Shahjalal University of Science and Technology
Institute of Information and Communication Technology

Speaker Diarization with Speech Recognition in Bangla

Arnab Saha
Reg. No.: 2017831049
4^{th} year, 2^{nd} Semester

Jagonmoy Dey
Reg. No.: 2017831025
4^{th} year, 2^{nd} Semester

Software Engineering, Institute of Information and Communication Technology, SUST

Supervisor
Mohammad Shahidur Rahman, PhD
Professor
Department of Computer Science and Engineering

26^{th} February, 2023

Speaker Diarization with Speech Recognition in Bangla

A Thesis submitted to the Software Engineering, Institute of Information and Communication Technology, Shahjalal University of Science and Technology, in partial fulfillment of the requirements for the degree of Bachelor of Science in Software Engineering.

By

Arnab Saha
Reg. No.: 2017831049
4^{th} year, 2^{nd} Semester

Jagonmoy Dey
Reg. No.: 2017831025
4^{th} year, 2^{nd} Semester

Software Engineering, Institute of Information and Communication Technology, SUST

Supervisor

Mohammad Shahidur Rahman, PhD

Professor

Department of Computer Science and Engineering

26th February, 2023

Recommendation Letter from Thesis/Project Supervisor

The thesis/project entitled *Thesis/Project title* submitted by the students

1. Arnab Saha
2. Jagonmoy Dey

is under my supervision. I, hereby, agree that the thesis/project can be submitted for examination.

Signature of the Supervisor:

Name of the Supervisor: Mohammad Shahidur Rahman, PhD

Date: 26th February, 2023

Certificate of Acceptance of the Thesis/Project

The thesis/project entitled *Thesis/Project title* submitted by the students

1. Arnab Saha
2. Jagonmoy Dey

on 26th February, 2023, hereby, accepted as the partial fulfillment of the requirements for the award of their Bachelor Degrees.

Director,IICT M. Jahirul Islam, PhD., PEng. Professor Institute of Information and Communication Technology	Chairman, Exam. Committee M. Jahirul Islam, PhD., PEng. Professor Institute of Information and Communication Technology	Supervisor Mohammad Shahidur Rahman, PhD Professor Department of Computer Science and Engineering

Abstract

This thesis presents a study on Bangla speaker diarization with speech recognition. Speaker diarization is the process of identifying "who spoke when" in an audio recording, while speech recognition is the process of transcribing speech into text. The combination of these two techniques is particularly useful for applications such as meeting transcription, call center analysis, and media content analysis.

In this study, we used a bidirectional long short-term memory (BiLSTM) model for speaker diarization and Google Cloud Speech-to-Text API for Bangla speech recognition. We also developed a data processing pipeline that included audio segmentation, speaker change detection, and language model integration. The performance of the proposed system was evaluated using the diarization error rate (DER) metric.

Our experimental results demonstrated that the proposed system achieved promising results in terms of speaker diarization accuracy and speech recognition performance. We also analyzed the impact of various factors on the system's performance, such as audio quality, speaker variability, and vocabulary size.

In conclusion, this study provides insights into the development of Bangla speaker diarization systems and highlights the potential of combining speaker diarization with speech recognition for various applications. Future work can focus on further improving the system's accuracy and extending the system to handle other languages and audio types.

Keywords: Speaker Diarization, Bangla, Bi-LSTM, Speech Recognition

Acknowledgements

We express our gratitude to the Department of Software Engineering at the IICT, SUST for their generous support, which enabled us to carry out this research.

We would like to extend our appreciation to our supervisor, Professor Mohammad Shahidur Rahman, Ph.D., of the CSE department at SUST for his invaluable guidance and advice.

Additionally, we wish to acknowledge those who have shown their support and also helped us with our annotation especially MD Toufiqul Islam Fahim, Nishat Rahman and Sayeda Suraiya Alam.

Contents

List of Figures

Chapter 1

Introduction

Speech recognition and speaker diarization are two vital components in the field of natural language processing. The main objective of speech recognition is to convert human speech into a digital format that can be processed by a machine, while speaker diarization aims to identify individual speakers and group them accordingly. With the recent advancements in deep learning, both these tasks have become more accurate and efficient.

There are many uses for the two fundamental speech-processing tasks of speaker diarization and speech recognition. Speech recognition entails turning speech into text . whereas speaker diarization requires detecting and separating different speakers in an audio recording [2, 3].

Bangla is the seventh most widely spoken language in the world and is the official language of Bangladesh and the Indian state of West Bengal. The increasing demand for automatic speech recognition (ASR) and speaker diarization systems in the Bangla language has led to the development of several research projects in this field.

The integration of speaker diarization and speech recognition for Bangla language is particularly useful in various applications, such as call center analytics, meeting transcription, voice search, and voice assistants. It allows for better understanding and analysis of speech data, thereby enabling the development of more intelligent and interactive systems.

Speaker diarization becomes more challenging when dealing with Bangla, which has a highly inflected morphology, rich in pronunciation variations, and may lack standardized orthography. The language's unique characteristics make it more difficult to design a robust system for speaker diarization and automatic speech recognition. The research and development of such systems for Bangla language are still in their infancy.

1.1 Definition

The act of automatically recognizing and dividing speech from an audio recording into unique segments corresponding to each individual speaker is known as speaker diarization. Identifying "who spoke when" in an audio recording without any prior knowledge of the speakers is, in other words, the task at hand. Speaker diarization normally produces a chronology of speech segments, each of which is labeled with the name of the speaker who delivered it[2, 3].

Speech-to-text technique, commonly referred to as speech recognition, translates spoken language into written text. It entails analyzing and deciphering the audio signal of spoken words using algorithms and artificial intelligence, then producing a textual representation of the spoken words and sentences [4].

1.2 Motivation

Bengali is a widely spoken language in South Asia, with over 250 million speakers worldwide. However, there hasn't been a Bengali-language system that combines speaker diarization with speech recognition up until now, which presents a significant opportunity for further exploration.

Speaker diarization and speech recognition are difficult undertakings, especially in languages like Bengali with intricate phonetics and accents. Speaker diarization also enables us to recognize and examine the speech patterns of several participants in a discussion, which has a variety of possible applications in fields including natural language processing, sentiment analysis, and customer feedback analysis.

There have been various joint voice recognition and speaker diarization systems proposed[5–7], but none for the Bengali language.

Speaker diarization, the process of identifying different speakers in an audio recording, can be used in many real-world applications when combined with speech recognition. Here are a few examples:

- Call center analytics: Speaker diarization can be used in a call center to distinguish specific speakers during a conversation between a customer and a representative. To analyze the dialogue and provide insights into consumer sentiment, the effectiveness of the representative's communication, and other crucial metrics, this can be integrated with speech recognition.
- Broadcast media: In the post-production of radio or television programs, speaker diarization, and speech recognition can be used to automatically transcribe the spoken information and distinguish distinct speakers. This can enable more effective editing and indexing of the content while also saving the production team time and effort.
- Meeting transcription: Meeting transcripts and speaker identification can both be done using speaker diarization and speech recognition. This is particularly helpful at large gatherings or conferences when it could be challenging to tell who is speaking. The final transcription can be used as a reference for further action items, for taking notes, or for indexing.
- Law enforcement and forensics: The examination of audio evidence in criminal investigations can be done using speaker diarization and speech recognition. Investigators can acquire crucial information that can be utilized in court by identifying distinct speakers and typing up the spoken speech.

Recognizing the potential impact of speaker diarization combined with speech recognition in Bengali in our daily lives, as well as acknowledging the fact that this field remains largely unexplored, has motivated us to pursue research in this area.

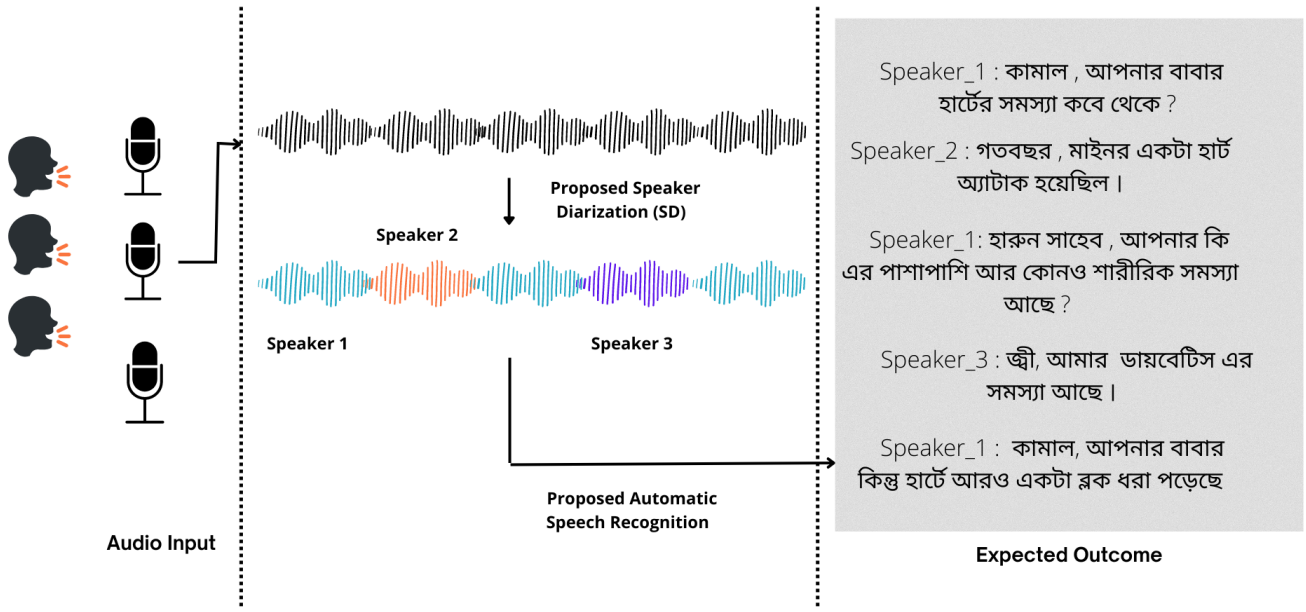


Figure 1.1: Expected Result

1.3 Research Question

To the best of our knowledge, there is currently no research that integrates speaker diarization with speech recognition for Bengali language. Therefore, in this thesis, we aim to investigate the following research questions:

- Can speaker diarization be effectively implemented for Bengali language with a significant diarization error rate (DER)?
- Is it feasible to combine speaker diarization with speech recognition to produce a transcript similar to the one shown in the figure above?

1.4 Objective

This thesis aims to explore the task of speaker diarization and automatic speech recognition for the Bangla language. In this thesis, we propose a novel approach using bi-directional Long

Short-Term Memory (BiLSTM) neural networks for speaker diarization, and the Google Cloud Speech-to-Text API for Bangla automatic speech recognition. We evaluate the proposed approach on a Bangla dataset of meetings, recorded under noisy conditions, with multiple speakers.

1.5 Organization of the Report

The remainder of this thesis is organized as follows:

- Chapter 2 provides background and related studies on speaker diarization and speech recognition systems.
- Chapter 3 presents the methodology and techniques used in the proposed system.
- Chapter 4 describes the data collection, preparation, and pre-processing techniques used for training and testing the system.
- Chapter 5 presents the experimental results and analysis of the proposed system.
- Chapter 6 concludes the thesis and proposes future directions for research.

Chapter 2

Background and Related Study

In speaker diarization, the audio recording is divided into smaller segments, such as speech turns or phrases, and each segment is then assigned to one of the speakers in the recording. There are numerous techniques for speaker diarization, including methods based on clustering, HMMs, neural networks, and other methods. In recent years, developments in machine learning and deep learning have significantly increased the accuracy of speaker diarization, making this a research topic of interest.

2.1 Speaker Diarization Approaches

Over time, speaker diarization techniques have changed dramatically as a result of advancements in machine learning and deep learning methods[4]. Early speaker diarization systems depended significantly on expert knowledge and manual system parameter tuning and were based on conventional clustering and HMM-based techniques.

To improve accuracy in speaker diarization tasks, current methods have emphasized the use of neural network-based models like LSTM and bi-LSTM models. End-to-end (E2E) speaker diarization models have additionally been developed, which may jointly learn the segmentation and clustering tasks without the need for additional processing steps. Overall, these developments have enhanced speaker diarization's accuracy and effectiveness, making it a crucial part of many

speech-related applications. Here is a brief description of some Deep Learning Based Speaker Diarization Approaches -

2.1.1 LSTM Based Speaker Diarization

2.1.1.1 i-vector and d-vector

Both i-vectors and d-vectors are utilized as feature extraction methods for speaker recognition and speaker diarization systems, aimed at extracting low-dimensional representations of speaker-specific information from speech signals. Despite their shared objective, these techniques differ in the manner by which they extract and represent such information.

I-vectors, which is a shortened form of identity vectors, are derived through the utilization of a Total Variability Modeling (TVM) factor analysis technique. This process entails creating a low-dimensional subspace model of speech feature variability across multiple speakers. Subsequently, the i-vector corresponding to a specific speaker is acquired by projecting their speech features onto this subspace.

In contrast to i-vectors, d-vectors - which are abbreviated for discriminative vectors - are generated through the implementation of a deep neural network that is specifically designed to extract speaker-specific information from speech signals. Typically, a substantial amount of speech data is employed to train the neural network, covering multiple speakers, and enabling the network to learn speaker-specific characteristics such as pitch, tone, and speaking style. The resulting output of the neural network for a particular speaker is then used as their corresponding d-vector.

To summarize, i-vectors are obtained through a factor analysis technique that models speaker variability within a low-dimensional subspace, while d-vectors are derived through a deep neural network that directly extracts speaker-specific information from speech signals. These techniques have both been demonstrated as effective in speaker recognition and speaker diarization systems, although the selection between them may depend on the specific application and the characteristics of the speech data under analysis.

2.1.1.2 RNN and LSTM

An RNN (Recurrent Neural Network) is an artificial neural network designed to process sequential data, such as speech or text. It operates by passing the current input and the previous hidden state to a set of neurons that produce the current output and a new hidden state. The hidden state serves as a memory that captures the sequence of previous inputs and is propagated to the next time step.

LSTM (Long Short-Term Memory) is a type of RNN that addresses the vanishing gradient problem, a common issue in conventional RNNs, which can lead to difficulties in training models with long sequences. This technique uses memory cells, allowing it to selectively retain or discard information over time. The LSTM architecture comprises three gates: the input gate, output gate, and forget gate, which regulate the flow of information into and out of the memory cells, thereby enabling the network to selectively store and retrieve information.

In conclusion, an RNN is a neural network that processes sequential data, and LSTM is a specific type of RNN that solves the vanishing gradient problem by incorporating memory cells and gating mechanisms.

2.1.1.3 Non-Parametric Clustering

Non-parametric clustering is an algorithmic technique used for clustering, which eliminates the requirement of predefining the number of clusters in the data. In other words, the algorithm automatically determines the number of clusters based on the input data, without any prior specification.

These clustering methods employ a similarity metric to measure the similarity between different data points, and then group them into clusters based on their similarity. The clustering can be performed hierarchically, where clusters are merged or split based on their similarity, or in a flat manner, where clusters are formed independently of each other.

2.1.1.4 Implementation

In order to create a novel d-vector based approach to speaker diarization, the authors of this approach [8] built on the success of d-vector based speaker verification[9] systems. They specifically coupled current non-parametric clustering work with LSTM-based d-vector [10] audio

embeddings to create a cutting-edge speaker diarization system. They found that, on average, d-vector based systems produce DER that is much lower than i-vector based systems. This is the first study to integrate spectral clustering and LSTM-based d-vector embeddings.

2.1.2 Bi-LSTM Based Speaker Diarization

2.1.2.1 What is Bi-LSTM ?

A Bi-LSTM is a type of neural network architecture used in natural language processing and other sequence modeling tasks. It combines both forward and backward information flow through a sequence of data. Unlike traditional LSTMs that compute hidden states only in the forward direction, Bi-LSTMs process the sequence in both forward and reverse directions, enabling the network to capture dependencies between the current input and both past and future inputs. This results in improved performance in tasks that require understanding of context and relationship between different parts of a sequence. Bi-LSTMs have double the number of parameters compared to traditional LSTMs, but they have been found effective in handling long and complex sequences.

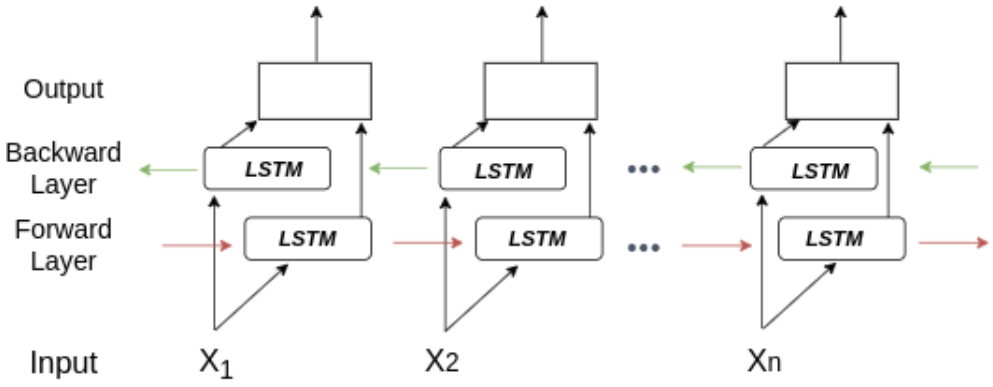


Figure 2.1: Bi-LSTM Architecture

2.1.2.2 Why Bi-LSTM is better than LSTM ?

The Bi-LSTM approach has a significant advantage over LSTM in handling overlapping speech during both training and inference. This feature makes the model well-suited for speaker diarization

tasks that involve real-recorded multi-speaker conversations. One of the main benefits of the Bi-LSTM approach is that it can be trained with multi-speaker segment labels, which makes the process much simpler and more efficient. On the other hand, the LSTM approach did not perform as well in handling speaker overlapping.

2.1.2.3 Implementation

In this research[11], a novel neural network-based speaker diarization model named EEND is proposed. This model can manage overlapping speech and can be immediately trained to reduce diarization errors. The model, which is characterized as multi-label classification[12], estimates joint speech activity of all speakers frame-by-frame using a recurrent neural network.
A permutation problem is when the output speaker labels are unclear in the training stage. The research introduces a permutation-free technique [12, 13] in the training objective function to address this problem. Using an objective function that minimizes diarization errors, the model is trained from beginning to end.

2.1.3 End-to-End Speaker Diarization with Self Attention

The End to End Speaker Diarization with Self Attention approach[14] , is a newer and more advanced approach. It also takes a sequence of acoustic features as input, but instead of using B-LSTM [15] cells, it uses self-attention mechanisms to encode the input sequence into a fixed-length representation.
Self-attention allows the model to focus on different parts of the input sequence and attend to relevant information, which can be particularly useful for speaker diarization tasks where different speakers may have different speaking styles or accents. The fixed-length representation is then fed into a feedforward network to predict speaker labels.

2.2 Our Approach for Speaker Diarization

Speaker diarization in the Bengali language has yet to be implemented. To tackle this, we adopted the Bi-LSTM approach for our model as it has several advantages[11].

- Firstly, it can handle overlapping speech efficiently by using it as input during both training and inference. Although we initially tried to exclude overlapped Bengali recordings during training and testing, the Bi-LSTM approach ensures that our model can still handle such scenarios.
- Secondly, the Bi-LSTM approach integrates the functionalities of speech activity detection, speaker identification, source separation, and clustering into a single neural network. This eliminates the need for separate modules for each task and simplifies the model architecture
- Lastly, the Bi-LSTM approach does not require clean, non-overlapping speech for training, making it possible to use domain adaptation with real overlapping speech conversations. We attempted to select recordings with minimal overlapped speech, but the Bi-LSTM approach provides us with the flexibility to train and test our model on a wide range of audio data.

2.3 Our Approach for Speech Recognition

Currently, we will employ Google's ASR for speech recognition purposes; however, our future plans involve utilizing our annotated data to train our own ASR model.

Chapter 3

Methodology

There are five primary sections to this thesis.

- Voice Activity Detection (webrtcvad)
- Based upon Bi-LSTM, speaker segmentation
- Extracting Embeddings (d-vector extraction)
- Clustering (k-MEANS and Mean Shift)
- Speech recognition

3.1 Voice Activity Detection (webrtcvad)

Using the WebRTC VAD library, we are conducting voice activity detection (VAD) on an audio file. The VAD is carried out by the code using the webrtcvad library, while audio file processing is handled using the librosa library.

WebRTC (Web Real-Time Communication) is an open-source technology that allows real-time communication between web browsers, mobile applications, and other platforms, using peer-to-peer (P2P) connections. WebRTC provides a set of APIs and protocols for web developers to build voice and video chat applications, file transfer, screen sharing, and other real-time communication applications.

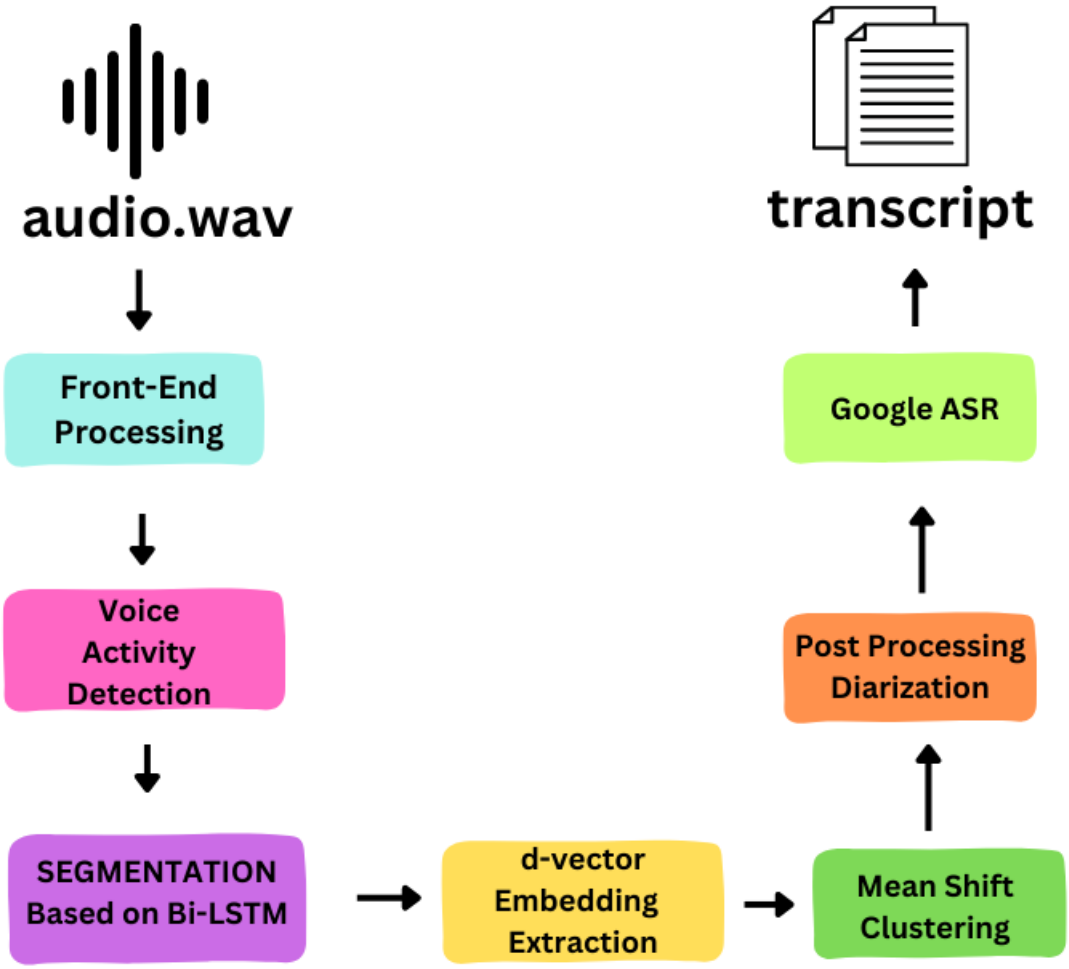


Figure 3.1: Full Pipeline

One of the main features of WebRTC is the ability to perform voice activity detection (VAD) in real-time. VAD is a process that separates speech segments from non-speech segments in an audio signal. This is an essential step in many speech processing applications, such as speech recognition, speaker identification, and transcription.

The WebRTC VAD library is a C++ library that provides an implementation of VAD for WebRTC. It works by analyzing the energy level and spectral characteristics of the audio signal to determine whether it contains speech or not. The library uses a set of heuristics and machine

learning techniques to make the VAD decision, and it can adapt to different environments and noise conditions.

The webrtcvad Python library provides a wrapper for the WebRTC VAD library, making it easy to use in Python applications. The library provides a Vad class that can be used to create a VAD object, and a is-speech method that takes an audio frame and a sample rate and returns a boolean value indicating whether the frame contains speech or not.
WebRTC VAD (Voice Activity Detection) uses a statistical model to detect voice activity in an audio signal. The specific model used is a Gaussian Mixture Model (GMM) based on Mel Frequency Cepstral Coefficients (MFCCs).
In speech processing, MFCCs are frequently used to extract information from an audio stream that are representative of the human auditory system. They are calculated by taking the Discrete Fourier Transform (DFT) of the signal and then applying a series of filters that mimic the human ear's frequency response. The resulting coefficients are then processed to reduce dimensionality and remove redundancy.
Once the MFCCs have been calculated, they are used to train a GMM to model the distribution of speech and non-speech frames in the training data. During detection, the GMM is used to calculate the likelihood that a given frame is speech or non-speech, and a threshold is applied to determine the final classification.
Mel-Frequency Cepstral Coefficients (MFCCs) are a commonly used feature extraction technique in speech recognition and audio analysis. The basic steps for computing MFCCs are as follows: 3.2)

- Pre-emphasis: The first step is to apply a high-pass filter to the audio signal to amplify high-frequency components and reduce the effect of low-frequency noise.
- Framing: The signal is divided into short overlapping frames, typically 20-40 ms long. The frames are often tapered using a window function to reduce spectral leakage.
- Fourier transform: A Fourier transform is applied to each frame to convert it from the time domain to the frequency domain. This results in a spectrum of frequency components for each frame.

- Mel-scale filterbank: A set of triangular filters is applied to the power spectrum to emphasize the frequency bands that are most important for human perception. The filters are spaced according to the Mel scale, which is a non-linear scale that maps frequency to pitch perception.

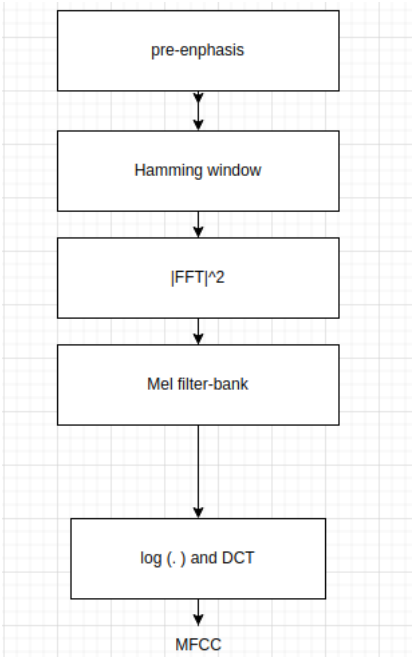


Figure 3.2: MFCC ([1])

- Logarithmic compression: The logarithm of the filterbank energies is taken to compress the dynamic range of the spectrum and emphasize the lower-frequency components.
- Discrete cosine transform (DCT): The resulting filterbank energies are transformed using a DCT to extract a set of cepstral coefficients that represent the spectral envelope of the signal. The DCT is a mathematical tool that converts a set of data points into a set of coefficients that capture the underlying pattern in the data.
- Cepstral mean normalization: The mean and variance of the cepstral coefficients are computed over all frames and used to normalize each frame's coefficients.

The resulting set of MFCCs represents a compact, low-dimensional representation of the spectral content of the signal. [16]

3.2 Based upon Bi-LSTM, speaker segmentation

The main objective of the design is to extract the lines separating the speech turns of various speakers from an audio recording. To do this, the architecture uses a binary sequence labeling task, where the output is a sequence of binary labels indicating whether a speaker change occurs at each time step.

The architecture begins by extracting MFCC features from the audio recording on a short, overlapping sliding window. These features are used as input to a recurrent neural network (RNN) that is trained to predict the binary label sequence. Then a binary sequence labeling task is created from the speaker change detection task. by defining

$$p = (p_1, p_2, \ldots, p_n) \in 0, 1^T \text{ such that } p_j = \begin{cases} 1, & \text{If a speaker is substituted during the } j\text{-th frame ,} \\ 0, & \text{if not} \end{cases}$$

The next step is to identify a function.

$$f : q \rightarrow p$$

that compares a label sequence and a feature sequence. We suggest utilizing a recurrent neural network trained with the binary cross-entropy loss function to simulate this function f.

The RNN architecture consists of two bidirectional LSTMs (Bi-LSTM i and ii) and a shared MLP (multi-layer perceptron). The Bi-LSTMs they use both past and future contexts by processing the input sequence both forward and backward. The MLP, which has three fully connected layers with tanh activation functions for the first two layers and a sigmoid activation function for the third layer, receives the concatenated outputs of the forward and backward LSTMs.

During training, the network is optimized using the Adam optimizer and the binary cross-entropy loss function. The binary cross-entropy loss function penalizes the model when it predicts a different binary label than the ground truth label. This encourages the model to learn to accurately predict the boundaries between speakers.

Once the model is trained, it can be used to predict the binary label sequence for new audio recordings. The boundaries between speakers can then be identified by finding the transitions between consecutive binary labels with different values.

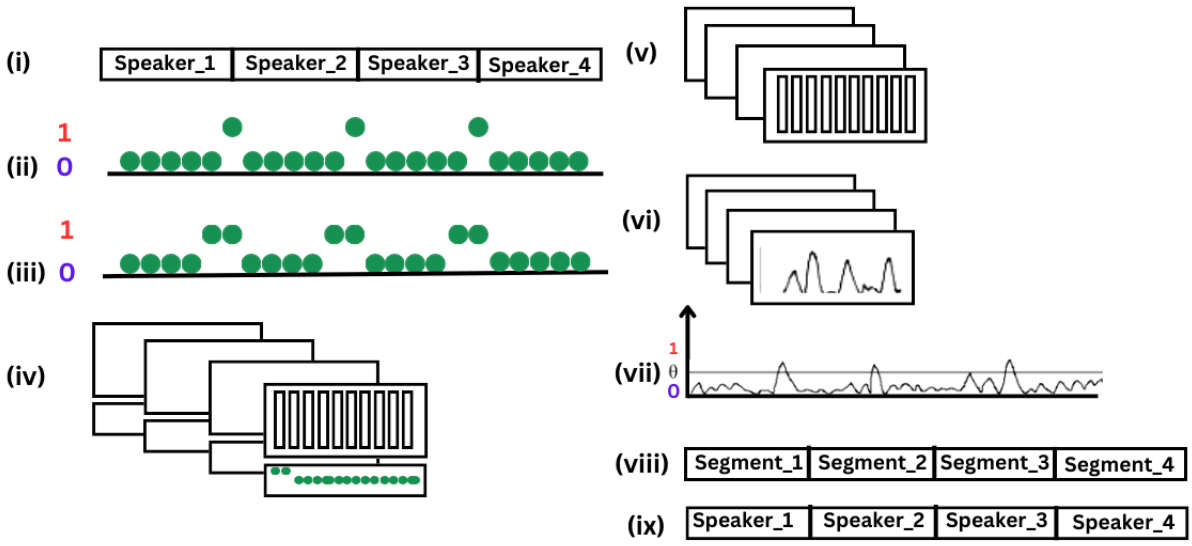


Figure 3.3: Speaker Segmentation

Overall, the architecture effectively turns the speaker segmentation task into a binary sequence labeling task, using an RNN with bidirectional LSTMs and a shared MLP to make predictions. This approach has been shown to be effective at accurately identifying the boundaries between speech turns of different speakers in audio recordings.

3.3 Mixture of VAD and Speaker Segmentation

The code snippet provided consists of three functions for performing VAD and speaker segmentation: groupIntervals, spliting, and finalReseg.

The groupIntervals function takes a list of intervals as input and returns a new list of merged intervals. It does so by iterating over the intervals and checking if they can be merged with the previous interval. If they can, it merges them. If they cannot, it appends the previous interval to the answer list and starts a new interval with the current one.

The spliting function takes the merged intervals and the original VAD output as input and returns a new list of intervals that accounts for speaker changes. It does so by iterating over the merged intervals and checking if the speaker changes within each interval. If it does, it inserts the timestamp of the speaker change into the interval.

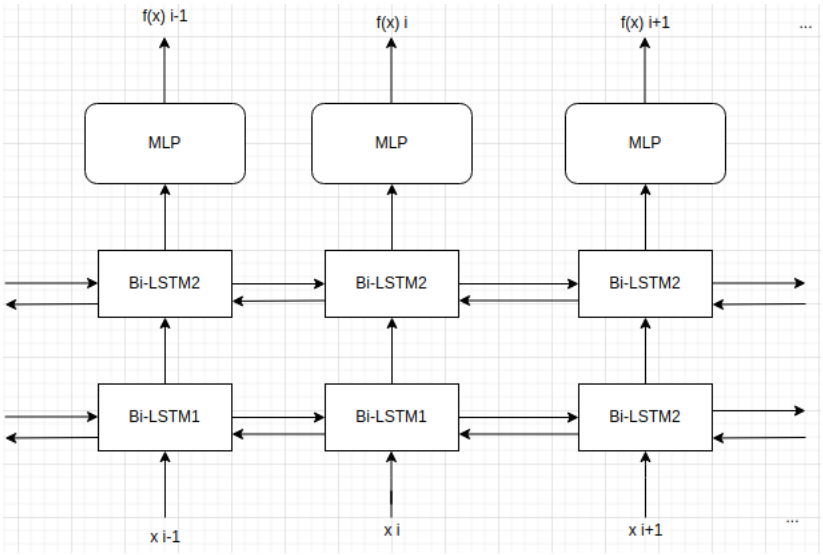


Figure 3.4: Model Architecture

The finalReseg function takes the output of spliting and returns a final list of intervals. It does so by iterating over each interval and checking if it has been split by a speaker change. If it has, it creates new intervals between each speaker change and appends them to the output list. If it has not been split, it appends the original interval to the output list.

Overall, these functions combine VAD and speaker segmentation by first using VAD to detect speech segments in an audio recording. The groupIntervals function then merges these segments into longer intervals that may include multiple speakers. The spliting function then uses speaker segmentation to split these longer intervals into shorter intervals that are specific to each speaker. Finally, the finalReseg function creates a final set of intervals that accounts for speaker changes within each interval. This allows for the accurate identification of each speaker's speech turn in the recording.

```
function groupIntervals(a)
    ans = []
    current = None

    for x in a:
        if current == None:
            current = x
```

```
            else:
                if x[0] - current[1] < 1:
                    current[1] = x[1]
                else:
                    ans.append(current)
                    current = x

        if current is not None:
            ans.append(current)

        d1 = convert ans to numpy array
        d2 = get unique values in d1
        d3 = reshape d2 to 2D array of shape (len(d2) / 2, 2)

        return d3

    function spliting(seg, arr)
        arr1 = copy arr to list
        temp = copy arr

        for i in range(len(seg) - 1):
            temp1 = convert seg[i] to float

            for j in range(len(arr)):
                if temp1 > arr[j][0] and temp1 < arr[j][1]:
                    arr1[j].insert(-1, temp1)

        for i in range(len(arr1)):
            size = len(arr1[i])
```

```
        if size >= 3:
            if arr1[i][-1] - arr1[i][-2] < 0.2:
                arr1[i].pop(-2)

    return arr1

function finalReseg(arr)
    z = []

    for i in arr:
        if len(i) == 2:
            z.append(i)
        else:
            t = len(i)

            for j in range(t - 1):
                if j != t - 1:
                    t1 = [i[j], i[j+1] - 0.01]
                    z.append(1)
                elif j == t - 1:
                    t1 = [i[j], i[j+1]]
                    z.append(temp1)

    return convert z to numpy array
```

3.4 Extracting Embeddings

We used pyannote/embedding for extraction of the the embedding[17].Pyannote is a Python package that provides tools for analyzing and processing audio data, including speaker diarization, speech activity detection, and speaker embedding extraction. Speaker embedding is a way of rep-

resenting a speaker's voice as a low-dimensional vector, which can be used for various downstream tasks, such as speaker verification or speaker classification.

Pyannote provides a range of speaker embedding models, including some pre-trained models, that can be used for speaker embedding extraction. Some of the models used in Pyannote include x-vector and time delay neural network (TDNN) models. These models are typically trained on large speech datasets, such as the VoxCeleb dataset, to learn to extract speaker embeddings that capture the unique characteristics of each speaker's voice.

In addition to pre-trained models, Pyannote also provides tools for training your own speaker embedding models, using your own data. This allows you to fine-tune a pre-trained model on your specific dataset, or to train a model from scratch if you have a large enough dataset.

Once a speaker embedding model is trained or loaded, Pyannote provides tools for extracting speaker embeddings from audio files. These embeddings can then be used for a variety of tasks, such as speaker diarization (i.e., clustering speech segments by speaker), speaker verification (i.e., determining if two speech segments are from the same speaker), or speaker classification (i.e., classifying a speech segment based on the speaker identity).

This model is built on the standard x-vector TDNN architecture, but instead of filter banks, it uses trainable SincNet features.[18]

3.5 Clustering

Clustering is a method of grouping similar items or data points together based on their similarity or distance. It is a common technique used in many areas of machine learning, including speaker diarization, which is the process of separating an audio recording into segments according to who is speaking.

In speaker diarization, clustering is used to group the speech segments into clusters based on the similarity of their acoustic features. Different types of clustering algorithms are used for speaker diarization, and each has its own strengths and weaknesses.

K-Means and Mean Shift are both clustering algorithms commonly used in machine learning and data analysis.

3.5.1 K-Means Algorithm

An iterative technique called K-Means that attempts to partition a given dataset into a set of K clusters, where K is a predefined number. The algorithm starts by randomly assigning each point to one of the K clusters, and then iteratively updates the cluster centroids to minimize the distance between each point and its assigned centroid. This process is repeated until the centroids no longer change significantly, or a maximum number of iterations is reached.

The basic steps of the K-Means algorithm are as follows:

- Randomly select K initial centroids from the dataset.
- Decide which centroid is the closest for each data point.
- Up till the centroids don't change, keep iterating. i.e., the clustering of data points remains constant.
- Calculate the sum of the squared distances between all of the centroids and the data points.
- Calculate each cluster's centroid again using the mean of all the data points that were allocated to it.
- Till the centroids stop fluctuating significantly or a predetermined number of iterations has been reached, repeat steps 2 and 3 as necessary.

The K-Means algorithm is computationally efficient and can handle large datasets. However, it has some limitations. One of the main drawbacks is that the algorithm is sensitive to the initial choice of centroids, which can result in different cluster assignments and centroids. Also, it requires a predefined number of clusters, which may not always be known.

3.5.2 Mean Shift Algorithm

A non-parametric clustering approach called Mean Shift is used to identify a density function's mode or peak.. In this algorithm, the mean of a set of points is shifted iteratively towards the highest density of data points until it converges to a local maximum. The points are then assigned to the nearest peak.

The basic steps of the Mean Shift algorithm are as follows:

- Initialize a kernel bandwidth value.
- Randomly select a data point as the starting point.
- Compute the mean of all the data points within the kernel bandwidth of the starting point.
- Shift the center of the kernel to the new mean.
- Repeat steps 3 and 4 until convergence.
- Find the closest peak to each data point.

The Mean Shift algorithm is computationally intensive and works well with small to medium-sized datasets. It is also able to detect the number of clusters automatically, which is a significant advantage over K-Means. However, the performance of the algorithm depends on the choice of the kernel bandwidth value, which can be challenging to tune.

For this case we used mean shift algorithm cause we don't know the number of speakers in the given audio.

3.6 Speech Recognition

We have used the Google ASR (Automatic Speech Recognition) system for speech recognition. For future works, we plan to develop a Bangla speech recognition model.

Google ASR works by taking an audio input and breaking it down into small units of audio called "frames." These frames are then analyzed to extract features such as frequency, amplitude, and duration. The ASR system then uses these features to recognize and transcribe the speech.

For Bangla speech recognition, Google uses deep learning models trained on large amounts of speech data. The specific model used for Bangla speech recognition by Google is not publicly disclosed. However, it is likely that the model uses a combination of neural network architectures, such as Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), to extract

and process features from the audio data and then perform the transcription.

Chapter 4

Data processing

There are several large datasets available for speaker diarization research, some of which are listed below:

- DIHARD II: This is a dataset for speaker diarization and recognition in meeting scenarios. It consists of over 200 hours of audio data from meetings, with simultaneous recording of audio and video. The data includes multiple microphones and speaker annotations, and the dataset has been designed to test the performance of diarization systems in real-world settings.
- CALLHOME: This is a dataset for speaker diarization and recognition in telephone conversations. It consists of over 120 hours of audio data, with conversations between English speakers and speakers of other languages. The data is annotated with speaker labels, and the dataset has been widely used for benchmarking diarization systems.
- VoxCeleb: This is a dataset for speaker recognition, which can also be used for diarization. It consists of over 1 million short audio segments, each containing a single spoken sentence from a celebrity. The data is diverse in terms of gender, age, and accent, and has been used to train and evaluate deep neural network-based diarization systems.
- NIST SRE: The NIST Speaker Recognition Evaluation (SRE) is a series of evaluations for speaker recognition and diarization systems. The evaluations use large datasets of telephone conversations, meetings, and broadcast news data, with annotations for speaker identity

and activity. The evaluations have been conducted periodically since 1996, and have been instrumental in advancing the state of the art in speaker recognition and diarization.

- AMI: The Augmented Multi-Party Interaction (AMI) corpus is a dataset for multi-modal interaction analysis, which includes speech, video, and physiological data. The dataset consists of over 100 hours of audio data from meetings, with annotations for speaker identity and turn-taking. The dataset has been used for research in diarization, speech recognition, and other related fields.

These datasets provide a wide range of challenges and scenarios for diarization systems, and have been instrumental in advancing the state of the art in speaker diarization research.We studied this datasets to understand how to collect and annotate audio data for training our models.

4.1 Data Collection

We gathered 8 hours' worth of audio data from Bangla YouTube videos for this investigation. Using the Y2Mate, the audio files were taken from the videos and converted from stereo to mono using the Audacity program.

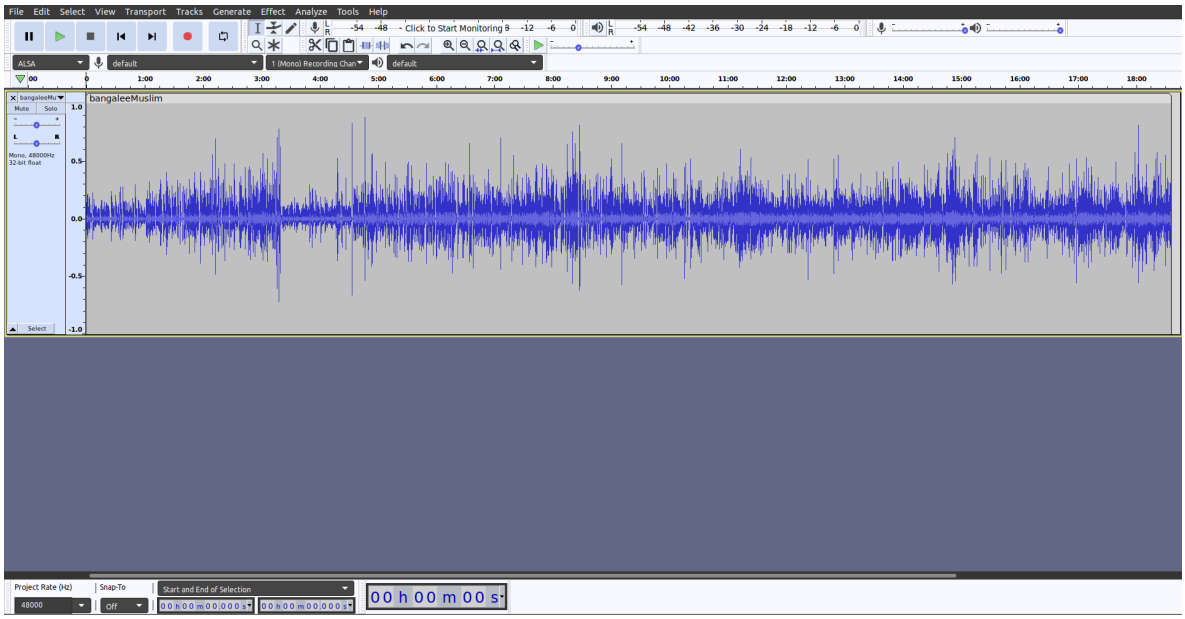


Figure 4.1: Audacity

Audacity was used to split the files, and the resultant split files were then exported as.wav files with sampling rates of 48000Hz and 16-bit PCM encoding.

4.2 Annotation

The audio data were annotated using the ELAN software.In Elan, we had to create two tiers: sentences and speakerId. In the sentence tier, we kept the start time and the end time of each sentence. We intend to use these sentences in future projects, such as developing our own ASR model for Bangla.

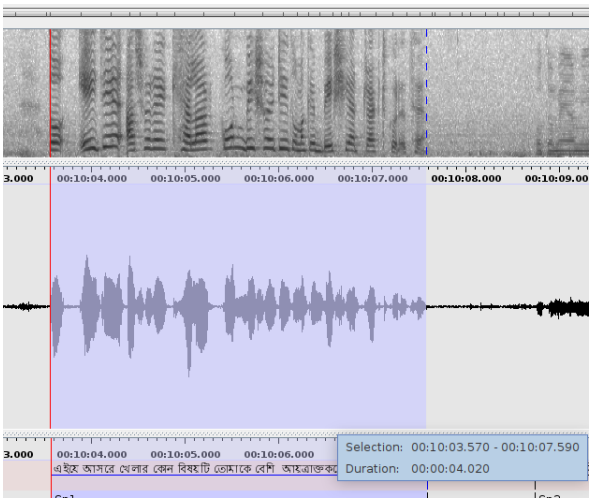


Figure 4.2: Elan Software for annotation

For speaker ID, we have considered two processes. The first process involves creating tiers for each speaker in ELAN. We would segment the audio or video file into smaller units, such as sentences or paragraphs, and then assign each unit to the appropriate speaker tier.

Once we have created the speaker tiers, we can use ELAN's search and filter functions to identify and analyze the speech of each speaker separately. This can be useful in various research contexts, such as discourse analysis, sociolinguistics, or phonetics.

However, we should note that creating speaker tiers manually can be a time-consuming and error-prone process, especially if the audio file contains multiple speakers or overlapping speech.

Figure 4.3: First Attempt

But as we are not making this model to deal with overlap speech, we changed our annotation process to speed up the annotation and also to get the whole range of the speaker's voice, not just the segmented sentence parts.

After considering different options for speaker identification, we decided to keep all the speakers in one tier called "speakerId" and store the start and end times for each speaker segment. This approach allowed us to speed up the annotation process and make it more efficient.

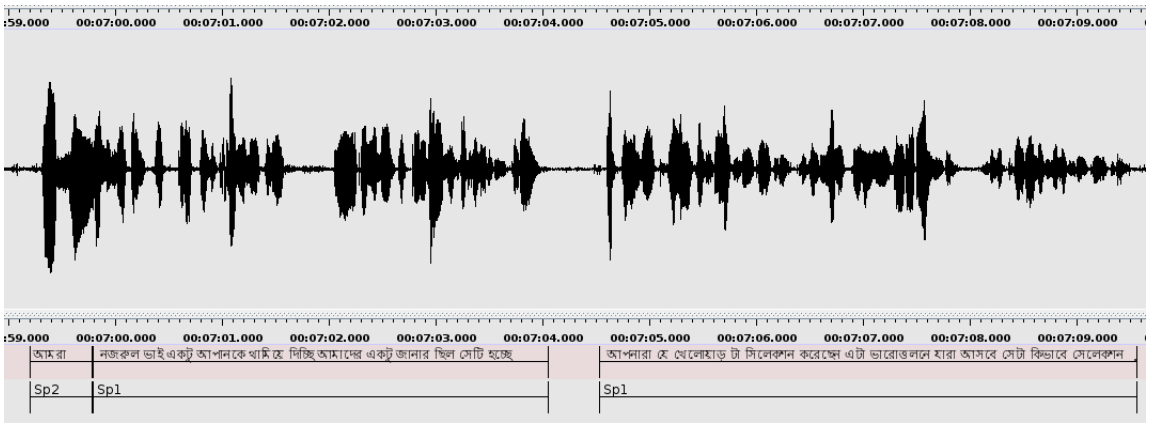


Figure 4.4: Second Attempt

Then we converted our eaf file to a CSV file using filename, Offset, Duration and SpeakerId as headers. We used the following script.

```
# import the required libraries
import pympi
import csv

# specify the path to the ELAN file
elan_file_path = "/path/to/elan/file.eaf"
filename = "name_of_output_file"
header = ["filename", "Offset", "Duration", "Speaker_id"]

# load the ELAN file
elan_file = pympi.Elan.Eaf(elan_file_path)

# open a CSV file to write
with open("/path/to/output/file.csv", "w", encoding="UTF8") as f:
    writer = csv.writer(f)

    # write the header row
    writer.writerow(header)

    # iterate over the speaker_id tier
    for tier_name in elan_file.get_tier_names():
        if tier_name != "Speaker_id":
            continue

        # extract the speaker segments from the tier
        tier_data = elan_file.get_annotation_data_for_tier(tier_name)
        for segment in tier_data:
```

```
        # write each segment to a new row in the CSV file
        writer.writerow([
            filename,
            segment[0] / 1000,  # convert start time to seconds
             # convert duration to seconds
            (segment[1] - segment[0]) / 1000,
            segment[2]  # speaker ID
        ])
```

4.3 Data Splitting

The audio data was divided into two sets: a training set and a test set. The test set and training set had completely different speakers to ensure unbiased performance evaluation of our model.

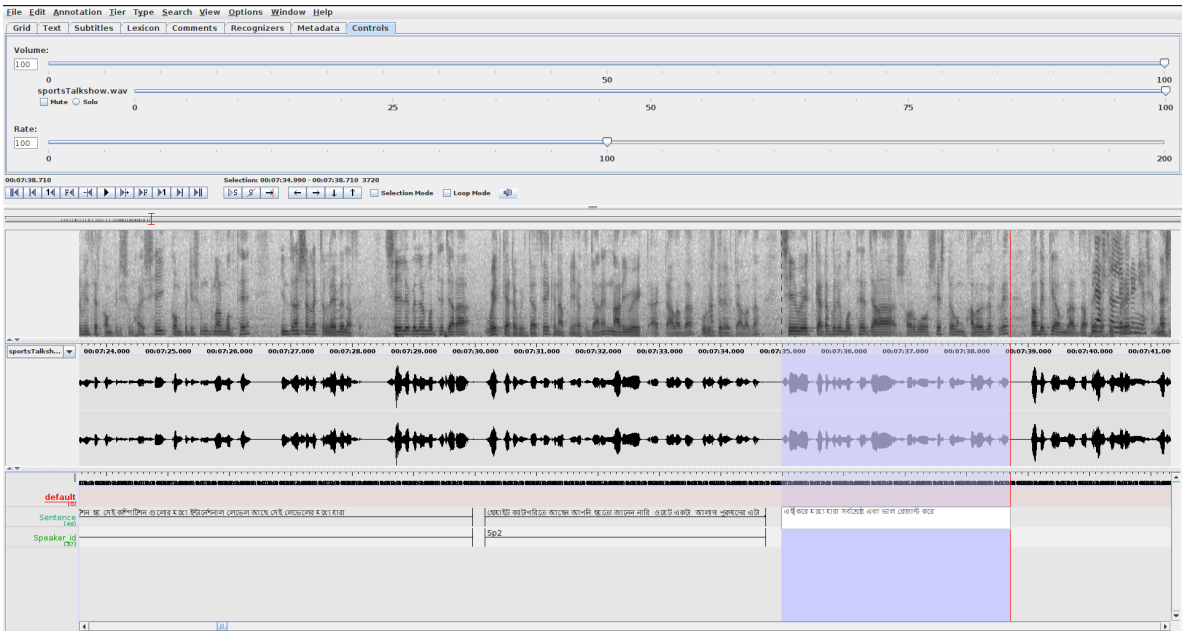


Figure 4.5: Elan Software for annotation

4.4 Data Preprocessing

Before training our model, we performed several preprocessing steps on the audio data. We converted the audio data into a feature representation that is suitable for training our model. We used the Mel-frequency cepstral coefficients (MFCCs) as the feature representation. Additionally, we applied normalization to the data to ensure that all features had the same mean and standard deviation.

This chapter has described the dataset collection and preprocessing procedures for developing a Bangla speech recognition system. The collected data were annotated using the ELAN software, and the audio files were split into training and test sets. We also performed data preprocessing steps to convert the audio data into a feature representation that is suitable for training our model. These procedures were essential in ensuring that our model was trained on high-quality data and was capable of accurately recognizing Bangla speech.

Chapter 5

Results and Analysis

5.1 Analysis

The model we are using is a sequential model in Keras with four layers, where the first two layers are bidirectional layers. Bidirectional layers process the input data in both forward and backward directions and combine the output from both directions, which is helpful in capturing the context of the data. The model has a total of 571,489 parameters, all of which are trainable.

The first layer is a bidirectional layer that has 256 units. The number of units determines the number of neurons in the layer, and thus the complexity of the layer. The output shape of the layer is (None, 137, 256), which means that the layer takes input with an unknown batch size (None) and processes it to output a 3D tensor of shape (137, 256), where 137 is the time dimension and 256 is the number of units in the layer. The second layer is also a bidirectional layer with 256 units, and it takes the output of the first layer as input. The output shape of this layer is also (None, 137, 256).

The third layer is a time distributed layer with 32 units. A time distributed layer applies the same layer to every time step of the input sequence independently. In this layer, the input is transformed from a tensor of shape (None, 137, 256) to a tensor of shape (None, 137, 32), where 32 is the number of units in the layer.

The fourth layer is also a time distributed layer, but it has only one unit. This layer takes the output of the previous layer as input and generates a tensor of shape (None, 137, 1).

Overall, this model takes a sequence of input data, processes it through two bidirectional layers,

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 bidirectional (Bidirectiona  (None, 137, 256)         167936
 l)

 bidirectional_1 (Bidirectio  (None, 137, 256)         394240
 nal)

 time_distributed (TimeDistr  (None, 137, 32)          8224
 ibuted)

 time_distributed_1 (TimeDis  (None, 137, 32)          1056
 tributed)

 time_distributed_2 (TimeDis  (None, 137, 1)           33
 tributed)

=================================================================
Total params: 571,489
Trainable params: 571,489
Non-trainable params: 0
```

Figure 5.1: Model

and then applies two time distributed layers to generate an output sequence. The output sequence has the same length as the input sequence, but each time step in the output is a single scalar value, which is the model's prediction for that time step.

Speaker change points are the core part of this speaker diarization process.The following visuals shows us the speaker change points one of training set audios.

Embedding and clustering are the most important parts of our model.

```
FUNCTION embeddings_(audio_path, resegmented, range):
    model = Load the pre-trained Pyannote audio model
    inference = Set up the inference model for Pyannote audio model
    y, sr = Load audio file and get the sample rate
    myDict = Create a dictionary to store audio information
    myDict['audio'] = audio_path
    myDict['duration'] = Calculate the duration of the audio file
    data = Create an empty list to store embeddings
```

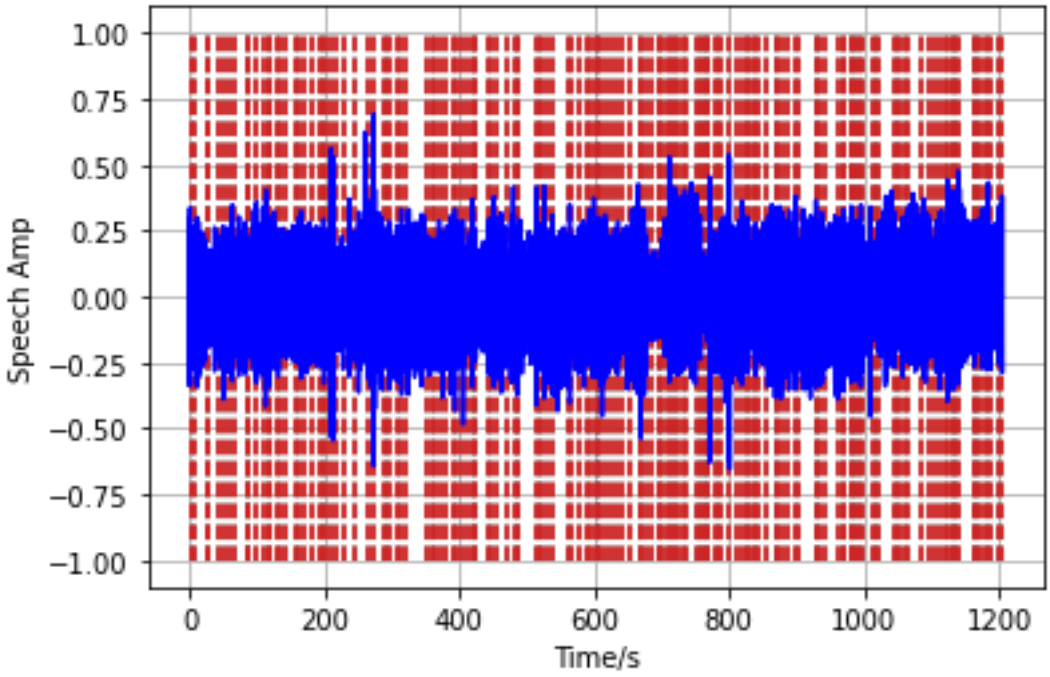


Figure 5.2: Speaker Change points

```
    FOR each segment in resegmented:
        excerpt = Create an excerpt for the current segment
        embedding = Extract the embedding of the current segment
        # Transpose the embedding and add to data list
        data.append(embedding.T)

    data = Convert data to a NumPy array
    data = Reshape the data array to be 2D
    RETURN data
END FUNCTION
```

The function 'embeddings' uses the Pyannote library to extract embeddings from an audio file for the purpose of speaker diarization. In order to discover d-vectors per frame, the function needs the path to the audio file, an array of re-segmented time intervals, and the length of each audio frame.

The Pyannote library loads a pre-trained model for extracting embeddings, creates an inference

object for the model with the entire audio file as the window, and loads the audio file and stores its path and duration in a dictionary. Then, for each segment in the 'resegmented' array, the function creates an excerpt based on the segment's start and end times and the given range, and uses the inference object to extract embeddings for the current excerpt from the audio file.

Finally, the embeddings are transposed and appended to a list, which is then converted to a NumPy array and reshaped to match the dimensions required for training the speaker diarization model. The function then returns the reshaped array of embeddings.

Number of Speakers in an audio is equal to the number of clusters formed.

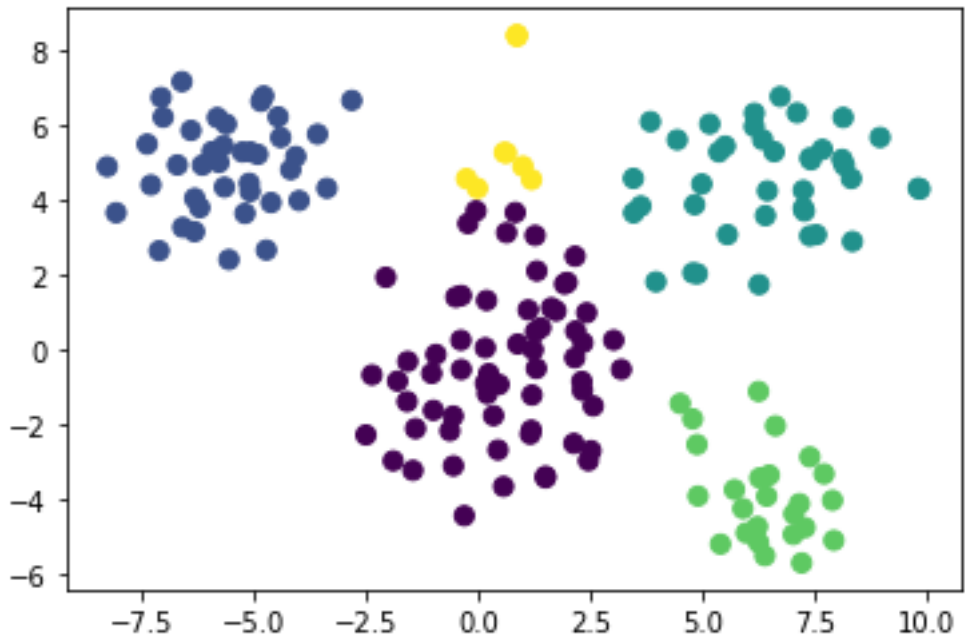


Figure 5.3: Clusters the same speakers

```
5
    Speaker_id       Offset          end
0            A     0.040000     3.194354
1            A     3.204354     8.442063
2            A     8.452063    25.717710
3            E    25.727710    28.713084
4            A    28.723084    43.434535
..         ...          ...          ...
171          A  1185.750204  1186.120000
172          A  1187.830000  1189.780476
173          A  1189.790476  1199.161338
174          A  1199.171338  1202.110000
175          A  1203.130000  1204.190000

[176 rows x 3 columns]
```

Figure 5.4: Speaker Timeline

5.2 Diarization error rate (DER)

Diarization Error Rate (DER) is a metric commonly used to evaluate the performance of a speaker diarization system. It is a measure of the degree to which the output of the system matches the reference or ground truth.

$$DER = \frac{(FalseAlarm + Miss + Overlap + Confusion)}{ReferenceLength}$$

where the numerator represents the sum of errors in the diarization output, and the denominator represents the length of the reference.

- False Alarm refers to the duration of segments in the system output that are considered to be speech but not present in the reference.
- Miss refers to the duration of segments in the reference that are considered to be speech but not present in the system output.
- Overlap refers to the duration of segments in the system output that overlap with segments in the reference, and are not properly assigned to a single speaker. However, our system does not consider overlap as an error.

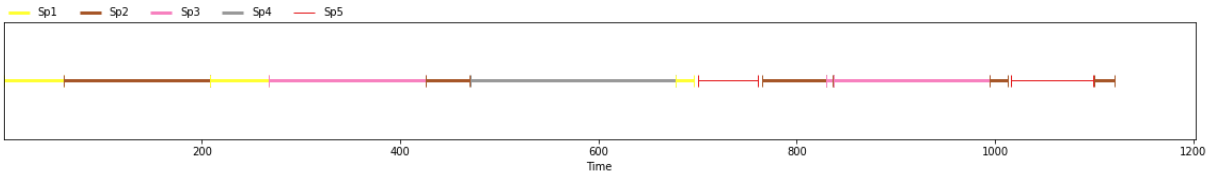


Figure 5.5: Our annotated file considered as ground truth

- Confusion refers to the duration of segments that are given to different speakers in the system output and the reference, after applying an optimal speaker assignment.

The numerator is the sum of these four types of errors, while the denominator is the total length of the reference.

It is important to note that DER can theoretically be larger than 1.0, indicating that the system output is worse than random. A DER value of 0.0 indicates a perfect match between the system output and the reference, while higher values indicate more errors in the output.

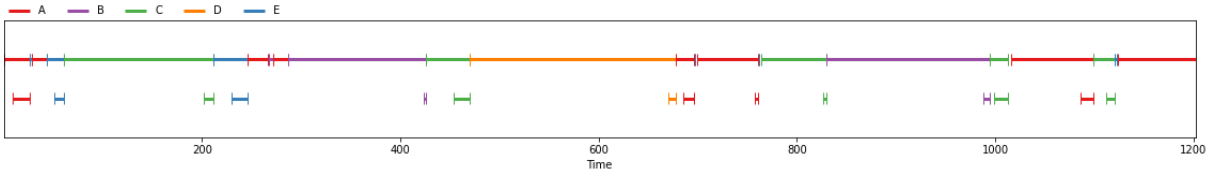


Figure 5.6: Output of our model considered as hypothesis

5.3 Results

For our test purposes we prepared our test audios and annotated those audio files to set the ground truth.

We choose this video for our testing purposes. The speaker change points for this certain test is given in the following figure

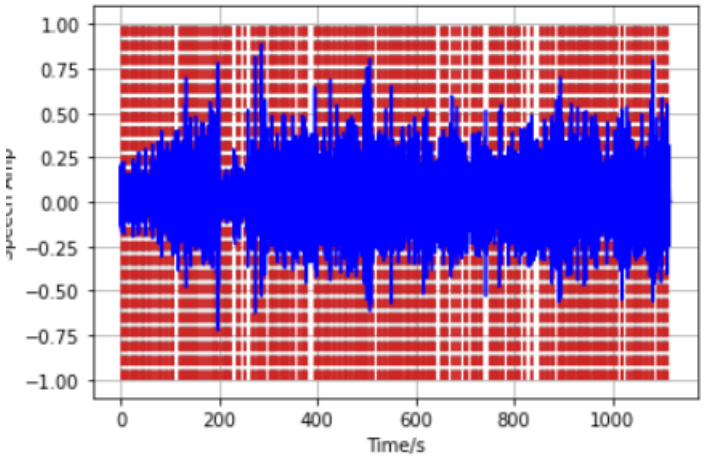
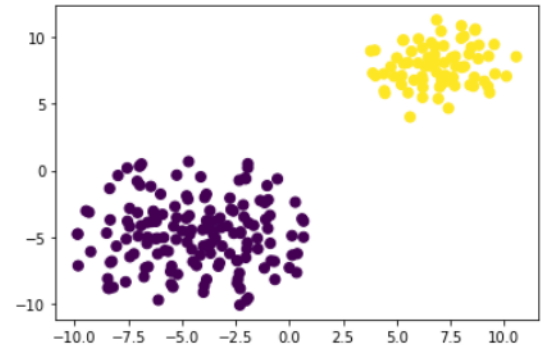


Figure 5.7: Speaker Change point for the test data

As there was 2 speakers in the audio, In the clustering phase we could see only two cluster form.



2

	Speaker_id	Offset	end
0	B	0.300000	2.590635
1	B	2.600635	4.773311
2	B	4.783311	9.301202
3	B	9.311202	12.621655
4	B	12.631655	16.336848
..	...	...	...
221	A	1091.709388	1094.741202
222	A	1094.751202	1101.312449
223	A	1101.322449	1105.515261
224	B	1105.525261	1108.417755
225	B	1108.427755	1116.630000

[226 rows x 3 columns]

Figure 5.8: Cluster and Speaker timeline

For our hypothesis we take the output of our model and for our ground truth we take the annotated file for the given test data. For DER we use -

```
# Import the required libraries
import pyannote.metrics.diarization as metrics

# Upload the reference annotation file
result_ref, ref_df = reference_gen('audioPath')

# Create a DiarizationErrorRate object
der = metrics.DiarizationErrorRate()

# Compute the DER between the reference and hypothesis diarization
der_score = der(result_ref, result_hypo)

# Print the DER score
print("DER={0:.3f}".format(der_score))

# Evaluate a particular segment of the audio file
detailed = True
uem = Segment(0, 40)
der_score = der(result_ref, result_hypo, detailed=detailed, uem=uem)

# Create a DiarizationPurity object
purity = metrics.DiarizationPurity()

# Compute the purity between the reference and hypothesis diarization for a
particular segment
purity_score = purity(result_ref, result_hypo, uem=uem)

# Print the purity score
```

```
print("Purity={0:.3f}".format(purity_score))

# Create a DiarizationCoverage object
coverage = metrics.DiarizationCoverage()

# Compute the coverage between the reference and hypothesis diarization for
particular segment
coverage_score=coverage(result_ref, result_hypo, uem=uem)

# Print the coverage score
print("Coverage={0:.3f}".format(coverage_score))
```

As output we get

DER = 0.188

Purity = 1.000

Coverage = 1.000

So we get around 18 pecent DER rate on the test audio we prepared.

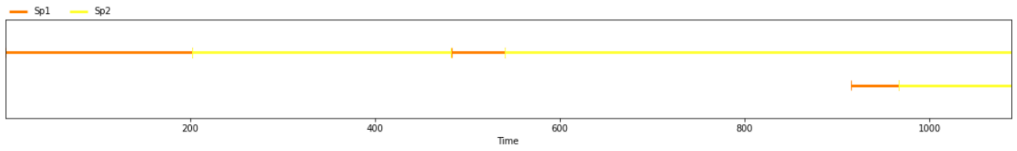


Figure 5.9: Ground Truth

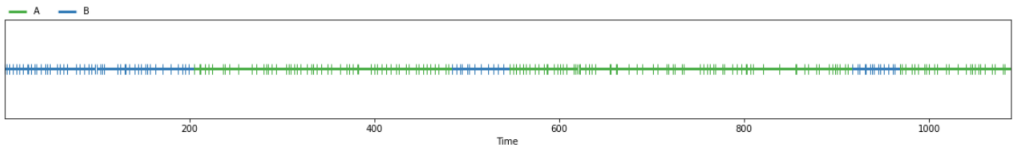


Figure 5.10: Hypothesis

5.3.1 Speech Recognition

For speech recognition part, we used google ASR.

- Define the input audio file path and the name of the bucket created before.
- Import the required libraries: AudioSegment, io, os, wave, google.cloud.speech, and google.cloud.storage.
- Set the Google Cloud credentials.
- Define the function "googleTranscribe" to transcribe the audio using the Google Cloud Speech API.
- Loop through the speaker-changed segments and extract the sentences spoken by each speaker.
- Print the speaker label and the extracted sentences.

Pseudo code:

5.3.2 Define input audio file path and bucket name

```
filepath = "/content/drive/My_Drive/SRU/"
bucketname = "diarization_thesis"
```

5.3.3 Import required libraries

```
from pydub import AudioSegment
import io
import os
from google.cloud import speech
import wave
from google.cloud import storage
```

5.3.4 Set Google Cloud credentials

```
os.environ['GOOGLE_APPLICATION_CREDENTIALS'] = 'xxx.json'
```

5.3.5 Define function to transcribe audio

```
def google_transcribe(audio_path):
# Load audio file using pydub
sound = AudioSegment.from_wav(audio_path)

# Export audio file to raw PCM bytes
with io.BytesIO() as wav_file:
    sound.export(wav_file, format="wav")
    wav_bytes = wav_file.getvalue()

# Set audio file properties
audio = speech.RecognitionAudio(content=wav_bytes)
config = speech.RecognitionConfig(
    encoding=speech.RecognitionConfig.AudioEncoding.LINEAR16,
    language_code="bn-BD",
    enable_word_time_offsets=True,
)

# Transcribe audio using Google Cloud Speech API
response = speech_client.recognize(config=config, audio=audio)

# Extract words and their start and end times
words = []
for result in response.results:
    for word in result.alternatives[0].words:
        words.append((word.word, word.start_time.seconds +
        word.start_time.microseconds * 1e-6, word.end_time.seconds +
        word.end_time.microseconds * 1e-6))
```

```
return words
```

5.3.6 Loop through speaker-changed segments and extract sentences spoken by each speaker

```
for segements in speaker_changed_seg:
# Initialize variable to store extracted sentences
sentences = ''
# Loop through words in transcript
for w in words:
# Check if word is spoken within the speaker segment
if w[1] >= segements[1] and w[2] <= segements[2]:
sentences += (w[0]+' ')
# Print speaker label and extracted sentences
print(segements[0]+':'+" "+sentences)
```

The results shown from this youtube video [link].

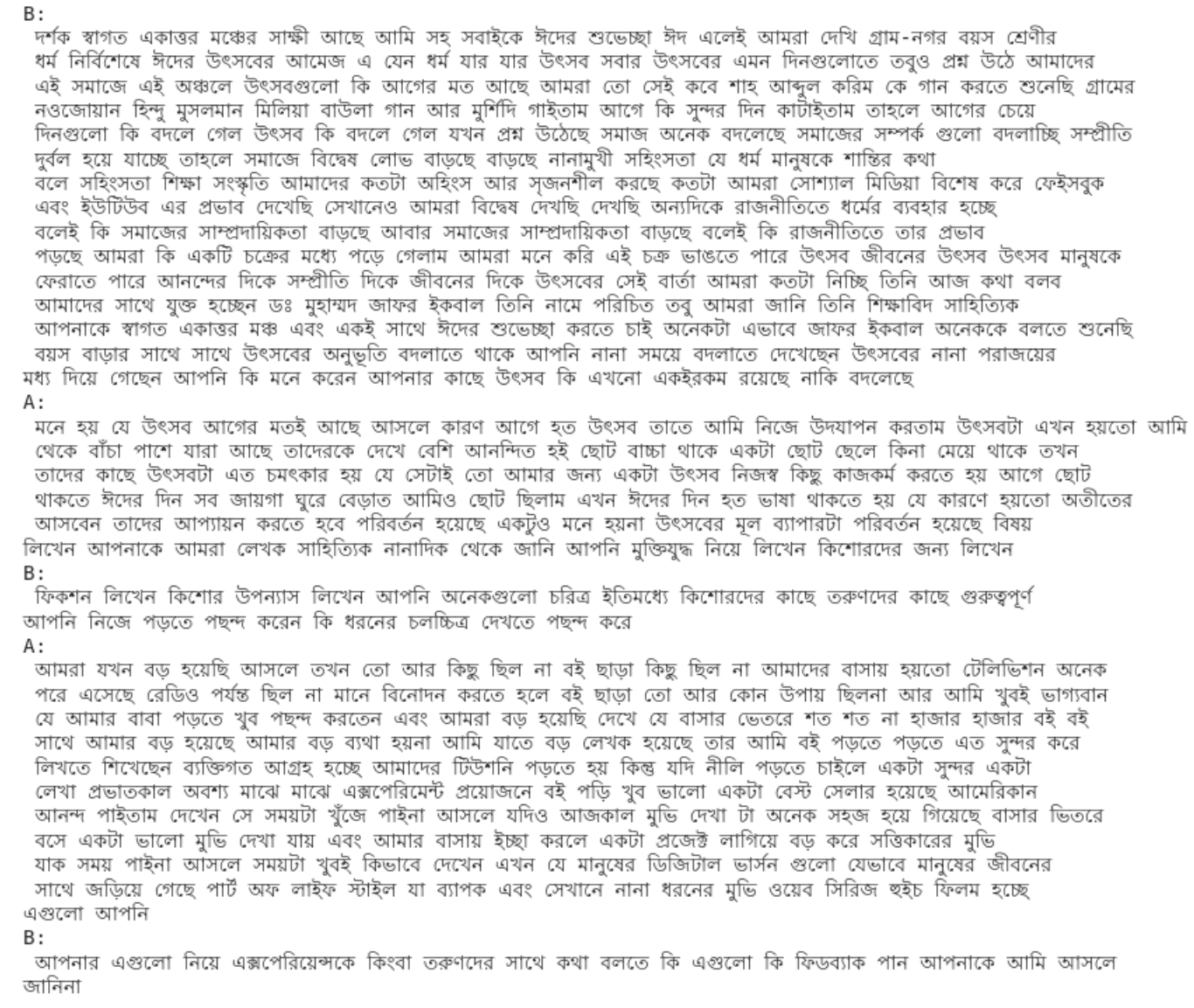


Figure 5.11: Bangla Speaker diarization with speech recognition

Chapter 6

Conclusion and Future works

6.0.1 Conclusion

In this project, we collected Bangla audio data from Youtube and annotated them using the ELAN software for sentence-level annotation, speaker segmentation, and speaker diarization. We then preprocessed the data by converting it to mono type and splitting the files using Audacity. We divided the data into training and test sets for our models.

We then used deep learning models, specifically a bi-directional LSTM model, to perform Speaker Diarization on the Bangla dataset. The model achieved DER of 18.8 percent on the test set, which shows promising results for future work in Bangla Speaker Diarization.

We also integrated Google Cloud ASR for Bangla language to automatically transcribe the audio into text. This can be useful for further downstream tasks such as topic segmentation or keyword spotting.

6.0.2 Future works

In future works, we can improve the accuracy of our ASR model by incorporating language models, and using more advanced pre-processing techniques like noise reduction, voice activity detection, and speech enhancement. We can also expand our dataset to include more diverse

speakers and speech scenarios.

For speaker diarization, we can explore more advanced models like deep clustering and end-to-end speaker diarization. We can also investigate the use of additional features such as speaker-specific prosody and lexical information to improve the performance of our model.

Overall, this project provides a solid foundation for future work in speaker diarization and Bangla ASR , which can have significant applications in various fields such as speech recognition, language modeling, and automatic subtitle generation.

References

[1] C. H. F. B. Florian Honig, Georg Stemmer, "Revising perceptual linear prediction (plp)," 2005.

[2] D. A. R. S. E. Tranter, "An overview of automatic speaker diarization systems," 2012, pp. 199–202.

[3] N. E. C. F. G. F. O. V. X. Anguera, S. Bozonnet, "Speaker diarization: A review of recent research," 2012, pp. 356–371.

[4] H. Z. J. Meng, J. Zhang, "Overview of the speech recognition technology," 2012, pp. 199–202.

[5] L. S. M. J. . C. G. Mao, H. H., "Speech recognition and multi-speaker diarization of long conversations," 2020.

[6] S. H. . S. I. Shafey, L. E., "Joint speech recognition and speaker diarization via sequence transduction." 2019.

[7] Y. F. Y. X. K. N. S. W. N. Kanda, S. Horiguchi, "Simultaneous speech recognition and speaker diarization for monaural dialogue recordings with target-speaker acoustic models," 2019.

[8] D. C. W. L. M. P. A. . M. I. L. Wang, Q., "Speaker diarization with lstm," 2017.

[9] A. P. Li Wan, Quan Wang and I. L. Moreno, "Generalized end-to-end loss for speaker verification,," 2017.

[10] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 11 1997. [Online]. Available: https://doi.org/10.1162/neco.1997.9.8.1735

[11] Y. Fujita, N. Kanda, S. Horiguchi, K. Nagamatsu, and S. Watanabe, "End-to-end neural speaker diarization with permutation-free objectives," 2019. [Online]. Available: https://arxiv.org/abs/1909.05952

[12] S. Adavanne and T. Virtanen, "A report on sound event detection with different binaural features," 2017. [Online]. Available: https://arxiv.org/abs/1710.02997

[13] J. R. Hershey, Z. Chen, J. L. Roux, and S. Watanabe, "Deep clustering: Discriminative embeddings for segmentation and separation," *CoRR*, vol. abs/1508.04306, 2015. [Online]. Available: http://arxiv.org/abs/1508.04306

[14] K. N. H. S. X. Y. N. K. . W. S. Fujita, Y., "End-to-end neural speaker diarization with self-attention," 2019.

[15] A. Graves and J. Schmidhuber, "Framewise phoneme classification with bidirectional lstm and other neural network architectures," 2005.

[16] M. Hossan, S. Memon, and M. Gregory, "A novel approach for mfcc feature extraction," 01 2011, pp. 1 – 5.

[17] H. Bredin, R. Yin, J. M. Coria, G. Gelly, P. Korshunov, M. Lavechin, D. Fustes, H. Titeux, W. Bouaziz, and M.-P. Gill, "pyannote.audio: neural building blocks for speaker diarization," May 2020.

[18] J. M. Coria, H. Bredin, S. Ghannay, and S. Rosset, "A comparison of metric learning loss functions for end-to-end speaker verification," in *Statistical Language and Speech Processing*, L. Espinosa-Anke, C. Martín-Vide, and I. Spasić, Eds. Springer International Publishing, 2020, pp. 137–148.