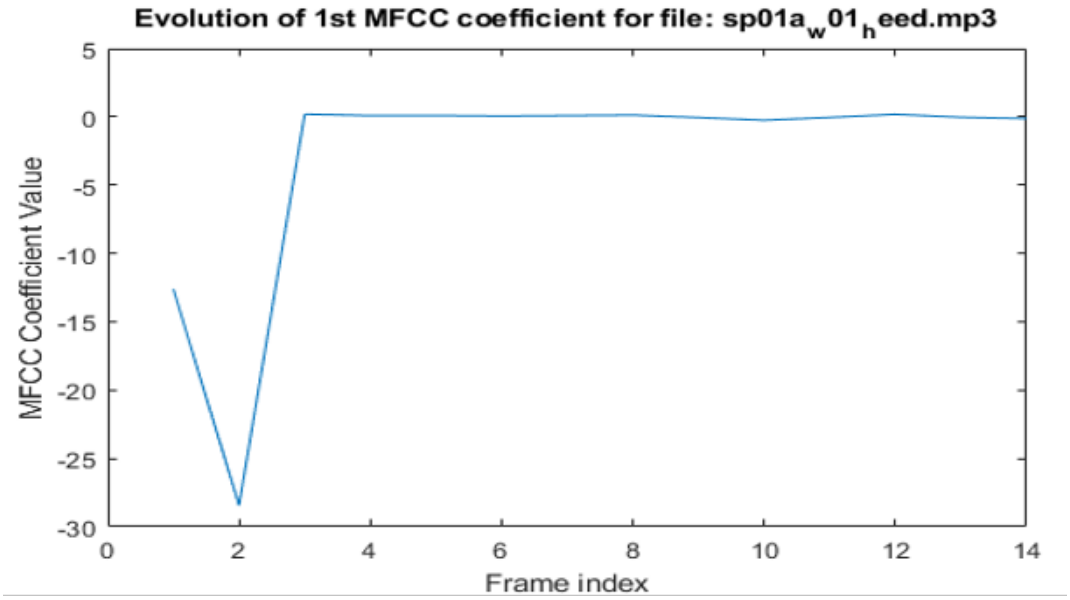


Figure 5: Evolution of the first MFCC coefficent.

- (a) The MFCC spectrogram highlights phonetic variations.
- (b) The waveform and audio spectrogram offer complementary views of the signal's temporal and frequency characteristics.
- (c) The coefficient evolution and histograms reveal trends and statistical properties, aiding in fine-tuning recognition models.

In turn allowing us to understand MFCC features and how they evolve over time. They provide valuable insights for further model development, enabling applications such as automatic speech recognition (ASR), speaker identification, and audio classification.

To conclude this section, feature extraction detailed lays the foundation for an effective speech recognition system. Demonstrating the power of how extracting the MFFC's creates a well-suited starting point for capturing meaningful information from speech signals and training HMMs, of which are detailed in later sections of this report.

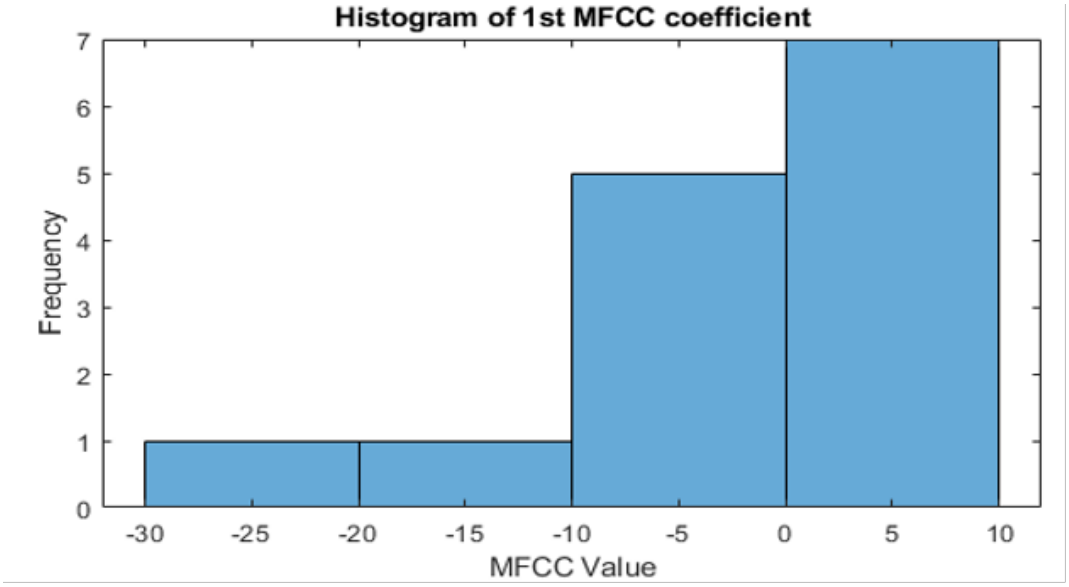


Figure 6: Histogram of sp01a_w01_heed.mp3 MFCC coefficents.

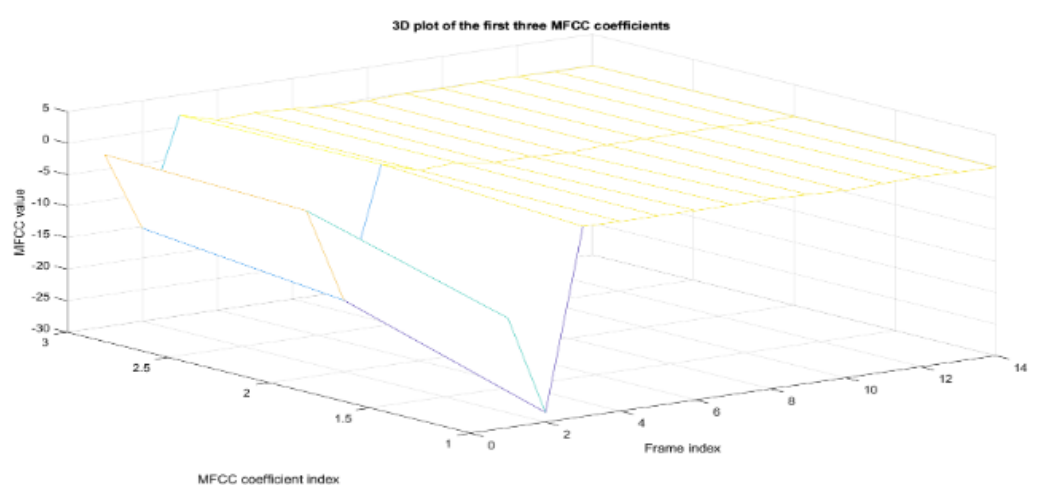


Figure 7: 3D Plot of sp01a_w01_heed.mp3 MFCC Features.

3 Model initialization

3.1 Paremeters of HMM initialization

Model initialization represents a fundamental stage in the training of HMMs. The efficacy of training algorithms such as the Baum-Welch algorithm is contingent upon the initialisation of model parameters in an optimal manner. These parameters encompass the initial state probabilities, represented by the symbol π, the state transition probabilities A, and the observation probabilities B. The implementation of an appropriate initialisation strategy provides a robust foundation for optimisation, whilst simultaneously preventing the convergence to suboptimal solutions.

Initial State Probabilities Π: $\Pi = [\pi_1, \pi_2, \ldots, \pi_N]$ represents the probability of starting in each state i , where N is the number of states. As indicated in the assignment document, the initial value of Π is provided as follows:

$$\pi_i = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{1}$$

State Transition Probabilities A: $A = [a_{ij}]$ represents the probability of transitioning from state i to state j, satisfying $\sum_{j=1}^{N} a_{ij} = 1$ for all i. Here, initial A is provided in the assigment document:

$$a_{ij} = \begin{bmatrix} 0.8 & 0.2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.8 & 0.2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.8 & 0.2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.8 & 0.2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.8 & 0.2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.8 & 0.2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.8 & 0.2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.8 \end{bmatrix} \tag{2}$$

Observation Probabilities B: B represents the probability of observing data given a state. For continuous observations, this is computed using a multivariate Gaussian probability density function PDF.

The observation probability for a given state i and observation x is defined as [2]:

$$B_i(x) = \frac{1}{(2\pi)^{d/2}|\Sigma_i|^{1/2}} e^{-\frac{1}{2}(x-\mu_i)^T \Sigma_i^{-1}(x-\mu_i)} \tag{3}$$

where:

- μ_i is the mean vector for state i,
- Σ_i is the covariance matrix for state i,
- d is the dimensionality of the observation vector.

In practice:

- μ_i can be initialized as the global mean of the observations.
- Σ_i can be initialized as a diagonal matrix where diagonal entries are the variances of the clustered observations.

3.2 Initializaton for Multivariate Gaussian PDF

In models with multivariate Gaussian probability density functions (PDFs), the initialisation of model parameters is of critical importance for the success of the training process, particularly when using iterative algorithms such as the Baum-Welch algorithm. The process of initialization provides the model with initial estimates for the emission probabilities, which are represented as Gaussian distributions, as well as the transition probabilities and initial state probabilities [3].

The initialisation of the multivariate Gaussian PDF necessitates the estimation of the mean vector, designated as μ, and the covariance matrix, designated as Σ, for each state within the HMM. These parameters serve to characterise the probability distribution of the observed data within each state [2].

3.2.1 Segments splitting for data

Divide the length of the feature data evenly into N parts, where $N = 8$, representing the number of hidden states. This converts the training samples into an $8N \times 13$matrix, facilitating the computation of the Multivariate Gaussian PDF.

3.2.2 Calculation for mean

To meet the requirements of this assignment, the feature data must be divided into N segments to calculate a 13×13 mean matrix. Each segment is processed iteratively to compute the mean. For a matrix X of size $m \times n$, the mean of each row is:

$$\mu_i = \frac{1}{n} \sum_{j=1}^{n} X_{i,j}, \quad i = 1, 2, \ldots, m \tag{4}$$

3.2.3 Calculation for variance

To meet the requirements of this assignment, the diagonal of the 13×13 covariance matrix is computed by splitting the feature data into N segments, following the same approach as used for calculating the mean. Each segment is processed iteratively to compute the covariance matrix. For a matrix X of size $m \times n$, the variance of each row is:

$$Varrow(X) = \frac{1}{n} \sum j = 1^n \left(X_{ij} - \mu_i\right)^2, \quad i = 1, 2, \ldots, m \tag{5}$$

where μ_i is the mean of the $i\text{-}th$ row.

3.3 Normalization and regularization

Normalization: Normalize the input data to have zero mean and unit variance to ensure robust parameter estimation. In practice, Min-Max Normalization is implemented. Scales all elements to the range [0, 1] like this:

$$\text{Normalized Value} = \frac{\text{Value} - \text{Min}}{\text{Max} - \text{Min}} \tag{6}$$

Regularization: Regularizing the covariance matrix can prevent issues like singular matrices or overfitting. A small value epsilon is added to the diagonal elements of the covariance matrix:

$$\text{Regularized Matrix} = \text{Covariance Matrix} + \epsilon \cdot I \tag{7}$$

where I is the identity matrix.

4 Model training with HMMs

The Baum-Welch approach, which effectively estimates model parameters through an iterative Expectation-Maximization process, is the mainstay of HMM training. In order to prevent numerical underflow, we used a scaling factor. The algorithm uses both the Forward and Backward procedures to compute intermediate variables. These calculations make it possible to calculate two essential elements that are essential for parameter re-estimation: occupation likelihoods (state probabilities) and transition likelihoods (state transition probabilities). Using the determined likelihoods to optimize the likelihood of witnessing the training sequences, the procedure iteratively finetunes the parameters (transition and emission probabilities) until convergence as shown in Figure 8.

4.1 Forward algorithm

By employing dynamic programming, the Forward algorithm effectively determines the probability of an observation sequence in a Hidden Markov Model without resorting to the exponential difficulty of adding up every potential state sequence. It functions in three essential steps:

Initialization Step: The procedure calculates the initial probability for each state □ for the first observation by multiplying two terms: the likelihood that state I would emit the first observation $b_i(o_1)$ and the probability of starting in state $i(\pi_i)$ [4].

$$\alpha_1^r(i) = \pi_i b_i(o_i^r) \tag{8}$$

Recursion Step: The algorithm considers every path that could lead to each current state for each succeeding time step. It calculates the probability for each state j at time t by taking into account Previous state probabilities $\alpha_{t-1}(i)$. Transition probabilities a_{ij} and Emission probability of current observation $b_j(o_t^r)$.

$$\alpha_1^r(j) = \left[\sum_{i=1}^{N} \alpha_{t-1}^r a_{ij}\right] b_i(o_i^r), \quad t = 1, 2, \ldots, T \tag{9}$$

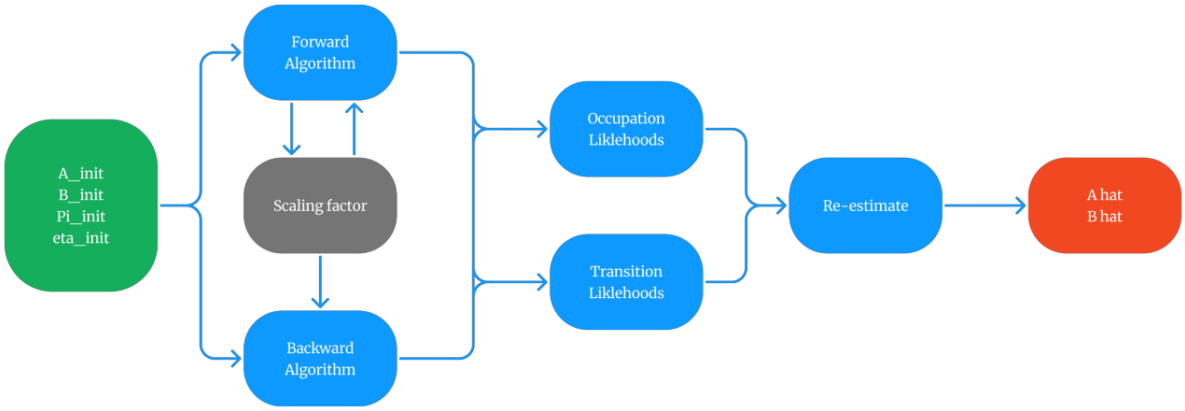


Figure 8: HMM training algorithm.

Termination Step: The terminal forward variables for each of the possible end states are added up to determine the final probability of the complete observation sequence.

$$P^r = P(\mathcal{O}|\lambda) = \sum_{i=1}^{N} \alpha_T^r(i)\eta_i \tag{10}$$

Without explicitly listing every conceivable state sequence, this method effectively builds on previously determined values to compute the overall likelihood by accumulating probabilities throughout the process.

4.2 Backward algorithm

Similar to its Forward counterpart, the Backward method uses dynamic programming to calculate the likelihood of a sequence in a Hidden Markov Model, but it operates in the opposite way. It functions as follows:

Initialization Step: The procedure initializes the backward variables for every state with a value of η at the last observation time T. Our starting point for moving backward through the sequence is established by this initialization.

$$\beta_T^r(i) = \eta_i \tag{11}$$

Recursion Step: The Backward algorithm's recursive computation from the final time step backwards is the fundamental component of its dynamic programming methodology. By taking into account all potential future routes, the algorithm determines the probability of producing the remaining observations for each state at each time step prior to T. We take into consideration the probability of transitioning to any possible next state, the probability that the next state would emit the next observation we actually see, and the probability of generating all remaining observations from that next state in order to account for all possible paths leading from each current state. By going backwards through the observation series, this recursive approach effectively combines probability without explicitly listing every possibility, accumulating information about likely state sequences.

$$\beta_1^r(i) = \sum_{j=1}^{N} a_{ij} b_j(o_{t+1}^r)\beta_{t+1}^r(j), \quad t = T-1, T-2, \ldots, 1 \tag{12}$$

Termination Step: The Backward algorithm's Termination step calculates the overall probability of the observation sequence by considering the initial state probabilities. After computing all backward variables back to time $t = 1$, we can obtain the total probability of our observation sequence by combining the initial state probabilities, the emission probabilities of the first observation, and the backward variables at time $t = 1$ for all possible states. This combination gives us the same total probability as the Forward algorithm, just computed in the reverse direction.

$$P^r = \sum_{i=1}^{N} \pi_i b_i(o_1^r) \beta_1^r(i) \tag{13}$$

4.3 Underflow problem

Both forward and backward algorithms in HMM encounter underflow issues when they constantly multiply probabilities, producing incredibly small quantities that are impossible for computers to process correctly. We employ scaling factors at each time step t to solve this. To maintain the values within a calculable range, we multiply the forward variables $\alpha_t^r(i)$ and backward variables $\beta_t^r(i)$ by a scaling coefficient c_t. To ensure consistency, the same scaling factors are applied to both algorithms, and these scaling coefficients are utilized to determine the final probability [5].

Forward scaling:

$$c_t = \frac{1}{\sum_{i=1}^{N} \alpha(i)} \tag{14}$$

$$\alpha(i) = c_t * \alpha(i) \tag{15}$$

Backward scaling:

$$\beta_t(i) = c_t * \beta(i) \tag{16}$$

4.4 Occupation likelihoods

complete observation sequence, is estimated in HMM after forward and backward variables have been calculated. It integrates data from backward variables $\beta_t(i)$, which takes future observations into account, and forward variables $\alpha(i)$, which takes observations up to time t into account. Each state's forward and backward variables are multiplied, and the result is normalized by the observation sequence's overall probability. Considering every observation, this gives a comprehensive picture of state occupancy. Understanding the most likely state sequence that produced the observations and re-estimating parameters in the Baum-Welch algorithm depend on these occupation probabilities.

$$\gamma_t(i) = \frac{\alpha_t(i)\beta(i)}{P(\mathcal{O}|\lambda)} \tag{17}$$

4.5 Transition likelihoods

Given the complete observation sequence and model parameters, transition likelihood estimation determines the probability of being in state i at time t and changing to state j at time $t + 1$. Forward variables α_{ij}, backward variables $\beta_j(o_{t+1})$, transition probabilities $a_{ij}$and emission probabilities $\beta_j(o_{t+1})$ are used to calculate this probability. Utilizing forward variables, the computation takes into account the likelihood of staying in state i until time t, transitioning to state j, producing the observation at $t + 1$, and then utilizing backward variables to account for all further observations from $t + 1$ onward. During the parameter re-estimation stage of the Baum-Welch method, these transition likelihoods are essential for updating the transition probability matrix.

$$\xi_t(i, j) = \frac{\alpha_{t-1}(i) a_{ij} b_j(o_t) \beta_t(j)}{P} \tag{18}$$

4.6 Re-estimation

Using the occupation and transition likelihoods that were previously determined, the re-estimation stage of the HMM (Baum-Welch algorithm) modifies the model parameters $\lambda = \{A, B, \pi\}$. The occupation likelihood at $t = 1$ is used to update the initial state probabilities π. To re-estimate the transition probabilities A, $\xi_t(i, j)$ and $\gamma_t(i)$ are used to calculate the ratio of expected transitions between states to total transitions from the source state. By using $\gamma_t(i)$ to compare observation occurrences in each state to total state occupations, the emission probabilities B are updated. The new model γ' formed by these modified parameters raises the likelihood $P(\mathcal{O}$ till convergence.

5 Evaluation

5.1 Viterbi Algorithm

The Viterbi algorithm is used for decoding the sequence of hidden states given the observation sequence. The algorithm uses dynamic programming to compute the maximum likelihood of the entire sequence by recursively calculating the most probable state at each time step.

Steps of the Viterbi Algorithm:

1. **Initialization:** At time t=1, the probability of each state is computed using the initial state probabilities Π and the emission probabilities B.
2. **Recursion:** For each subsequent time step t, the algorithm computes the maximum probability of each state by considering all transitions from previous states. This is done by combining the previous state's probability and the transition probability.
3. **Termination:** After processing all time frames, the algorithm selects the state with the highest probability at the final time frame.
4. **Backtracking:** The state sequence is traced back using the back pointer matrix, which keeps track of the state transitions that led to the optimal path.

Numerical Stability: To avoid underflow issues, probabilities are represented in logarithmic space. This ensures that the product of small probabilities does not result in numerical errors. Instead of multiplying probabilities, we add their logarithms.

By monitoring the optimal route to every state at every time step, the Viterbi algorithm determines the most likely state sequence in HMM. It employs a dynamic programming technique with two primary variables: $\psi_t(i)$ holds the prior state that resulted in this highest probability, and $\delta_t(i)$ indicates the highest likelihood of a path terminating in state i at time t. As it advances through time, the algorithm uses emission probabilities $b_j(o_t)$ and transition probabilities a_{ij} to calculate these variables. It requires scaling in order to avoid underflow, just like forward-backward algorithms. It finds the ideal state sequence by going back through $\psi_t(i)$ after arriving at the last observation. We have used logarithmic transformation and sum of logs instead of scaling as in forward and backward algorithms [6].

5.2 Evaluation on the development set

Upon analyzing the data, the development set consists of 30 groups, each containing 11 samples representing the pronunciations of 11 words spoken by the same individual. The dataset includes three voice types: male, female, and children. Specifically, there are 14 groups of male voices, 15 groups of female voices, and 1 group of children's voices. To ensure a balanced distribution of training and testing data across different voice types, the development set is split based on gender. Since there is only one group of children's voices, it is included in the training set. The training set comprises 80% of the development set.

Recognition error rate: This is the percentage of misclassified predictions in the test set compared to the ground truth. The formula for calculating the error rate is:

$$ErrorRate = \frac{NumberofIncorrectPredictions}{TotalPredictions} \tag{19}$$

Confusion Matrix: A confusion matrix is a table used to describe the performance of a classification model. Each row represents the true labels, while each column represents the predicted labels. It helps visualize the types of misclassifications made by the model.

In accordance with the specifications set forth in the assignment, the HMM model was trained for a total of 15 epochs. The error rates are calculated on the basis of the outputs of the Viterbi algorithm, and the resulting error rates on the development set are presented in the accompanying Figure 9 and the error rates data is in the Table 3.

From Figure 9, it can be seen that the error rates fluctuate around 0.4 as the epochs increase, reaching the lowest value of 0.363636 at the 11th epoch. This indicates that the model's performance does not necessarily improve with an increasing number of epochs. The reasons for this include the following:

- **Overfitting:** As the training progresses, the model may start overfitting the training data, capturing noise or irrelevant patterns instead of generalizable features. This reduces its ability to perform well on unseen data.
- **Local Minima or Suboptimal Convergence:** The optimization process (e.g., Baum-Welch algorithm) may converge to a local minimum or suboptimal solution. This could prevent the model from finding the best parameter settings, even with more epochs.
- **Lack of Sufficient Data Augmentation:** If the dataset is not sufficiently diverse or augmented, the model may not learn robust features. Consequently, additional training epochs will not improve performance significantly.
- **Learning Rate and Parameter Initialization:** Poorly chosen learning rates or suboptimal initial parameter values (e.g., transition probabilities, emission probabilities) can lead to slow or ineffective learning during training.
- **Plateauing of Performance:** HMMs may reach a plateau in performance, where further training does not lead to significant improvement because the model has already captured all the learnable patterns in the data.

Table 3: Error rates for HMM model training on the development set

Epoch	Error rate
1	0.393939
2	0.393939
3	0.5
4	0.530303
5	0.424242
6	0.5
7	0.409091
8	0.454545
9	0.393939
10	0.424242
11	0.363636
12	0.424242
13	0.424242
14	0.515152
15	0.424242

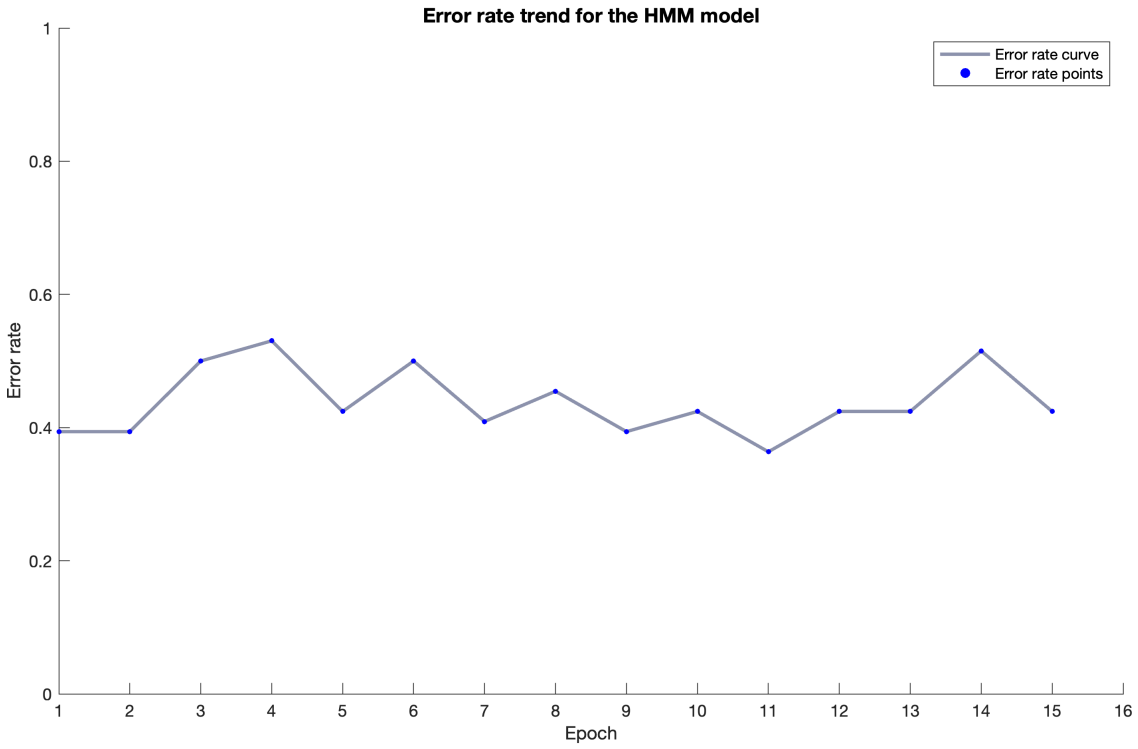


Figure 9: Error rate trend for HMM model training on the development set.

5.3 Evaluation on the evaluation set

The evaluation is conducted on the evaluation set provided in the assignment. To ascertain the accuracy of the HMM model, the maximum cumulative likelihoods are initially calculated using the Viterbi algorithm. Subsequently, the accuracy is calculated based on the output of the Viterbi algorithm. The most optimal HMM model exhibits an accuracy of 0.318182 on the evaluation set. The equation for calculating accuracy is as follows:

$$Accuracy = \frac{NumberofCorrectPredictions}{TotalPredictions} \tag{20}$$

The recognition outputs should be evaluated and a confusion matrix generated. The corresponding confusion matrix is presented in graphical form in Figure 10.

6 Data augmentation

By analyzing the characteristics of the training and validation datasets, as well as the confusion matrix of the predictions on the validation dataset, strategies for optimizing the model's performance can be developed. The following outlines the optimization of the model's performance using data augmentation methods.

Characteristics of the validation data: The pronunciations in the validation dataset consist entirely of male voices across 11 groups of words.

Evaluation results of the validation data:

- The prediction accuracy for ”had”, ”hard”, ”heed”, ”hod”, ”hud” and ”whod” is 0

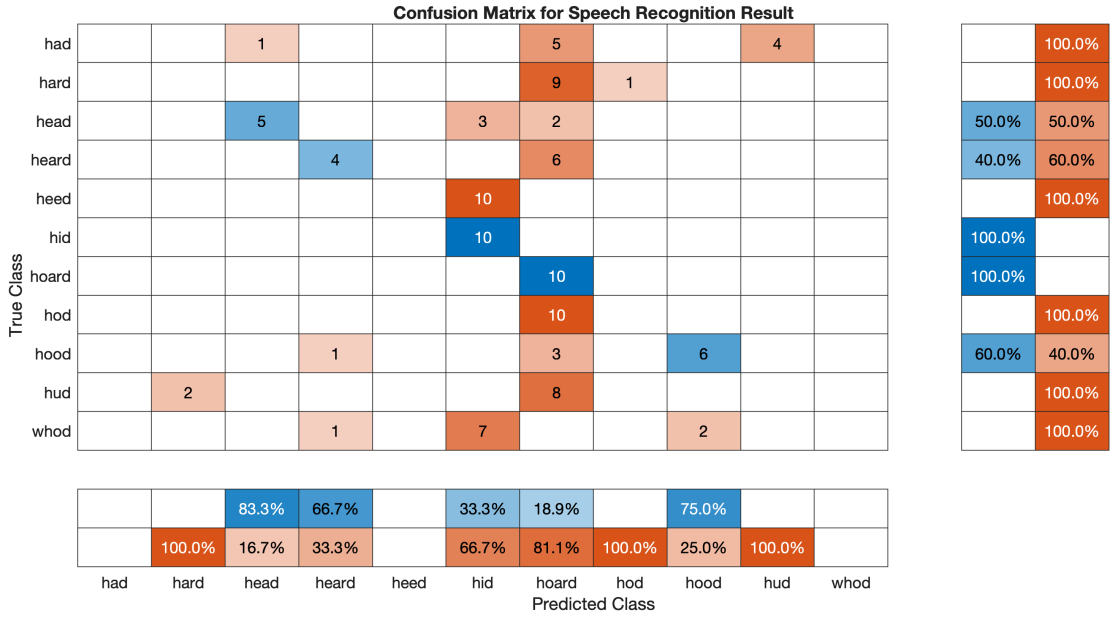


Figure 10: Confusion matrix for HMM model training on the evaluation set.

- The prediction accuracy for “hid” and “hoard” is relatively high, at 100%.

Augmentation measures for the Data:

1. Train only on male voices.
2. Perform 3x or 5x repetition augmentation for male voice recordings.
3. Perform 3x or 5x repetition augmentation for audio samples of the words ”had”, ”hard”, ”heed”, ”hod”, ”hud” and ”whod”.

Following the application of data augmentation techniques, the resulting recognition outputs are evaluated using the optimal HMM model. The associated confusion matrix is presented in Figure 11.

The confusion matrix serves to illustrate that the application of data augmentation in accordance with the specific characteristics of the dataset can result in a notable enhancement of the model’s performance. In the future, further speech data augmentation techniques, such as time stretching, time shifting and pitch shifting, can be investigated in order to enhance the results even further.

7 Conclusion

This report highlights the application and evaluation of the Hidden Markov Models (HMMs) in speech recognition. By emphasising the usability of the Backward Algorithm’s role in probability computation as well as Forward and Viterbi algorithms, Group 1 was able to integrate theoretical speech recognition concepts with practical applications. Thus, illustrating the capabilities and constraints of HMM-based speech recognition with the provided speech data. The model’s performance had a varying degree of success and failure when recognising the 11-word vocabulary, with a 0% predicted accuracy with had”, ”hard”, ”heed”, ”hod”, ”hud” and ”whod” , and a 100% predicted accuracy with hid” and “hoard”. This was likely due to overfitting, suboptimal convergence, and limited data diversity.

To conclude, this project has offered valuable insights into the effectiveness of maximum likelihood training and its impact on improving speech recognition. Moreover, the use of data augmentation, of which targeted under-represented speech patterns, showed promise in enhancing accuracy. This suggests value of further exploration

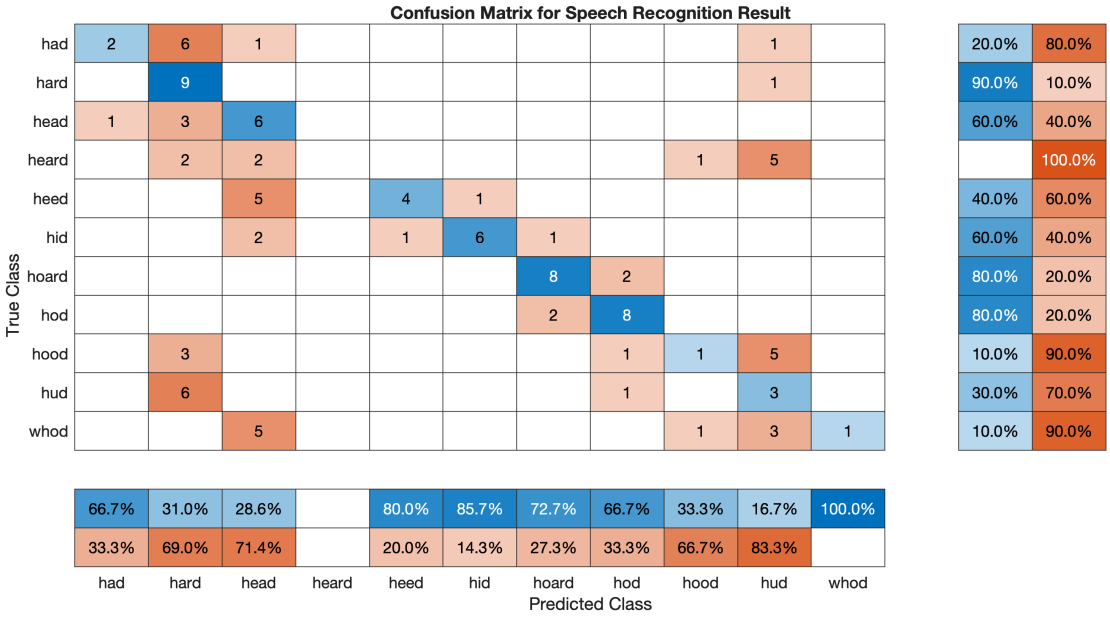


Figure 11: Confusion matrix for HMM model training on the data augmentation set.

into techniques like time stretching and pitch shifting. Therefore, for future work, when looking to improve results, the integration of diverse datasets, advanced augmentation, and optimised model parameters is recommended. This could potentially be done by combining HMMs with modern machine learning methods for thorough speech recognition.

References

[1] W. Han, C.-F. Chan, C.-S. Choy, and K.-P. Pun, “An efficient mfcc extraction method in speech recognition,” in *2006 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 4 pp.–, 2006.

[2] P. Jackson, “Speech and language processing.” [Lecture], 2024.

[3] S. Young, G. Evermann, M. Gales, T. Hain, D. Kershaw, X. Liu, G. Moore, J. Odell, D. Ollason, D. Povey, V. Valtchev, and P. Woodland, *The HTK book*. 01 2002.

[4] D. Jurafsky and J. H. Martin, *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition with Language Models*. 3rd ed., 2024. Online manuscript released August 20, 2024.

[5] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006.

[6] T. S. R. Z. H. Z. E. P. Xing, “Case studies: Hmm and crf.” [Lecture], 2020.

A Appendix: Structure of the codes

```
.
├── HMM_model                                    # HMM model
│   ├── Baum_Welch.m                             # Baum-Welch algorithm
│   ├── compute_backward_likelihood.m            # compute backward likelihood
│   ├── compute_error_rate.m                     # compute error rate
│   ├── compute_forward_likelihood.m             # compute forward likelihood
│   ├── compute_occupation_likelihood.m          # compute occupation likelihood
│   ├── compute_pdf.m                            # compute pdf
│   ├── compute_pdf_gm.m                         # compute pdf of gaussian mixture model
│   ├── compute_probability.m                    # compute probability
│   ├── compute_transition_likelihood.m          # compute transition likelihood
│   ├── obtain_likelihoods.m                     # obtain likelihoods
│   ├── plot_errors.m                            # plot errors
│   ├── speech.mlx
│   └── viterbi.m                                # viterbi algorithm
├── README.md                                    # Readme file
├── compute_confusion_matrix.m                   # compute confusion matrix
├── conf
│   └── GlobalSetting.m                          # global setting
├── data                                         # data
│   ├── EEEM030cw2-DevelopmentSet-2024
│   ├── EEEM030cw2-EvaluationSet-2024
│   ├── data_augmentation
│   ├── evaluationData.mat
│   ├── graphs
│   │   ├── Error_rate_trend_for_HMM_model.png
│   │   └── confusion_matrix.png
│   ├── hmm_models
│   ├── recognition_result.mat
│   ├── testData.mat
│   └── trainData.mat
├── document                                      # Report
│   ├── Enhanced Report.docx
│   ├── Report for Speech Recognition.tex         # LaTeX source code
│   ├── bibliography
│   ├── refs.bib
│   └── speech.docx
├── feature_extraction
│   ├── mfccFeature.m                             # MFCC feature extraction
│   └── shared_feature_extraction.mlx             # Feature extraction shared function
├── model_initialization                          # HMM model initialization
│   ├── Gussian_mix.m                             # Gaussian mixture initialization
│   ├── init_hmm.m                                # HMM initialization
│   ├── meanVariance.m                            # Mean and variance calculation
│   ├── normalizeMatrix.m                         # Matrix normalization
│   └── regularizeCovariance.m                    # Covariance regularization
├── preprocessing                                 # Data preprocessing
│   ├── data_augmentation.m                       # Data augmentation
│   ├── data_exploration.m                        # Data exploration
│   └── make_train_evaluation_data.m              # Make train and evaluation data
├── test                                          # Test
│   ├── test.m
│   └── testMfccFeature.m
├── test_model.m                                  # Test model
└── train_model.m                                 # Train model
```

Figure 12: Structure of the codes.

B Appendix: Codes

Listing 1: train_model.m

```
%% Author: Xiaoguang Liang (PG/T - Comp Sci & Elec Eng)
%%         Zeyad Abdelmonem (PG/T - Comp Sci & Elec Eng)
%% University of Surrey, United Kingdom
%% Email address: xl01339@surrey.ac.uk
%%                za00632@surrey.ac.uk
%% Time: 22/11/2024 17:44
%%       28/11/2024 15:55
close all;
clear;

% Load train data
trainDataFile = GlobalSetting.TRAIN_DATA;
% trainDataFile = [GlobalSetting.DATA_AUGMENTATION, '/data_augmentation_male_x1.mat'];
% trainDataFile = [GlobalSetting.DATA_AUGMENTATION, '/data_augmentation_male_x2.mat'];
% trainDataFile = [GlobalSetting.DATA_AUGMENTATION, '/data_augmentation_male_x3.mat'];
% trainDataFile = [GlobalSetting.DATA_AUGMENTATION, '/data_augmentation_male_x5.mat'];
% trainDataFile = [GlobalSetting.DATA_AUGMENTATION, '/data_augmentation_male_x10.mat'];
% trainDataFile = [GlobalSetting.DATA_AUGMENTATION, '/data_augmentation_male_x50.mat'];
% trainDataFile = [GlobalSetting.DATA_AUGMENTATION, '/data_augmentation_word_x2.mat'];
% trainDataFile = [GlobalSetting.DATA_AUGMENTATION, '/data_augmentation_word_x5.mat'];
% trainDataFile = [GlobalSetting.DATA_AUGMENTATION, '/data_augmentation_male_word_x5.mat'];
% trainDataFile = [GlobalSetting.DATA_AUGMENTATION, '/data_augmentation_male__x5_female.mat'];
% trainDataFile = [GlobalSetting.DATA_AUGMENTATION, '/data_augmentation_male__x5_word_x3.mat'];
trainData = load(trainDataFile, 'trainData').trainData;
K=size(trainData,2);

% Load test data
testDataFile = GlobalSetting.TEST_DATA;
testData = load(testDataFile, 'testData').testData;
K_test=size(testData,2);

% The state of HMM model
N = GlobalSetting.N;
% Parameter dimension
D = GlobalSetting.D;

WORDS = GlobalSetting.WORDS;
words_len = length(WORDS);

wordData = {trainData.word};

% Calculate MFCCs
```

```
disp('Calculating MFCCs...');
for k = 1:K
    if isfield(trainData(k),'data') & ~isempty(trainData(k).data)
        continue;
    else
        sampleData = trainData(k).sampleData;
        sampleRate = trainData(k).sampleRate;
        mfccCoeff = mfccFeature(sampleData, sampleRate, D);
        trainData(k).data = mfccCoeff;
    end
end

% Train HMM model
epochs = GlobalSetting.EPOCHS;
all_errors = zeros(epochs, 1);
hmm_models = struct();
lowest_error = 1;

for loop = 1:epochs
    fprintf('Starting Epoch %d ... \n', loop)
    % Run training with different group data based on different word.
    for i=1:words_len
        word = WORDS{i};
        fprintf('Start training %s group data\n', word);
        % Find the index of word
        idxes = find(strcmp(wordData, word));

        samples = trainData(idxes);
        % The number of pdf in each state
        PDFS = repmat(D, 1, N);
        if loop == 1
            hmm = init_hmm(samples, PDFS);
        else
            % update hmm model
            hmm = hmm_models.(word);
        end
        hmm_models.(word)=Baum_Welch(hmm, samples);
    end

    % Evaluate the model
    v_output(loop)=0;
    for k = 1:K_test
        data = testData(:,k);
        sampleData = data.sampleData;
        sampleRate = data.sampleRate;
        % Compute MFCC coefficients
        mfccCoeff = mfccFeature(sampleData, sampleRate, D);
        for j=1:words_len
            word = WORDS{j};
            hmm = hmm_models.(word);
            [viterbi_res, ~] = viterbi(hmm, mfccCoeff);
            pout(j) = viterbi_res;
            v_output(loop) = v_output(loop) + viterbi_res;
        end
        [d,n] = max(pout);
        word_res = WORDS{n};
```

```
        testData(k).prediction = word_res;
    end

    % Compute error rate
    trueSequence = {testData.word};
    predictedSequence = {testData.prediction};
    [errorRate] = compute_error_rate(trueSequence, predictedSequence);
    all_errors(loop) = errorRate;

    fprintf('%d Epoch, The recognition error rate is %f\n', loop, errorRate)

    % Update the best model if current error rate is lower than the best one
    if errorRate <= lowest_error
        lowest_error = errorRate;
        best_hmm = hmm_models;
        fprintf('New best model found: model of loop %d !\n', loop);
    end

    % Convergence monitoring
    % Compare the distance between two HMMs.
    if loop>1
        difference = abs((v_output(loop)-v_output(loop-1))/v_output(loop));
        fprintf('Difference of the viterbi output = %d\n', difference);
        if  difference < 2e-5
            fprintf('The model converges!\n');
            % break
        end
    end

    % Save the HMM model
    saveDir = GlobalSetting.HMM_MODEL;
    if ~exist(saveDir, 'dir')
        % Create the new directory
        mkdir(saveDir);
    end

    str_loop = num2str(loop);
    model_file = [saveDir, '/hmm_models_', str_loop, '.mat'];
    % model_file = strjoin(model_file, '');
    save(model_file, 'hmm_models');

end

% Plot the error rate
epoch_list = 1:epochs;
plot_errors(epoch_list, all_errors);
```

Listing 2: test_model.m

```
  %% Author: Xiaoguang Liang (PG/T - Comp Sci & Elec Eng)
%% University of Surrey, United Kingdom
%% Email address: xl01339@surrey.ac.uk
%% Time: 22/11/2024 23:05

clc;
close all;
clear;
```

```
9
10 D = 13;
11
12 % Load test data
13 % testDataFile = GlobalSetting.TEST_DATA;
14 % testData = load(testDataFile, 'testData').testData;
15
16 % Load evaluation data
17 evaluationDataFile = GlobalSetting.EVALUATION_DATA;
18 evaluationData = load(evaluationDataFile, 'evaluationData').evaluationData;
19
20 % Load HMM model
21 model_num = 11;
22 model_file = [GlobalSetting.HMM_MODEL, '/hmm_models_', num2str(model_num), '.mat'];
23 hmm_model = load(model_file, 'hmm_models').hmm_models;
24 WORDS = GlobalSetting.WORDS;
25 words_len = length(WORDS);
26
27 Ndata = size(evaluationData,2);
28 % Ndata = size(testData,2);
29
30 p = 0;
31 n = 0;
32 for i=1:Ndata
33     data = evaluationData(:,i);
34     sampleData = data.sampleData;
35     sampleRate = data.sampleRate;
36     true_word = data.word;
37     % Compute MFCC coefficients
38     mfccCoeff = mfccFeature(sampleData, sampleRate, D);
39
40     % Recognize the word
41     for j=1:words_len
42         word = WORDS{j};
43         hmm = hmm_model.(word);
44         pout(j) = viterbi(hmm, mfccCoeff);
45     end
46     [d,n] = max(pout);
47
48     word_res = WORDS{n};
49     evaluationData(i).prediction = word_res;
50     fprintf('The NO. %d word is recognized as %s\n', i-1,word_res)
51     if strcmp(word_res, true_word)
52         p = p+1;
53         fprintf('The recognition is correct\n')
54     else
55         n = n+1;
56         fprintf('The recognition is incorrect\n')
57     end
58
59 end
60
61 accuracy = p/Ndata;
62 fprintf('The recognition rate is %f\n', accuracy)
63
64 % Save the recognition result
```

```
file = GlobalSetting.RECOGNITION_RESULT;
save(file, 'evaluationData');
```

Listing 3: compute_confusion_matrix.m

```
  %% Author: Xiaoguang Liang (PG/T - Comp Sci & Elec Eng)
%% University of Surrey, United Kingdom
%% Email address: xl01339@surrey.ac.uk
%% Time: 03/12/2024 22:19

clc;
close all;
clear;

% Load recognition result
evaluationDataFile = GlobalSetting.RECOGNITION_RESULT;
evaluationData = load(evaluationDataFile, 'evaluationData').evaluationData;


true_word = {evaluationData.word};
prediction = {evaluationData.prediction};

% Plot the confusion matrix
fig = figure;
cm = confusionchart(true_word,prediction, 'RowSummary','row-normalized','
    ColumnSummary','column-normalized');
cm.Title = 'Confusion Matrix for Speech Recognition Result';

fig_Position = fig.Position;
fig_Position(3) = fig_Position(3)*1.5;
fig.Position = fig_Position;

% Save the graph as a high-resolution PNG file
saveDir = GlobalSetting.GRAPH_PATH;
if ~exist(saveDir, 'dir')
    % Create the new directory
    mkdir(saveDir);
end
savePath=[saveDir, '/confusion_matrix.png'];

% exportgraphics(fig, savePath, 'Resolution', 300);
% Save the figure as a PNG file with specified margins
print(gcf, savePath, '-dpng', '-r300');  % '-r300' sets 300 DPI for high resolution
```

Listing 4: GlobalSetting.m

```
  classdef GlobalSetting
    %GLOBALSETTING Summary of this class goes here
    %   Detailed explanation goes here

    properties(Constant=true)
        % Dataset path
        DATASET_FOLDER = 'data/EEEM030cw2-DevelopmentSet-2024';
        EVALUATION_FOLDER = 'data/EEEM030cw2-EvaluationSet-2024';
        TRAIN_DATA = 'data/trainData.mat';
        TEST_DATA = 'data/testData.mat';
        EVALUATION_DATA = 'data/evaluationData.mat';
```

```
        HMM_MODEL = 'data/hmm_models'
        RECOGNITION_RESULT = 'data/recognition_result.mat'
        GRAPH_PATH = 'data/graphs'
        DATA_AUGMENTATION = 'data/data_augmentation'

        % words for speech recognition
        WORDS = {'heed', 'hid', 'head', 'had', 'hard', 'hud', 'hod', 'hoard', 'hood
            ', 'whod', 'heard'}

        % Epochs for training
        EPOCHS = 15

        % HMM parameters
        % The state of HMM model
        N = 8;
        % The dimension of continuous probability density function
        D = 13;

        % Based on the instructions in the assignment, we can get initial A and Pi
            in Table 1
        A_init = [0.8 0.2 0.0 0.0 0.0 0.0 0.0 0.0,
            0.0 0.8 0.2 0.0 0.0 0.0 0.0 0.0,
            0.0 0.0 0.8 0.2 0.0 0.0 0.0 0.0,
            0.0 0.0 0.0 0.8 0.2 0.0 0.0 0.0,
            0.0 0.0 0.0 0.0 0.8 0.2 0.0 0.0,
            0.0 0.0 0.0 0.0 0.0 0.8 0.2 0.0,
            0.0 0.0 0.0 0.0 0.0 0.0 0.8 0.2,
            0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.8];
        Pi_init = [1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0];
        eta_init = [0 0 0 0 0 0 0 0.2]';

        % Replace NaNs with 0
        REPLACE_NAN = 0;

    end

    methods
        function obj = GlobalSetting()
            %GLOBALSETTING Construct an instance of this class
            %   Detailed explanation goes here
        end
    end
end
```

Listing 5: mfccFeature.m

```
  function mfccCoeff = mfccFeature(sampleData, sampleRate, Dimension)
% MFCCFEATURE Summary of this function goes here
%
% [OUTPUTARGS] = MFCCFEATURE(INPUTARGS) Explain usage here
%
% Examples:
%
% Provide sample usage code here
%
% See also: List related files here

```

```
% Author: Tofarati Onatunde, Xiaoguang Liang, University of Surrey
% Date: 2024/11/22 23:26:20
% Revision: 0.1


% PARAMETERS
frame_length_ms = 30;   % Frame length in ms
hop_size_ms = 10;       % Hop size in ms

% Frame size and hop size in samples
frame_length_samples = round(frame_length_ms / 1000 * sampleRate);  % Convert ms to samples
hop_size_samples = round(hop_size_ms / 1000 * sampleRate);        % Convert ms to samples

% Define a function for extracting MFCCs from a windowed frame
extract_mfcc_from_frame = @(audio, fs, frame_length, hop_size, num_coeffs) ...
    mfcc(audio, fs, 'Window', hamming(frame_length, 'periodic'), ...
    'OverlapLength', frame_length - hop_size, 'NumCoeffs', num_coeffs, 'LogEnergy', 'replace');

% Extract MFCCs for each audio file using frame-based processing
mfccCoeff = extract_mfcc_from_frame(sampleData, sampleRate, frame_length_samples, hop_size_samples, Dimension);

end
```

Listing 6: Develop_Set_Evaluation.mlx

```

  % Define the speaker numbers and words
  speakers = 1:30;
  words = {'heed', 'hid', 'head', 'had', 'hard', 'hud', 'hod', 'hoard', 'hood', 'whod', 'heard'};

  % Generate the audio file list using loops
  audio_files = cell(1, length(speakers) * length(words));  % Preallocate cell array
  index = 1;  % Initialize index for the audio_files array

  for speaker = speakers
      for word_idx = 1:length(words)
          % Create the filename for each combination of speaker and word
          audio_files{index} = sprintf('sp%02da_w%02d_%s.mp3', speaker, word_idx, words{word_idx});
          index = index + 1;  % Increment the index
      end
  end


  % Initialise variables to store MFCCs
  num_files = length(audio_files);
  mfccs = cell(num_files, 1);  % Cell array to hold the MFCCs for each audio file

  % PARAMETERS
  frame_length_ms = 30;   % Frame length in ms
  hop_size_ms = 10;       % Hop size in ms
```

```
num_mfcc_coeffs = 13;   % Number of MFCC coefficients (including zeroth)

% Assuming the sampling rate is the same for all files, we read the first file to get fs
[example_audio, fs_example] = audioread(audio_files{1});

% Frame size and hop size in samples
frame_length_samples = round(frame_length_ms / 1000 * fs_example);  % Convert ms to samples
hop_size_samples = round(hop_size_ms / 1000 * fs_example);        % Convert ms to samples

% Define a function for extracting MFCCs from a windowed frame
extract_mfcc_from_frame = @(audio, fs, frame_length, hop_size, num_coeffs) ...
    mfcc(audio, fs, 'Window', hamming(frame_length, 'periodic'), ...
         'OverlapLength', frame_length - hop_size, 'NumCoeffs', num_coeffs);

% Loop through the audio files and extract MFCCs for each one
for i = 1:num_files
    [audio_data, fs] = audioread(audio_files{i});  % Read the current audio file
    mfccs{i} = extract_mfcc_from_frame(audio_data, fs, frame_length_samples, hop_size_samples, num_mfcc_coeffs);
end

% Now 'mfccs' contains the MFCC features for each audio file

% Choose the file you want to display the MFCC for
file_index = 1;  % Change this to select a different file

% Extract the MFCCs for the selected file
mfcc_data = mfccs{file_index};

% PLOT MFCCs SPECTOGRAM-LIKE IMAGE
figure;
imagesc(mfcc_data');  % Transpose to make time on x-axis and MFCC coefficients on y-axis
colorbar;  % Display color bar to represent the magnitude of the MFCCs
xlabel('Frame index');
ylabel('MFCC coefficient index');
title(sprintf('MFCCs for file: %s', audio_files{file_index}));
axis tight;  % Adjust axis limits to fit the data (adjust axis for better readability)

%PLOT THE AUDIO WAVEFORM
figure;
plot((1:length(audio_data))/fs, audio_data);  % Time vs. amplitude plot
xlabel('Time (s)');
ylabel('Amplitude');
title(sprintf('Waveform of file: %s', audio_files{file_index}));

%PLOT THE SPECTOGRAM OF THE AUDIO SIGNAL
figure;
spectrogram(audio_data, hamming(frame_length_samples), frame_length_samples - hop_size_samples, 1024, fs, 'yaxis');
title(sprintf('Spectrogram of file: %s', audio_files{file_index}));
colorbar;
```

```
75    ylabel('Frequency (Hz)');
76    xlabel('Time (s)');
77
78    %%MFCC COFFICENT EVOLUTION OVER TIME
79    figure;
80    plot(1:size(mfcc_data, 2), mfcc_data(1, :));  % Plot the 1st MFCC coefficient
          over time
81    xlabel('Frame index');
82    ylabel('MFCC Coefficient Value');
83    title(sprintf('Evolution of 1st MFCC coefficient for file: %s', audio_files{
          file_index}));
84
85    %%COMPARISON OF MFFC FOR MULTIPLE FILES
86    figure;
87    subplot(2,1,1);  % First subplot for file 1
88    imagesc(mfccs{1}');
89    title('MFCCs for file 1');
90    colorbar;
91
92    subplot(2,1,2);  % Second subplot for file 2
93    imagesc(mfccs{2}');
94    title('MFCCs for file 2');
95    colorbar;
96
97    %HISTOGRAM OF MFCC COEFFICENTS
98    figure;
99    histogram(mfcc_data(1, :));  % Histogram of the 1st MFCC coefficient
100   xlabel('MFCC Value');
101   ylabel('Frequency');
102   title('Histogram of 1st MFCC coefficient');
103
104   %%3D PLOT OF MFCC FEATURES
105   figure;
106   mesh(1:size(mfcc_data, 2), 1:3, mfcc_data(1:3, :));  % Plot 3 first MFCC
          coefficients
107   xlabel('Frame index');
108   ylabel('MFCC coefficient index');
109   zlabel('MFCC value');
110   title('3D plot of the first three MFCC coefficients');
```

Listing 7: Gussian_mix.m

```
1     function [mix] = Gussian_mix(vector, M, nn)
2  % GUSSIAN_MIX Summary of this function goes here
3  %
4  % [OUTPUTARGS] = GUSSIAN_MIX(INPUTARGS) Explain usage here
5  %
6  % Examples:
7  %
8  % Provide sample usage code here
9  %
10 % See also: List related files here
11
12 % Author: Xiaoguang Liang, University of Surrey
13 % Date: 2024/12/03 19:56:54
14 % Revision: 0.1
15
```

```
[mean, var] = meanVariance(vector');
mean = mean';
var = var';

% Calculate the number of elements in each cluster and normalize them to obtain the
    weights for each PDF.
weight = zeros(M,1);
for j = 1:M
    weight(j) = sum(find(j==nn));
end
weight = weight/sum(weight);

% Save the results.
mix.M      = M;
mix.mean   = mean;
mix.var    = var;
mix.weight = weight;

end
```

Listing 8: init_hmm.m

```
  function [hmm] = init_hmm(samples, M)
% INIT_HMM Summary of this function goes here
%
% [OUTPUTARGS] = INIT_HMM(INPUTARGS) Explain usage here
%
% Examples:
%
% Provide sample usage code here
%
% See also: List related files here

% Author: Xiaoguang Liang, University of Surrey
% Date: 2024/12/03 00:08:58
% Revision: 0.1

% The number of samples
K = length(samples);
% The number of states
N = length(M);
hmm.N = N;
hmm.M = M;

% Initial state probabilities
hmm.init = GlobalSetting.Pi_init;

% Transition probabilities A
hmm.trans = GlobalSetting.A_init;

% Make N segements
for k = 1:K
    T = size(samples(k).data,1);
    samples(k).segment=floor([1:T/N:T T+1]);
end

% Obtain a continuous Gaussian mixture distribution.
```

```
for i = 1:N
    % Combine vectors with the same cluster and the same state into a single vector
        .
    vector = [];
    nn = [];
    for k = 1:K
        seg1 = samples(k).segment(i);
        seg2 = samples(k).segment(i+1)-1;
        segment_data = samples(k).data(seg1:seg2,:);
        tmp_nn = repmat(k,1,size(segment_data,1))';
        nn = [nn ; tmp_nn];
        vector = [vector ; segment_data];
    end
    mix(i) = Gussian_mix(vector, M(i), nn);
end

hmm.mix = mix;
```

Listing 9: meanVariance.m

```
  function [meanData, varianceData] = meanVariance(mfccCoeff)
% MEANVARIANCE Calculate the mean and variance of MFCC coefficients.
%
% [MEANDATA, VARIANCEDATA] = MEANVARIANCE(MFCCCOEFF) computes the mean and variance
% of the input MFCC coefficients matrix. The input matrix has dimensions D x M,
% where D is the number of MFCC coefficients and M is the number of observations.
% The function returns two matrices, MEANDATA and VARIANCEDATA, each of size D x D,
% containing the mean and variance values, respectively.
%
% Examples:
%   mfccCoeff = rand(13, 100); % Example MFCC coefficients matrix (13 coefficients,
    100 observations)
%   [meanData, varianceData] = meanVariance(mfccCoeff);
%   disp(meanData);
%   disp(varianceData);
%
% See also: mfcc, featureExtraction

% Author: Xiaoguang Liang, University of Surrey
% Date: 2024/11/22 23:28:41
% Revision: 0.1

% Get the dimensions of the input MFCC coefficients matrix
[D, M] = size(mfccCoeff);  % M : number of observations

% Initialize the mean and variance data matrices
meanData = zeros(D, D);
varianceData = zeros(D, D);

% Calculate the number of observations per segment
k = floor(M / D);

% Loop through each segment to calculate the mean and variance
for i = 1:D-1
    % Calculate the mean and variance for the current segment
    meanData(:, i) = mean(mfccCoeff(:, k * (i - 1) + 1:i * k), 2);
    varianceData(:, i) = var(mfccCoeff(:, k * (i - 1) + 1:i * k), 0, 2);
```

```
end

% Handle the last segment, which may have fewer observations
i = D;
meanData(:, i) = mean(mfccCoeff(:, k * (i - 1) + 1:end), 2);
varianceData(:, i) = var(mfccCoeff(:, k * (i - 1) + 1:end), 0, 2);

% Normalize the mean and variance
meanData = normalizeMatrix(meanData);
varianceData = normalizeMatrix(varianceData);

% Regulate the variance
% Regularization parameter
epsilon = 1e-8;
varianceData = regularizeCovariance(varianceData, epsilon);

end
```

Listing 10: normalizeMatrix.m

```
  function normalizedMatrix = normalizeMatrix(matrix, method)
% NORMALIZEMATRIX Summary of this function goes here
%
% [OUTPUTARGS] = NORMALIZEMATRIX(INPUTARGS) Explain usage here
%
% Examples:
%
% Provide sample usage code here
%
% See also: List related files here

% Author: Xiaoguang Liang, University of Surrey
% Date: 2024/12/05 19:01:05
% Revision: 0.1

% Default method if none is provided
if nargin < 2
    method = 'minmax';
end

switch lower(method)
    case 'minmax'
        % Min-max normalization: scale values to [0, 1]
        minVal = min(matrix(:));
        maxVal = max(matrix(:));
        if maxVal == minVal
            warning('Matrix has no variation. Returning zero matrix.');
            normalizedMatrix = zeros(size(matrix));
        else
            normalizedMatrix = (matrix - minVal) / (maxVal - minVal);
        end

    case 'zscore'
        % Z-score normalization: mean = 0, standard deviation = 1
        meanVal = mean(matrix(:));
        stdVal = std(matrix(:));
        if stdVal == 0
```

```
            warning('Matrix has no variation. Returning zero matrix.');
            normalizedMatrix = zeros(size(matrix));
        else
            normalizedMatrix = (matrix - meanVal) / stdVal;
        end

    otherwise
        error('Unsupported normalization method. Use ''minmax'' or ''zscore''.');
end
end
```

Listing 11: regularizeCovariance.m

```
  function regularizedCov = regularizeCovariance(covMatrix, epsilon)
% REGULARIZECOVARIANCE Summary of this function goes here
%
% [OUTPUTARGS] = REGULARIZECOVARIANCE(INPUTARGS) Explain usage here
%
% Examples:
%
% Provide sample usage code here
%
% See also: List related files here

% Author: Xiaoguang Liang, University of Surrey
% Date: 2024/12/05 19:02:12
% Revision: 0.1

% Validate inputs
if ~ismatrix(covMatrix) || size(covMatrix, 1) ~= size(covMatrix, 2)
    error('Input must be a square covariance matrix.');
end

if epsilon <= 0
    error('Regularization term epsilon must be a positive scalar.');
end

% Add epsilon to the diagonal elements
regularizedCov = covMatrix + epsilon * eye(size(covMatrix));

% Ensure symmetry (numerical stability)
regularizedCov = (regularizedCov + regularizedCov') / 2;
end
```

Listing 12: data_augmentation.m

```
  %% Author: Xiaoguang Liang (PG/T - Comp Sci & Elec Eng)
%% University of Surrey, United Kingdom
%% Email address: xl01339@surrey.ac.uk
%% Time: 04/12/2024 17:12

clc;
close all;
clear;

% Characteristics and Evaluation Results of Validation Data:
%
```

```
% Characteristics of Validation Data:
% The pronunciations in the validation dataset consist entirely of male voices across 11 groups of words.
%
% Evaluation Results of Validation Data:
%       •       The prediction accuracy for ""had and ""hard is 0%.
%       •       The prediction accuracy for ""heard and ""whod is 10%.
%       •       The prediction accuracy for ""hid and ""hoard is relatively high, at 90% and 100%, respectively.
%
% Enhancement Measures for the Data:
%       1.      Train only on male voices.
%       2.      Perform 2x or 3x repetition augmentation for male voice recordings.
%       3.      Perform 2x or 3x repetition augmentation for audio samples of the words "had", "hard", "heard", and "whod".

% Male voice prefixes
manPrefix = {'sp01a', 'sp03a', 'sp05a', 'sp07a', 'sp11a', 'sp13a', 'sp14a', 'sp15a', 'sp17a', 'sp22a', 'sp24a', 'sp26a', 'sp28a', 'sp30a'};
womanPrefix = {'sp02a', 'sp04a', 'sp06a', 'sp08a', 'sp09a', 'sp10a', 'sp12a', 'sp16a', 'sp18a', 'sp19a', 'sp20a', 'sp23a', 'sp25a', 'sp27a', 'sp29a'};
childPrefix = {'sp21a'};


% Set the train ratio
trainRatio = 1;

manPrefixLen = length(manPrefix);
numManTrain = round(trainRatio*manPrefixLen);
randManIndex = randperm(manPrefixLen);
trainManPrefix = manPrefix(randManIndex(1:numManTrain));

% womanPrefixLen = length(womanPrefix);
% numWomanTrain = round(trainRatio*womanPrefixLen);
% randWomanIndex = randperm(womanPrefixLen);
% trainWomanPrefix = womanPrefix(randWomanIndex(1:numWomanTrain));

% Get all the audio files
datasetFolder = GlobalSetting.DATASET_FOLDER;
allFiles = dir(fullfile([datasetFolder,'/*.mp3']));
filesLen = length(allFiles);

% Define training and testing data
trainData = struct('name', {}, 'path', {}, 'gender', {}, 'word', {}, 'sampleData', {}, 'sampleRate', {}');

% Specify the repetition factor
repetitionFactor = 5;
repetitionFactor_word = 3;

% Get all words
WORDS = GlobalSetting.WORDS;
words_len = length(WORDS);

% special_words = {'had', 'hard', 'whod', 'heard'};
special_words = {'had', 'head'};
```

```
62
63 % Loop through all files to categorize them into training and testing sets.
64 for i = 1:filesLen
65     fileName = allFiles(i).name;
66     fileNameSplited = split(fileName, "_");
67     filePrefix = fileNameSplited{1};
68     word = fileNameSplited{3};
69     wordSplited = split(word, ".");
70     word = wordSplited{1};
71     filePath = [datasetFolder '/' fileName];
72     [y,Fs]=audioread(filePath);
73
74     if ismember(filePrefix, trainManPrefix)
75         % Repeat the male data
76         for j = 1:repetitionFactor
77             trainData(end+1).name = fileName;
78             trainData(end).path = filePath;
79             trainData(end).gender = 'man';
80             trainData(end).word = word;
81             trainData(end).sampleData = y;
82             trainData(end).sampleRate = Fs;
83         end
84
85         % Repeat the special words
86         if ismember(word, special_words)
87             for j = 1:repetitionFactor_word
88                 trainData(end+1).name = fileName;
89                 trainData(end).path = filePath;
90                 trainData(end).gender = 'man';
91                 trainData(end).word = word;
92                 trainData(end).sampleData = y;
93                 trainData(end).sampleRate = Fs;
94             end
95         end
96     % elseif ismember(filePrefix, trainWomanPrefix)
97     %     trainData(end+1).name = fileName;
98     %     trainData(end).path = filePath;
99     %     trainData(end).gender = 'woman';
100    %     trainData(end).word = word;
101    %     trainData(end).sampleData = y;
102    %     trainData(end).sampleRate = Fs;
103    % else
104    %     trainData(end+1).name = fileName;
105    %     trainData(end).path = filePath;
106    %     trainData(end).gender = 'child';
107    %     trainData(end).word = word;
108    %     trainData(end).sampleData = y;
109    %     trainData(end).sampleRate = Fs;
110    end
111
112 end
113
114 % Save the training and testing data to files
115 saveDir = GlobalSetting.DATA_AUGMENTATION;
116 if ~exist(saveDir, 'dir')
117     % Create the new directory
```

```
    mkdir(saveDir);
end
savePath=[saveDir, '/data_augmentation_male__x5_word_x3.mat'];
save(savePath, 'trainData');
```

Listing 13: data_exploration.m

```
  %% Author: Xiaoguang Liang (PG/T - Comp Sci & Elec Eng)
%% University of Surrey, United Kingdom
%% Email address: xl01339@surrey.ac.uk
%% Time: 22/11/2024 16:24

clc;
close all;
clear;

% Set the train ratio
trainRatio = 0.8;
% trainRatio = 1;

% By observing the data, we can find that the dataset contains a total of
% 30 groups of data, with each group consisting of 11 pieces of data.
% Each group represents the pronunciations of 11 words by the same person.
% Among the 30 groups, the voice types include male voices, female voices,
% and 'childrens voices, with 14 groups being male voices, 15 groups
% female voices, and 1 group 'childrens voices.

% Based on the identical prefixes of each 'groups files, the following
% classifications can be determined.
manPrefix = {'sp01a', 'sp03a', 'sp05a', 'sp07a', 'sp11a', 'sp13a', 'sp14a', 'sp15a'
    , 'sp17a', 'sp22a', 'sp24a', 'sp26a', 'sp28a', 'sp30a'};
womanPrefix = {'sp02a', 'sp04a', 'sp06a', 'sp08a', 'sp09a', 'sp10a', 'sp12a', '
    sp16a', 'sp18a', 'sp19a', 'sp20a', 'sp23a', 'sp25a', 'sp27a', 'sp29a'};
childPrefix = {'sp21a'};


% Split the data into training and testing sets, based on different genders
% Because 'childrens voices only have one sample, just put it into train data
manPrefixLen = length(manPrefix);
numManTrain = round(trainRatio*manPrefixLen);
randManIndex = randperm(manPrefixLen);
trainManPrefix = manPrefix(randManIndex(1:numManTrain));

womanPrefixLen = length(womanPrefix);
numWomanTrain = round(trainRatio*womanPrefixLen);
randWomanIndex = randperm(womanPrefixLen);
trainWomanPrefix = womanPrefix(randWomanIndex(1:numWomanTrain));

% Split dataset based on prefix data
datasetFolder = GlobalSetting.DATASET_FOLDER;
allFiles = dir(fullfile([datasetFolder,'/*.mp3']));
filesLen = length(allFiles);

numTrainFiles = numManTrain+numWomanTrain+1;

% Define training and testing data
trainData = struct('name', {}, 'path', {}, 'gender', {}, 'word', {}, 'sampleData',
```

```
    {}, 'sampleRate', {}');
testData = struct('name', {}, 'path', {}, 'gender', {}, 'word', {}, 'sampleData',
    {}, 'sampleRate', {}');

% Loop through all files to categorize them into training and testing sets.
for i = 1:filesLen
    fileName = allFiles(i).name;
    fileNameSplited = split(fileName, "_");
    filePrefix = fileNameSplited{1};
    word = fileNameSplited{3};
    wordSplited = split(word, ".");
    word = wordSplited{1};
    filePath = [datasetFolder '/' fileName];
    [y,Fs]=audioread(filePath);
    if ismember(filePrefix, trainManPrefix)
        trainData(end+1).name = fileName;
        trainData(end).path = filePath;
        trainData(end).gender = 'man';
        trainData(end).word = word;
        trainData(end).sampleData = y;
        trainData(end).sampleRate = Fs;
    elseif ismember(filePrefix, trainWomanPrefix)
        trainData(end+1).name = fileName;
        trainData(end).path = filePath;
        trainData(end).gender = 'woman';
        trainData(end).word = word;
        trainData(end).sampleData = y;
        trainData(end).sampleRate = Fs;
    elseif ismember(filePrefix, childPrefix)
        trainData(end+1).name = fileName;
        trainData(end).path = filePath;
        trainData(end).gender = 'child';
        trainData(end).word = word;
        trainData(end).sampleData = y;
        trainData(end).sampleRate = Fs;
    else
        testData(end+1).name = fileName;
        testData(end).path = filePath;
        testData(end).gender = 'man';
        testData(end).word = word;
        testData(end).sampleData = y;
        testData(end).sampleRate = Fs;
    end
end


% Save the training and testing data to files
trainDataFile = GlobalSetting.TRAIN_DATA;
save(trainDataFile, 'trainData');
testDataFile = GlobalSetting.TEST_DATA;
save(testDataFile, 'testData');
```

Listing 14: make_train_evaluation_data.m

```
  %% Author: Xiaoguang Liang (PG/T - Comp Sci & Elec Eng)
%% University of Surrey, United Kingdom
%% Email address: xl01339@surrey.ac.uk
```

```
%% Time: 01/12/2024 23:28

clc;
close all;
clear;

% Define a function to make train and evaluation data
function [data] = make_data(datasetFolder)
% Define data structure
data = struct('name', {}, 'path', {}, 'word', {}, 'sampleData', {}, 'sampleRate',
    {}');
% Split dataset based on prefix data
allFiles = dir(fullfile([datasetFolder,'/*.mp3']));
filesLen = length(allFiles);
% Loop through all files to store them into data.
for i = 1:filesLen
    fileName = allFiles(i).name;
    fileNameSplited = split(fileName, "_");
    word = fileNameSplited{end};
    wordSplited = split(word, ".");
    word = wordSplited{1};
    filePath = [datasetFolder '/' fileName];
    [y,Fs]=audioread(filePath);

    % Store data into data
    data(end+1).name = fileName;
    data(end).path = filePath;
    data(end).word = word;
    data(end).sampleData = y;
    data(end).sampleRate = Fs;
end

end

% Get the train data and evaluation data
trainFolder = GlobalSetting.DATASET_FOLDER;
evaluationFolder = GlobalSetting.EVALUATION_FOLDER;

trainData = make_data(trainFolder);
evaluationData = make_data(evaluationFolder);

% Save the data to
trainDataFile = GlobalSetting.TRAIN_DATA;
save(trainDataFile, 'trainData');

evaluationFile = GlobalSetting.EVALUATION_DATA;
save(evaluationFile, 'evaluationData');
```