JOHANNES KEPLER
UNIVERSITY LINZ

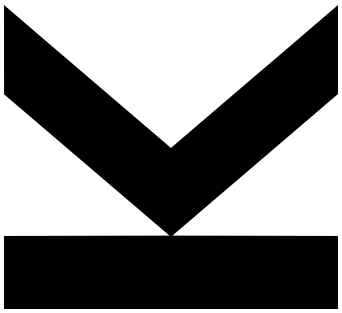
Author
Michael Reinegger
k12102754

Submission
Institute of Computational Perception

Thesis Supervisor
a.Univ.-Prof, Dr. **Josef Scharinger**

August 2024

Optimizing LSTM-based Speaker Diarization: Comparing Architectures and Clustering Methods



Bachelor Thesis

to obtain the academic degree of

Bachelor of Science

in the Bachelor's Program

Informatik

JOHANNES KEPLER UNIVERSITY LINZ
Altenbergerstraße 69
4040 Linz, Austria
www.jku.at
DVR 0093696

Abstract

This thesis explores the optimization of Long Short-Term Memory (LSTM) neural networks[1] for the task of speaker diarization. Speaker diarization[2] involves identifying and labeling unique speakers in an audio file, delineating segments with start and end times for continuous speech segments. The core idea of the speaker diarization system, used for this implementation, lies in the utilization of d-vectors, generated by the LSTM network from audio features like Mel-Frequency Cepstral Coefficients (MFCCs)[3] or filter banks (FBANKs)[4], which distinguish between different speakers.

Initially, 30ms audio segments are processed to extract spectral features that serve as inputs to the LSTM network. The LSTM then learns to produce d-vectors[5], which are similar for the same speaker and distinct for different speakers. These d-vectors are subsequently clustered using clustering algorithms to generate a group of segments per speaker, which ideally contains all segments that belong to the speaker.

This study evaluates various clustering algorithms, including Spectral Clustering[6], offline K-Means[7, 8], DBSCAN[9], and Agglomerative Clustering[10], assessing their impact on system performance. Our implementation achieves a Diarization Error Rate (DER)[11], on the Vox Converse dataset[12], of approximately 20%, which, while not bad, falls short compared to the state-of-the-art, of 4-9%[13, 14] due to constraints such as a limited dataset and computational resources. However, the research questions discussed in this paper will show how one can improve this system further.

We will take a close look at the LSTM-based speaker diarization[15] by comparing different neural network architectures and clustering techniques. It also investigates the influence of hyperparameters such as layer size and cluster parameters on diarization accuracy. The results of our experiments show how to enhance the diarization accuracy through different network architectures and hyperparameter tuning, as well as further refinements of the d-vector based approach to speaker diarization.

Kurzfassung

Diese Arbeit untersucht die Optimierung von Long Short-Term Memory (LSTM) neuronalen Netzwerken[1] für die Sprecherdiarisation. Sprecherdiarisation[2] ist die Identifizierung und Kennzeichnung einzelner Sprecher in einer Audiodatei, wobei Segmente mit Start- und Endzeiten für zusammenhängende Sprachabschnitte abgegrenzt werden. Die Kernidee des für diese Implementierung verwendeten Sprecherdiarisationssystems liegt in der Nutzung von d-Vektoren, die vom LSTM-Netzwerk aus Audiomerkmalen wie Mel-Frequency Cepstral Coefficients (MFCCs)[3] oder Filterbänken (FBANKs)[4] generiert werden und zwischen verschiedenen Sprechern unterscheiden.

Zunächst werden 30ms lange Audiosegmente verarbeitet, um spektrale Merkmale zu extrahieren, die als Eingabe für das LSTM-Netzwerk dienen. Das LSTM lernt dann, d-Vektoren[5] zu erzeugen, die für denselben Sprecher ähnlich und für verschiedene Sprecher unterschiedlich sind. Diese d-Vektoren werden anschließend mittels Clustering-Algorithmen gruppiert, um eine Gruppe von Segmenten pro Sprecher zu generieren, die idealerweise alle zu diesem Sprecher gehörenden Segmente enthält.

Diese Studie evaluiert verschiedene Clustering-Algorithmen, einschließlich Spectral Clustering[6], offline K-Means[7, 8], DBSCAN[9] und Agglomerative Clustering[10], und bewertet deren Auswirkungen auf die Systemleistung. Unsere Implementierung erreicht eine Diarization Error Rate (DER)[11] von etwa 20% auf dem Vox Converse Datensatz[12], was nicht schlecht ist. Aber im Vergleich zum aktuellen Stand der Technik von 4-9%[13, 14] aufgrund von Einschränkungen wie einem begrenzten Datensatz und begrenzten Rechenressourcen zurückbleibt. Die in dieser Arbeit diskutierten Forschungsfragen zeigen jedoch, wie man dieses System weiter verbessern kann.

Wir werden die LSTM-basierte Sprecherdiarisation[15] genau untersuchen, indem wir verschiedene neuronale Netzwerkarchitekturen und Clustering-Techniken vergleichen. Zudem wird der Einfluss von Hyperparametern wie Schichtgröße und Cluster-Parametern auf die Diarisationsgenauigkeit untersucht. Die Ergebnisse unserer Experimente zeigen, wie die Diarisationsgenauigkeit durch unterschiedliche Netzwerkarchitekturen und Hyperparameter-Tuning sowie weitere Verfeinerungen des d-Vektor-basierten Ansatzes zur Sprecherdiarisation verbessert werden kann.

Contents

1 Introduction

1.1 Motivation

Speaker Diarization involves the use of algorithms, artificial intelligence or other methods to automating the task of labeling audio files with multiple speakers[2]. The information provided by such a system consists of segments that belong to a unique speaker. Each segment contains the start time and either the length or the end time of the segment, plus also either the name of the speaker or some unique ID. The possible use of such a system is basically everywhere where multiple speakers are talking, and the process should not just be digitally recorded but also transcribed. Possible scenarios would be a court hearing or a normal meeting. To generate such an automated transcript, speaker Diarization is only a small part.[15, 16]

Other parts of such a system may be Speech Detection[17] to remove empty segments, possible noise reduction or other processes to clean the audio to achieve better result, a Speech-to-Text system to generate the actual text and finally Speaker Diarization to group together the words with the individuals who said them.

Due to the nature of most recording scenarios, automated transcribing is an extremely challenging task. Problems include but are not limited to background noise, overlapping speech, different accents, different languages, speakers with similar sounding voices and much more. Nevertheless, the field of Speaker Diarization is constantly evolving and improving[18].

In this thesis, we will focus on the use of an LSTM network[1] that generates d-vectors[5] from speaker features for Speaker Diarization[2]. In particular, we will explore how LSTM[1] networks can obtain d-vectors (discriminative or deep vectors)[5] from speaker features to enhance the accuracy and efficiency of the Diarization process. The LSTM's ability to capture long-term dependencies in sequential data makes it especially interesting for analyzing the temporal aspects of speech, where we can use information from the past as well[19]. The whole implementation is based on the wonderful paper "Speaker Diarization with LSTM" by Wang et al. [15].

1.2 Research Questions

This thesis explores three research questions:

1. How do hyperparameters affect the performance of the Diarization system. In particular, we will look at how the parameters σ and $p_{percentile}$ affect the spectral clustering[6]. Furthermore, the parameters for the calculation of the Mel filter bank energies[4], in particular the number of Fast Fourier Transformations[20] and the hop length, will be tested.

2. How do different clustering methods affect the performance of the system. We will only look at clustering algorithms that can generate clusters without knowing the number of clusters, as the number of speakers is not fixed or known in advance. In particular, we will look at Spectral Clustering[6], offline K-Means[7, 8], DBSCAN[9] and Agglomerative Clustering[10]. Here, the evaluation focuses on how well the clusters per speaker are formed and how accurate the algorithms are at predicting the correct number of speakers.

3. What impact does the structure of a neural net have on the system's performance. First a more recent RNN[21] node will be examined, the BiLSTM (Bidirectional LSTM)[22] which has the ability to take the forward and backward input since it consists of the LSTM[1] node facing opposite direction. In theory, this should improve the amount of information obtained for the temporal speech features[23]. But the number of hidden layers as well as the number of nodes per layer and their performance impact will be part of this question.

1.3 Outline

This introduction chapter highlights the motivation behind this thesis and lays out the important research question that will be answered. Next, we will go over the used Materials and the theory needed to create a LSTM-based Speaker Diarization system. Chapter 3 will focus on the implementation by explaining the theory with real code examples and useful libraries. Subsequently, Chapter 4 briefly explains how to use the implementation before moving on to Chapter 5. This chapter contains an in-depth analysis of the research question. Finally, we will summarize the research questions and look at possible ways to improve the implementation.

2 Material and Methods

2.1 Material

As for the implementation, this paper heavily relies on the proposed structure from "Speaker Diarization with LSTM" by Wang et al. [15] but we use different datasets as well as explore different clustering algorithms in the later chapters. Different datasets were used for different purposes.

For training, the Vox Celeb 1 [24] dataset was used because it has a relatively large number of different speakers, however the amount of speech segments was limited to 150 Minutes to be able to train the model on a normal pc within a day instead of multiple days or weeks.

The testing three testing datasets sets were used, most notably the Call Home dataset[25] as it only consists of 2 Person speech, it is great for implementing the training and testing parameters for the feature extraction without having to rely on the clustering algorithm. Then, for testing the system with sort of perfect conditions, the Vox Celeb dataset was used to synthetically generate audio files with a known number of people. But for the actual Diarization Error Rate[11] Calculation, the Vox Converse[12] dataset which features real live news recordings was used.

2.2 Methods

This section presents the theoretical foundation that is needed to build the system. In particular, we will look at how to extract audio features from an audio file. As well as, how a neural net functions with special focus on sequential data. And finally, how clustering algorithms, especially offline K-Means[7], Spectral Clustering[6] and DBSCAN (Density-Based Spatial Clustering of Applications with Noise)[9] generate clusters. The methodology for the actual experiments performed on the implementation of the LSTM based speaker diarization system are explained in the respective chapter 5 "Experimental Results".

2.2.1 Feature Extraction and Audio Processing

Mel Filter bank (FBANK)

Mel is a unit for sound similar to decibel, but more focused on the human voice frequency range [26]. With this representation, we can generate the filter bank [4].

Input: The input to the Mel Filter Bank is typically the power spectrum of an audio signal, obtained through a Fourier transform of the windowed signal.

Output: The output is a set of filter bank energies, each corresponding to a specific frequency range that approximates how the human auditory system perceives sound.

Calculation: The Mel Filter Bank is computed by applying a series of triangular filters to the power spectrum. These filters are spaced according to the Mel scale, which is approximately linear below 1000 Hz and logarithmic above. The process consits of the following steps:

1. Converting the frequency scale to the Mel scale
2. Creating a set of triangular filters on this Mel scale
3. Applying these filters to the power spectrum
4. Summing the energy in each filter

For a more detailed explanation of the calculation and equations, refer to [4].

Another often used feature is the MFCC or Mel Frequency Cepstral Coefficients[3], which basically is a compressed form of the filter bank to reduce the computational load by only removing as little important information as possible for speech features.

VAD

A Voice Activity Detector (VAD)[27] is an algorithm to filter segments of audio if they contain speech or not. An elementary approach for a VAD would be to use the average volume or energy in a segment. However, this only works if there is no background noise or music. But to be able to detect just speech, such a system can get very complex and resource intensive. Like a Gaussian Mixture Model (GMM)[28] as it needs to have its own dataset and training. That is why this implementation uses the fast but reasonably reliable WebRTC VAD[17] for speech detection.

2.2.2 Neural Nets for Sequential Data

Neural networks are mathematical structures inspired by the human brain. They consist of nodes, similar to neurons, organized in layers. These networks learn from data, adjusting the strength of connections between neurons to recognize patterns and make predictions. They are widely used in various fields like computer vision, natural language processing, speech recognition and many more[29]. There are two main types of neural nets, convolutional (CNNs) and recurrent (RNNs). CNNs are better at picture related machine learning, whereas RNNs are better with sequential data like natural language processing.[30]

This is because of their ability to retain information from the past and use that to influence their output. They do this by saving an internal state, like a memory, that gets updated with each new input. This memory allows RNNs[21] to capture dependencies across different time steps. However, basic RNNs can have problems with long-term dependencies due to issues like vanishing gradients. That's where advanced RNN architectures like Long Short-Term Memory (LSTM) [1] or Gated Recurrent Units (GRU) [31] come into play. These architectures introduce special gating mechanisms, as described in section 2.2.2, allowing the network to selectively remember or forget information over long sequences.

Long-Short-Term-Memory (LSTM)

Long Short-Term Memory (LSTM) [1] nodes are part of RNNs[21] that are good at capturing and preserving long-term dependencies in sequential data, like audio. These nodes possess internal mechanisms-namely, input, forget, and output gates—that regulate the flow of information, allowing the network to selectively retain or discard relevant speaker-specific features across extended time intervals. The nodes process acoustic features sequentially, learning to identify and track speaker-specific characteristics such as pitch, tone, and speaking style. This temporal information allows the node to maintain speaker identities across utterances. The core architecture of an LSTM node consists of the following key elements, that can also be seen in figure 2.1:

- **Cell State**: This is the primary mechanism for information flow. Its purpose is to provide the information sequentially to all processing gates.
- **Input Gate**: This gate decides if new information should be stored in the cell state. This could be a specific acoustic feature of a particular speaker.
- **Forget Gate**: This component decides if information should be discarded from the cell state. This could ideally be some static background noise that is not needed.

- **Output Gate**: This gate controls what information from the cell state should be used as output. This basically decides how it is contributing to the d-vector, which ideally would be vastly different for different speakers.

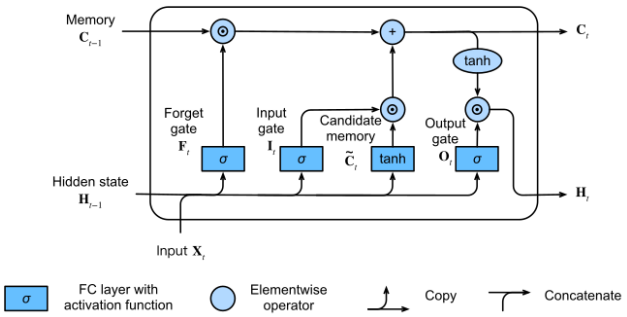


Figure 2.1: LSTM Architecture[32]

Bidirectional-Long-Short-Term-Memory (BiLSTM)

Basically, a BiLSTM[22] is just two LSTM Nodes[1] connected, with each other one in the forward pass direction and one in the backward pass direction. Figure 2.2 shows how this connection works inside the BiLSTM node as well as between multiple nodes.
With this added amount of memory, the BiLSTM node has twice the number of parameters to tune for the same neural net architecture. This can improve the performance of the model but also increase the computational resources to train and run the model compared to a LSTM neural network.[23]

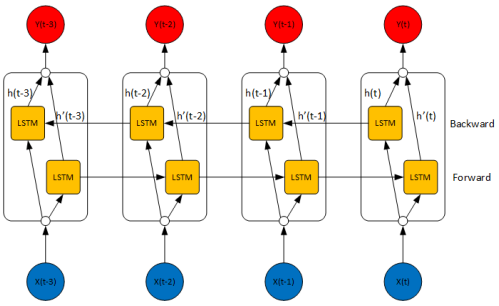


Figure 2.2: BiLSTM Architecture[32]

Generalized-End-to-End loss

A loss function is used to determine how well a neural net is performing during training. The loss is used by the optimizer to change the parameters of the neural net to reduce the amount of loss. For this implementation, the GE2E-loss function is used as suggested by Wang et al. [15].

GE2E[33] works by processing multiple utterances from multiple speakers to generate embedding vectors. It then creates a similarity matrix by computing the cosine similarity between each utterance embedding and the centroid (average) of all embeddings for each

speaker. The loss function aims to maximize the similarity for positive pairs (same speaker) while minimizing it for negative pairs (different speakers), typically using a softmax or sigmoid function and cross-entropy loss. The mathematical formulation of GE2E loss can be expressed as:[33]

$$L_{GE2E} = \sum_{j,i} -\log \frac{\exp(w \cdot \cos(e_{ji}, c_j) + b)}{\sum_k \exp(w \cdot \cos(e_{ji}, c_k) + b)} \tag{2.1}$$

Where e_{ji} is the embedding of the i-th utterance from speaker j, c_j is the centroid of speaker j, w is a trainable weight, and b is a trainable bias. This loss function is ideal for speaker diarization as it excels at speaker discrimination.[33]

2.2.3 Clustering

A clustering algorithm is one of the most important parts of a machine learning system that generates data without labels. This is because the labels are generated by the clustering algorithm. There are two main categories for clustering algorithms. That is offline, the algorithm only produces the labels after all embeddings are generated, and online, labels are generated directly after a segment is generated without knowing about the future segments.[34] This paper uses four different clustering algorithms, that will be explained in this chapter, which are all offline algorithms.

Spectral Clustering

This clustering method works by following these 5 Steps[6]:

1. Construct the affinity matrix A as follows:

   a) When $i \neq j$, the element A_{ij} represents the cosine similarity between the embeddings of the i-th and j-th segments.

   b) For the diagonal elements (where $i = j$), set each A_{ii} to the largest value found in its corresponding row, excluding the diagonal position itself.

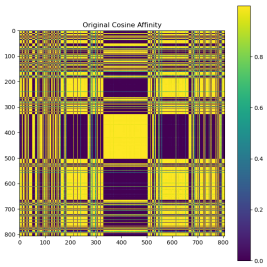


Figure 2.3: Affinity Matrix

2. Apply a sequence of refinement operations described in figure 2.4

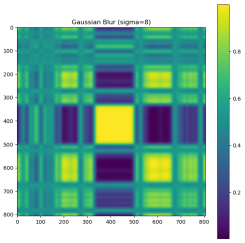


Figure 2.4.a: Gaussian Blur

1. The matrix is first blurred using a Gaussian filter with a standard deviation of σ.

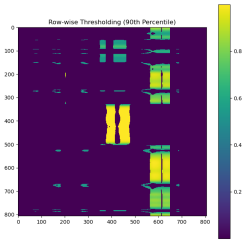


Figure 2.4.b: Threshold

2. For each row, elements smaller than the row's p-percentile are scaled with a small multiplier, like 0.01.

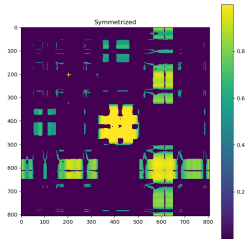


Figure 2.4.c: Symmetrize

3. The matrix is symmetrized to ensure it is symmetric with $Y_{ij} = max(A_{ij}, A_{ji})$.

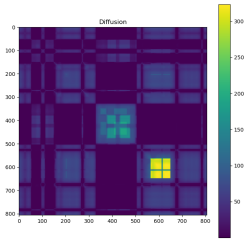


Figure 2.4.d: Diffusion

4. The matrix is diffused using $Y = AA^T$.

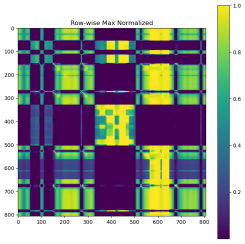


Figure 2.4.e: Normalize

5. The matrix is normalized with Row-wise Max Normalization with $Y_{ij} = \frac{A_{ij}}{max_k A_{ik}}$

Figure 2.4: Matrix Refinement Steps for Spectral Clustering

3. Perform eigen-decomposition on the refined affinity matrix with the eigenvalues as $\lambda_1 > \lambda_2 > ... > \lambda_n$. The maximal eigen-gap determines the number of clusters k:

$$k = arg \max_{1 \leq k \leq n} \frac{\lambda_k}{\lambda_{k+1}} \tag{2.2}$$

4. Let $v_1, v_2, \ldots, v_{\tilde{k}}$ denote the eigenvectors corresponding to the $\tilde{k}$ largest eigenvalues. We then replace the i-th segment embedding with the corresponding components from these eigenvectors $e_i = [v_{1i}, v_{2i}, \ldots, v_{\tilde{k}i}]$. This transformation effectively projects each segment embedding onto the subspace spanned by the principal eigenvectors, thereby capturing the most significant features of the data.

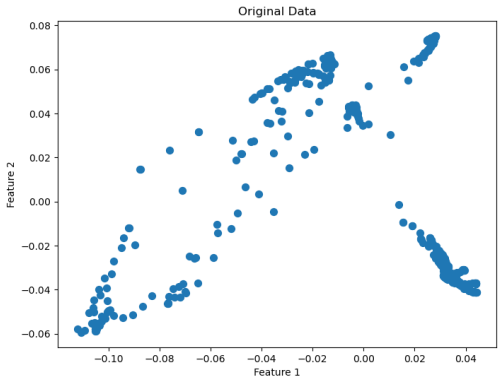


Figure 2.5: Spectral Clustering Input PCA

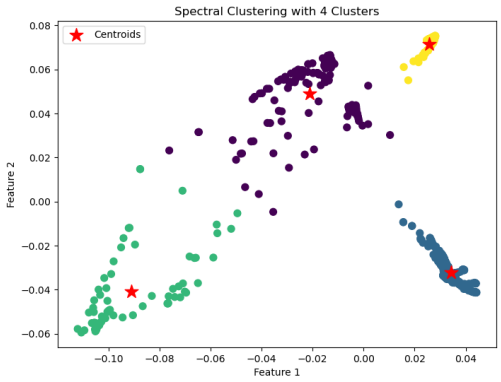


Figure 2.6: Spectral Clustering Output

5. Now that the data is transformed into a usable subspace, we can use the k-means++ algorithm [35] to generate the clusters and centroids as seen in figure 2.6 from the data shown in figure 2.5. One downside to this approach is that the time complexity is $O(n^3)$ and therefore not really ideal for large datasets.

Offline K-Means with Elbow Technique

The K-Means[7] algorithm needs to know the number of clusters in advance. This is not possible with our data. That is why we use the normal K-Means algorithm to generate clusters with progressively increasing cluster size. to determine which cluster size is likely, it uses the elbow technique[8], which determines that optimal point by at which point the fit does not improve anymore. With a time complexity of $O(n * k * d * i)$, with n the number of data points, d the dimension of a data point, k the number of clusters and i the number of iterations to find the best fit, k-means plus the elbow technique can become slow if there are many speakers in a long audio file.

Density-Based Spatial Clustering of Applications with Noise (DBSCAN)

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is a powerful clustering algorithm that is known for its ability to identify clusters of arbitrary shape within datasets, as well as outliers[9]. Since it does not need the number of clusters in advance, it uses the concept of density to determine clusters, grouping together points that are closer together while marking points in low-density regions as outliers or noise. This ability to generate a cluster of outliers is especially interesting for speaker diarization as the Voice Activity Detection (VAD) component occasionally misclassifies non-speech segments

as speech, introducing potential errors. This clustering algorithm is also quite fast as it has a time complexity of $O(nlogn)$

Agglomerative Clustering

Agglomerative clustering, also known as hierarchical clustering, is a bottom-up approach to cluster generation that begins with each data point as a separate cluster and iteratively merges the closest clusters until a stopping condition is met [10]. This method creates a tree-like structure called a dendrogram[36]. The algorithm operates by calculating the distances between all pairs of clusters using a specified metric (e.g., Euclidean distance, cosine similarity) and a linkage criterion (e.g., single linkage, complete linkage, or Ward's method) to determine cluster proximity. It is similar to DBSCAN[9] as it generates the clusters dynamically, finding the optimal number of clusters on its own. However, it cannot find outliers. In the context of speaker diarization, Agglomerative Clustering can be effectively employed to group similar speech segments. However, it's worth noting that the algorithm's computational complexity is $O(n^3)$, therefore, not good for massive datasets.

2.2.4 Diarization Metrics

There are plenty of different ways to measure the performance of a speaker diarization system. The one used most is the Diarization Error Rate (DER)[11]. The DER is defined as:[11]

$$DER = \frac{FA + MISS + SE}{TST} \tag{2.3}$$

- False Alarm (FA): This is when our system thinks there's speech, but it's actually silence.
- Missed Detection (MISS): This happens when there's actually speech, but our system doesn't catch it.
- Speaker Error (SE): This is when our system correctly detects speech but assigns it to the wrong speaker.
- Total Speech Time (TST): This is the total amount of time that speech is present in the audio file.

Another used metric is the Jaccard Error Rate (JER)[37] which is an improved version of DER but since it has little adoption until now, we will not use it as there is limited data to compare it to other system. For each speaker, JER looks at the intersection (overlap) and union of the reference and hypothesis speaker segments. It's calculated like this:[37]

$$JER = 1 - \frac{1}{N} \sum_{n=1}^{N} \frac{|REF_n \cap HYP_n|}{|REF_n \cup HYP_n|} \tag{2.4}$$

Where N is the number of speakers, REF_n is the reference for speaker n, and HYP_n is the hypothesis for speaker n.

A few more metrics that would be interesting but are also seldom published and are more interesting for testing and understanding the behavior of the system are:[11]

- Purity: This measures how well our system groups speech segments from the same speaker together.
- Coverage: This tells us how well our system avoids splitting a speaker's speech across multiple clusters. Or how good the vad performs as to not cut away speech segments.
- Completeness: This is defined as the ratio of the total duration of the longest segment for each speaker to the total duration of all segments for that speaker in the reference annotation.

Other metrics like Identification Error Rate[11] are not relevant for our system as we cannot identify speakers, we only differentiate between speakers.

3 Implementation

This section will explain how the LSTM-based Speaker Diarization system proposed by Wang et al.[15] can be implemented in Python. The implementation was done in python with the help of a few libraries. Namely, PyTorch[38], librosa[39], WebRTC[17], NumPy[40], Matplotlib[41], sklearn[42], SciPy[43] and pyannotate[44] as well as python as the programming language used for the implementation. The Neural-Net-Architecture as well as the data preprocessing and clustering of d-vectors is, as mentioned before, based on the paper "Speaker Diarization with LSTM" by Wang et al.[15].

3.1 Feature extraction and Data pre- and post-processing

As stated in chapter 2 we need to prepare data from 3 different datasets: Vox Celeb 1[24], Vox Converse[12] and Call Home[25]. Each Dataset consists of audio files with random IDs as names that correspond to the names of the label file, that have the same ID. Librosa [39] is used to read the audio file with a sampling rate of 16khz. This audio data is then split into segments according to the label files. Those files are processed as follows.

Vox Celeb 1 and Vox Converse use the same label data, RTTM[45] (Rich Transcription Time Marked) files. Those files consist of multiple entries, one per line with space separated values. For speaker diarization only the 4th 5th and 8th values are important as they are the start time, duration, and speaker ID of the segment. An entry could look like this "`SPEAKER ahnss 1 252.320000 46.680000 <NA> <NA> spk00 <NA> <NA>`". Here we can see the segment starts at 252.32 seconds and has a duration of 46.68 seconds and the speaker for this segment is spk00. This makes parsing the files easy as we can read the RTTM file line by line and then split every line by the whitespace and only take the 3 relevant segments and save them in a tuple together with the split part of the audio from the original audio file.

For the Call Home dataset the label extraction is a bit different as the data comes in 3 arrays with the first one being all start timestamps, the second one the end timestamps and the third one being the speaker IDs. This means that for each entry, the data we need to put into the tuple has the same index. For example, the arrays `[10.0, 18.5, 20.7, ...]` `[15.7, 19.5, 34, ...]` `[‘‘A’’, ‘‘B’’, ‘‘A’’, ...]` would tell us that the speaker B has spoken from second 18.5 to 19.5 in one segment.

Now that we have tuples like this `(Speaker_ID, Start_Time, Duration, Audio_data_Segment)` we further split them into 30ms segments. The 30ms were chosen as it is common to use segments between 10 and 50ms[15] and the WebRTC[17] is only compatible with 10, 20 and 30ms windows. Subsequently, we can once again use the library librosa to generate the Mel-spectrograms for each segment like this.

```
mel_spectrogram = librosa.feature.melspectrogram(y=audio_segment, sr=sr,
n_mels=n_mels, n_fft=n_fft, hop_length=hop_length, power=1)
db_spectrogram = librosa.power_to_db(mel_spectrogram, ref=np.max)
```

This code first turns the librosa audio into Mel-spectrograms and then uses a db scale to turn them into Mel-Log-spectrograms to better captures the human speech range.

Now that the data is in a machine useable format, we need to apply the WebRTC [17] to remove every 30ms segment that has no speech to increase the likelihood of the neural net to learn the difference between different speakers. This step is fairly straightforward as the WebRTC library has the function `vad.is_speech(Audio_segment, SampleRate)` and returns true if speech was detected. But there is one small caveat the library expects the audio data to be in 16-bit PCM but librosa audio data is in floating point, therefore the audio data needs to be converted before being used by the VAD.

Now the data can be used for training the model or evaluating the model. For running the model, we use the same steps but without the label data.

3.2 Neural Network

Generating a neural net in PyTorch[38] is relatively easy, as we only need to define the structure and do not need to implement the different nodes and layer connections. The network structure proposed by Wang et al. [15] would be defined in PyTorch like this:

```
class LSTMModel(nn.Module):
def __init__(self, input_size, hidden_size, num_layers, projection_size):
super(LSTMModel, self).__init__()
self.lstm = nn.LSTM(input_size, hidden_size, num_layers, batch_first=True,
bidirectional=True)
self.projection = nn.Linear(hidden_size, projection_size)

def forward(self, x):
_, (hidden, _) = self.lstm(x)
```

```
        d_vector = hidden[-1]  # Get the last hidden state
        d_vector = self.projection(d_vector)  # Apply linear projection
        return d_vector
```

This neural net can now be used for training or running an already trained model. It is important to note that for running the model, one needs to use the same parameters as the trained model used. We will investigate changing some parameters and how this can affect the performance in the chapter 5.3. Furthermore, part of the LSTM Model is the Generalized End to End (GE2E) loss function[33]. This function has two parameters, namely b and w, that are part of the GE2E loss function described in equation (2.2), that are also needed to be tuned during training as described in the next chapter. The GE2E loss function is defined in python with the help of the NumPy[40] library by following the exact steps described in "Generalized end-to-end loss for speaker verification"[33].

3.3 Training the Model

The training consists of the following segments:

- Batch generation and sampling, where the prepared data is put into batches and the sampler selects a batch to give the model for training.
- The Model and Parameter Generation, where the model, parameters for the loss function, the learning rate scheduler and the optimizer are created and configured.
- The training loop, here the model is trained and evaluated with the help of the loss function and after each evaluation the optimizer changes the parameters of the model to try to reduce the loss.

Those steps will be further explained in the following sections.

Batchsampler and Dataloader

We use the prepared data as described in chapter 3.1 to generate batches of data with a specific size. If we used the default implementations of the batch sampler, it would just randomly select segments. This can lead to a problem if the selection only takes segments from a single speaker or to little segments per one speaker. Such a skewed batch can negatively impact the performance of the model during training as the model sometimes sees data that results in a bad loss calculation as the GE2E[33] loss function needs at least two different speakers. And if the amount of data is too small per speaker, the model

never learns to differentiate between patterns as there is too little information. To ensure the data is diverse enough, we implemented our own custom batch sampler to ensure at least two different speakers and limit the amount of max different speakers and a fixed number of segments per speaker to guarantee enough data points. This batch sampler has a custom iterator that selects a random number of speakers randomly defined by min and max number of speakers. Then it iterates of the segments of those speakers and takes a random selection. The selected segments cannot be used again, and as soon as a speaker has no more segments it can also no longer be selected. To ensure repeatability, the seed for the random number generator can be set in the constructor. This iterator implementation looks like this:

```
def __iter__(self):
    random.seed(self.seed)
    indices_per_speaker = {speaker: list(indices) for speaker, indices in
                           self.speaker_to_indices.items()}
    num_batches = len(self)
    if Hyperparameters.dyn_num_speakers:
        self.speakers_per_batch = random.randint(min_num_speakers,
                                                 max_num_speakers)
    for _ in range(num_batches):
        speakers = random.sample(list(indices_per_speaker.keys()),
                                 self.speakers_per_batch)
        batch = []
        for speaker in speakers:
            if len(indices_per_speaker[speaker]) >= self.samples_per_speaker:
                batch.extend(random.sample(indices_per_speaker[speaker],
                                           self.samples_per_speaker))
            else:
                batch.extend(indices_per_speaker[speaker])
                indices_per_speaker[speaker] = []
        yield batch
```

It can then be used by the default PyTorch[38] data sampler implementation like this:

```
batch_sampler = MultipleSpeakersBatchSampler(speaker_to_indices)
train_loader = DataLoader(train_dataset, batch_sampler=batch_sampler)
```

As for changing the number of speakers and segments, please refer to chapter 4. Now that the data is prepared and loaded, we can continue with configuring the model for training.

Creating Model and parameters

We have already discussed how the LSTM-Model class is defined. We can now create the model as well as the optimizer that tunes the parameters during training as well as the learning rate scheduler. The following code describes how to create the model in PyTorch[38] to load the parameters for the GE2E[33] loss function such that the Adam optimizer[46] can tune them as well, and the learning rate scheduler to adjust the learning rate during the training. The Adam optimizer is a widely used optimizer and is known to work well with LSTM networks.

```
model = LSTMModel(input_size, hidden_size, num_layers, projection_size)
w = nn.Parameter(torch.tensor(50.0))  # Initialize w as a learnable parameter
b = nn.Parameter(torch.tensor(-20.0))  # Initialize b as a learnable parameter
optimizer = torch.optim.Adam(list(model.parameters()) + [w, b],
                             lr=learning_rate)
scheduler = torch.optim.lr_scheduler
                        .ReduceLROnPlateau(optimizer, mode='min', factor=0.1,
                                           patience=3, threshold=0.01)
```

One small side note is that the initial values for w and b are different from the ones proposed by Wan et al. in “Generalized end-to-end loss for speaker verification”. As the amount of training could be shortened by roughly 10% with the values seen in the code above.

Training loop

Each training loop is called an epoch. For each epoch, the following sequence of steps is executed:

The epoch begins by initializing the epoch training loss to 0.0 and setting the model to training mode. The training data loader is then iterated over. For each batch from the data loader, segments and labels are moved to the specified device. The Adam[46] optimizer’s gradients are zeroed out, and embeddings are generated from the segments using the model. The GE2E loss is calculated and backpropagation is performed. Gradient clipping is applied to prevent exploding gradients, as this is a common problem with LSTM[1]

Networks, and the model parameters are updated by the optimizer. The current loss is added to the epoch training loss.

After working through all batches, the average epoch training loss is calculated and appended to the list of training losses for later visualization. The model is then set to evaluation mode, and the evaluation process begins. The evaluation data loader is iterated over, with segments and labels moved to the device for each batch. Embeddings are generated, and the GE2E[33] loss is calculated. The validation loss is once again averaged over all batches.

Once the evaluation is complete, the average validation loss is appended to the list of validation losses, for later visualization. The learning rate is updated based on this loss, by the learning rate scheduler. If the validation loss has improved significantly, the best loss is updated, the patience counter is reset, and the current model state is saved as the best model.

If the validation loss hasn't improved, the patience counter is incremented. Should the patience counter reach its limit, the training loop is stopped by the early stopping mechanism. This is then repeated until the max number of epochs is reached or the early stopping mechanism is triggered. Afterward, the best model is saved.

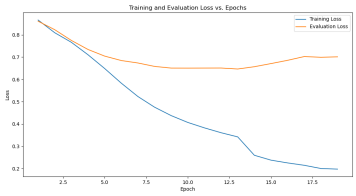


Figure 3.1: Loss without Overfitting

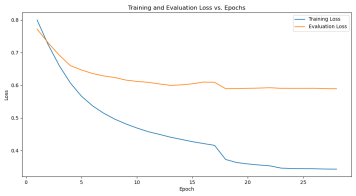


Figure 3.2: Loss with Overfitting

The early stopping mechanism[47] is quite an important part of training a neural net. As it prevents the model from overfitting, as can be seen in figure 3.1 and figure 3.2, when looking at the validation loss. We use the validation loss as overfitting can be seen better in this data as it is new data to the model but us from the same domain. This ensures that only the best model is saved after it has started to plateau and not learn anything important but rather starts to learn too much detail of the dataset, called overfitting. Without this mechanism, one would have to guess the correct number of epochs in advance, which is basically impossible.

3.4 Inference with a trained Model

Now that training has been implemented, the model can be exposed to new, unseen data for speaker diarization. The process begins with the same preprocessing and feature extraction steps as described in section 3.1.

Once the features are extracted, they are fed into the trained model, which we load into PyTorch[38]. Here, it is important to note that the parameters of the model class generated in python must be the same as the model one tries to load. Otherwise, the weights cannot be set correctly.

The next step is to then do the same post-processing as described in the chapter 3.1 to get the labeled segment. Now we can generate a visualization of diarization system using matplotlib[41] and return useable rttm[45] files with the help of pyannotate[44].

3.5 Clustering and Evaluation

As per the original paper from, Wang et al. [15] this implementation also uses Spectral Clustering[6] to group the d-vectors[5] together. But before we aggregate the d-vectors to 400ms segments. This is done to get rid of small outliers where there are 10 segments and 3 of them are a different cluster as the other 7 it is likely in such a small time frame that those are also part of the dominant speaker. Therefore, we L2 normalize the d-vectors of the segment and then average them to create the embedding of this segment. Not only does this improve the reliability of the system by removing outliers, we also significantly reduce the embeddings the clustering algorithm needs to cluster and therefore reducing computation time.[15]

Now that the embeddings are finished, we can start with the spectral clustering to generate the labels. The Spectral clustering consists of 5 steps as described in section 2.2.3. The Matrix calculations are done with the help of NumPy[40] and refinement operations like Gaussian filters are achieved with the library SciPy[43]. Then the transformed embeddings are clustered by the K-Means++[35] implementation from sklearn[42]. For the research questions also, the clustering algorithms DBSCAN[9] and Agglomerative Clustering[10] were implemented with the sklearn library.

After the clustering, the next step to clean up the data is to cluster together consecutive speaker data that has small missing segments. Those segments are most likely short speech pauses like taking a breath or building up suspense. To achieve this, we implemented the

function `group_segments()` that works by first taking all the segments and iterating over them to check the time gap between the segments if they are from the same speaker and then aggregates the consecutive segments until either the time difference is larger than a threshold, 0.2 seconds in this case, or the next segment is from a different speaker.

The final step in the implementation is to evaluate the performance of the model. The package pyannotate[44] thankfully already takes all the complicated math away by providing a few simple functions to calculate DER, JER, Identification Error Rate, etc[11]. The Data needs to be turned back into RTTM[45] files, which works exactly in reverse to the extraction described in section 3.1. With the completed data, we can now compare the model's performance and also visualize the generated labels with the help of the library matplotlib[41], against the ground truth as shown in figure 3.3 and 3.4.

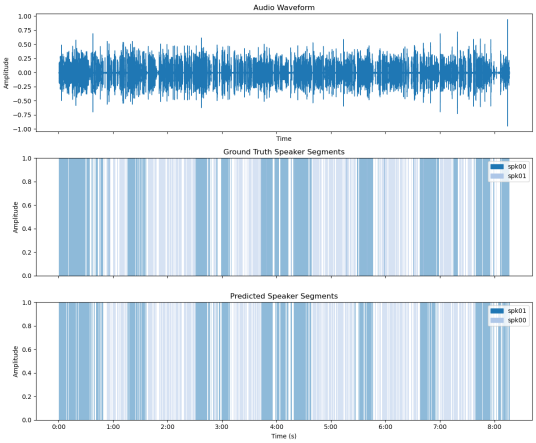


Figure 3.3: Diarization System Output Example with 2 Speakers

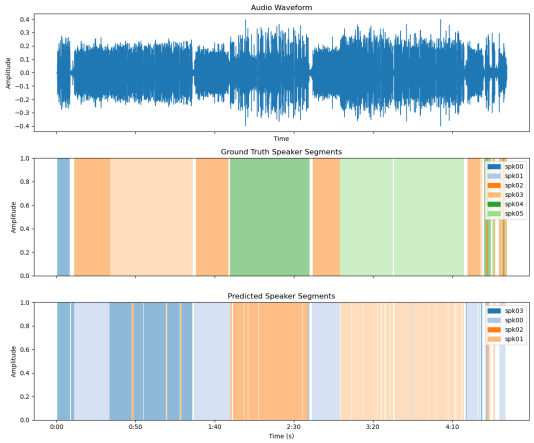


Figure 3.4: Diarization System Output Example with 6 Speakers

4 User Guide

This chapter will present how one can configure and use the implementation. There are 3 important files and a few files containing library functions such as feature extraction or diarization metrics calculations:

1. `train.py` contains the data preparation for training and the training loop.
2. `run.py` contains the data preparation for inference on the specified model and the visualizations, as well as statistical evaluation of the model performance.
3. `config.py` contains the variables to change the behavior of the script.

Before being able to run anything, one must first install all dependencies with `pip install -r requirements.txt`. This will install all libraries and dependencies the implementation uses. To run the training script, set the configurations explained below correctly and then use the command "`python train.py`". As for running the run script, the command "`python run.py path_to_model.pth`" can be used.

To configure the structure of the Neural Net, the variables `input_size`, `hidden_size`, `num_layers` and `projection_size` can be used as follows:

- `input_size, int`: Sets the shape of the input vector. If you change this value, also the resolution of the Mel Vector will change, as that is the input.
- `hidden_size, int`: Defines the number of nodes per hidden layer. This should be in the range of a few hundreds to maybe a few thousands. Otherwise, the Neural Net gets too large.
- `num_layers, int`: Defines the number of hidden layers of the Neural Net. This should not be too large as it will significantly increase the number of nodes of the Neural Network.
- `projection_size, int`: This value sets the output vector size of the Neural Network. Changing this will impact the performance of the clustering algorithm, as too small values will have insufficient information and too large values will significantly increase the computation needed for clustering.

Now that our RNN structure is set, we also need to configure the parameters for training:

- `num_epochs, int`: Defines the maximum amount of training loops. This can be set quite high, as the early stopping mechanism will prevent overfitting.
- `batch_size, int`: The number of data samples per batch.

- `learning_rate, float`: This is the initial learning rate that the learning rate scheduler uses as initial value.

Finally, we have a few variables to change the behavior of the data-batchers and data-samplers.

- `min_num_speakers, int`: This variable defines the minimum number of speakers per batch. This must not be set lower than 2 as the loss function needs at least two different speakers.
- `max_num_speakers, int`: Defines the maximum number of speakers. This makes sure that, in a large dataset with hundreds of speakers, the size of a batch can be smaller with enough data per speaker. If set too high, the maximum number of speakers in the dataset is used.
- `num_segments, int`: Specifies how many segments each batch should contain per speaker.

5 Experimental Results

For training all 20 different models with different parameters and Neural Net sizes, two models with three hidden layers and 768 nodes per layer were trained. One model had LSTM[1] nodes, and the other had BiLSTM[22] nodes. Both models were trained on the full Vox Celeb 1 dataset[24]. The LSTM model was used for testing the spectral clustering[6] parameter tuning as well as the comparison against the aforementioned BiLSTM model. The other 18 models were used for different testing purposes. Nine models were used for the Mel spectrogram[4] parameter testing, each with three hidden layers and 768 nodes per layer. The remaining nine models were used for testing different network structures. These 18 models only used a part of the full Vox Celeb 1 dataset[24] for training. 200 randomly selected speakers were used to reduce the amount of computation time, especially for training, as the full-sized models took nearly 20 hours each to train.

For evaluating each of the trained models, the Vox Converse dataset [12] was used, as it features real-world audio recording scenarios. Here, 10 randomly selected audio files, with the same seed for repeatability, were selected and put through the speaker diarization system explained in chapter 3 where only the model and the parameters that were tested changed. Thereafter, the average of one run is used for the comparison. However, the DER[11] is not the only metric that is important for comparing Speaker Diarization System, but for the purpose of this system it is more than sufficient. One thing to note about the results is that they are very dependent on the clustering algorithms, therefore depending on how well the spectral clustering was tuned, the result could vary. However, to reduce the amount of impact the clustering algorithms have on the experimental results for the questions about parameter tuning and Neural Net structures, the number of speakers was fixed as this is a common practice[15], but not for the comparison of the clustering algorithms.

For testing the performance of the best model, which is the model proposed by Wang et al. in “Speaker Diarization with LSTM”[15], with three hidden layers and 768 nodes per layer, the average DER over the whole Vox Converse Dataset[12] was 20%. While this is quite a bit worse than the current state-of-the-art[13, 14] it is still a good result considering the implementation was originally proposed in 2017, over 7 years ago.[15]

5.1 Different Parameters for Pre- and Post-processing

Important factors and parameters for training a Neural Net are the batch size, the number of epochs and the learning rate. As mentioned in 3.3 the training loop uses learning rate scheduling for adjusting the learning rate to an appropriate value and also early stopping[47] to detect plateaus and save the model before overfitting. Therefore, these values will not be tested here. For the batch size, after some experimenting, I found 64 to give the best results, but for more information on generating the batches see 3.1.

Spectral Clustering

For the spectral clustering[6], especially the parameters sigma and percentile are interesting. To figure out the best sigma and percentile combination, the following experiment was set up. We randomly select 3 audio files that we use for testing. The audio files are then put through the Speaker diarization system and averaged to get the DER[11] value for the specific setting. This is repeated through sigma 0.1 to sigma 17 with a step of 1 and percentile 70 to 99 with step size of 1.5.

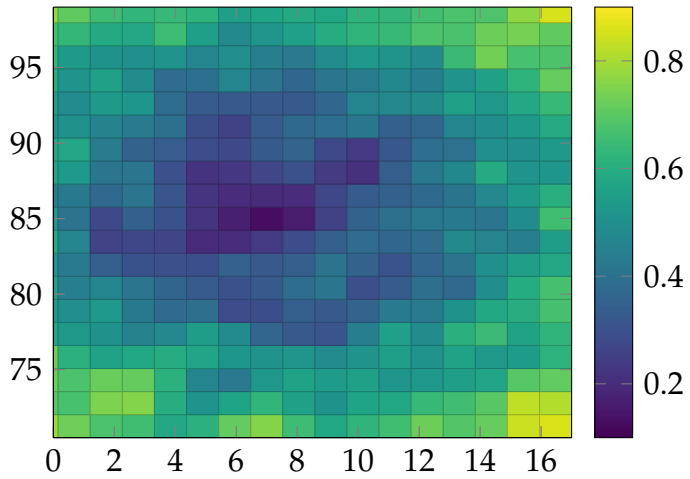


Figure 5.1: Heatmap of DER Result depending on σ and Percentile Configuration

We can clearly see an ideal data point and how increasing and decreasing both σ and percentile result in worse Diarization result in both directions.

With this, we can conclude that, for this specific model, a sigma of 7 and a percentile of 85% return the best results. Again, it is important to emphasize that this can vary depending on the training data as well as the Model Structure and even the evaluation set. Therefore, it is logical to customize the system for each particular task. This necessity arises because the

degree of blur is highly dependent on the distance between data points. For instance, if the speakers have very similar vocal characteristics, a smaller sigma, perhaps in the range of 0 to 1, would be more suitable. For example, in a scenario involving two female speakers with similar voice traits, a smaller sigma is better. On the other hand, in the case of a man and a woman with distinctly different voices, where men typically have lower frequencies, a larger sigma, possibly around 50, would likely result in better diarization.[6]

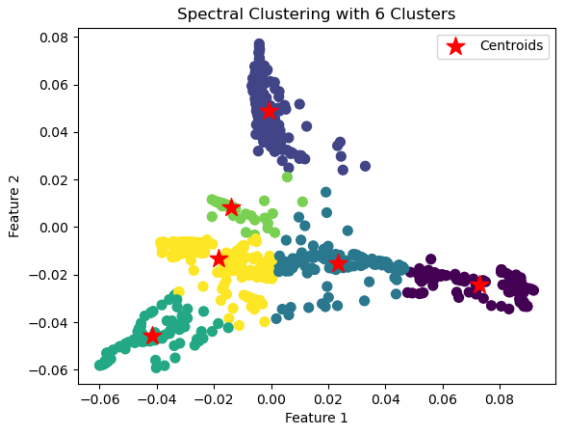


Figure 5.2: Spectral Clustering: Good Example

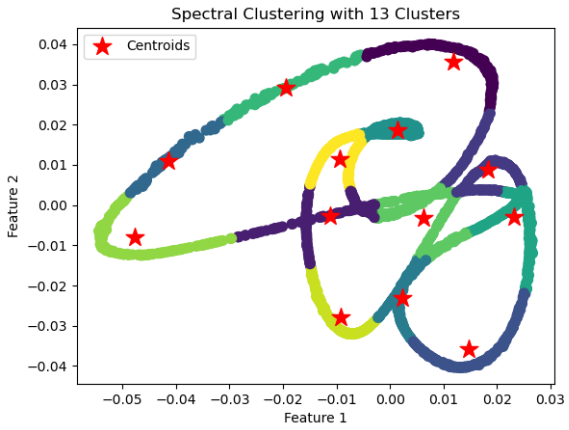


Figure 5.3: Spectral Clustering: Bad Example

Just to visualize the impact of those values, are two results, with the figure 5.2 begin the best tuned parameters and figure 5.3 the worst tuned parameters. We can clearly see that the number of speakers is wrong and that the data transformation created a bad scenario for the clustering such that it can not really create any useful data.

Mel Filter bank

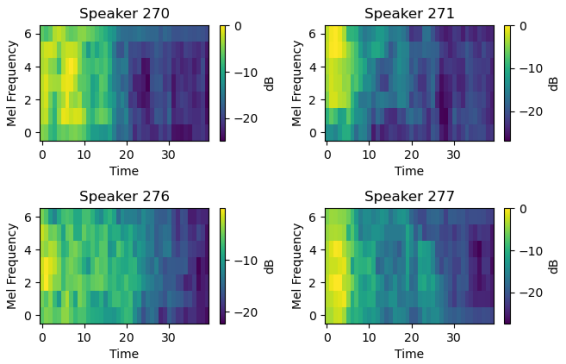


Figure 5.4: Mel Filter Banks with low temporal resolution

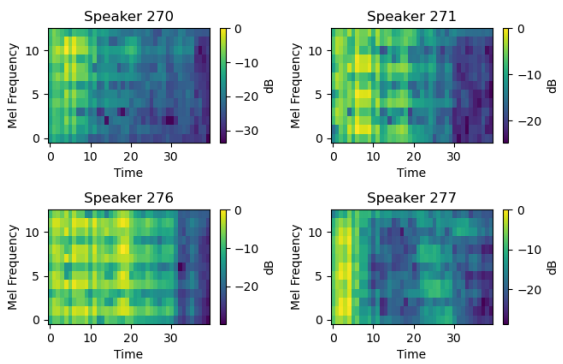


Figure 5.5: Mel Filter Banks with high temporal resolution

The feature extraction via Mel filter banks[4] is a crucial step for defining the temporal resolution and therefore the ability of the Neural Net to learn patterns. We can see this difference in the amount of differentiating patterns exist by comparing figure 5.4, that shows a hop length of 160 and 256 FFTs[20], with figure 5.5, that shows a picture with a hop length of 40 and 128 FFTs. For this experiment, the number of Mel dimension will be fixed to 40 as suggested by[15]. The test only changes the number of Fast-Fourier-Transforms and the hop length. One thing to note about the result is that we included a 512 FFT result to show that there is a large Problem with the small window size of 30ms that has to do with the small hop length resulting in too little data points for the number of FFT. Since there is not enough data, overlap occurs and results in a reduction of the pattern clarity that the LSTM model needs to be able to learn. That is why instead we looked at an even smaller hop length for the promising number of FFTs, with 128. But as we can once again see by the data in figure 5.6, with only 64 FFTs we also cannot train a model for 20, 80, and 160 hop length. And for a hop length of 40, while able to train, the model showed a poor performance.

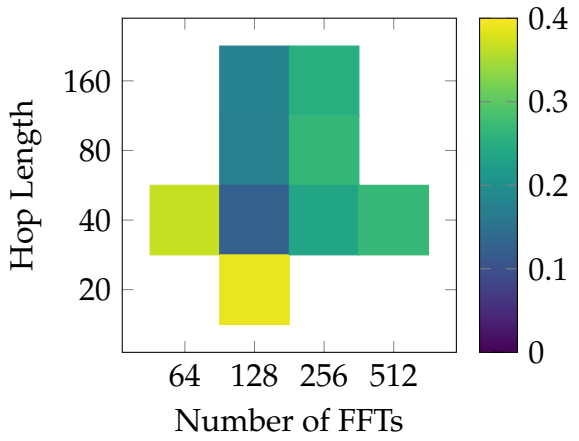


Figure 5.6: Heatmap: Number of FFTs against Hop Length

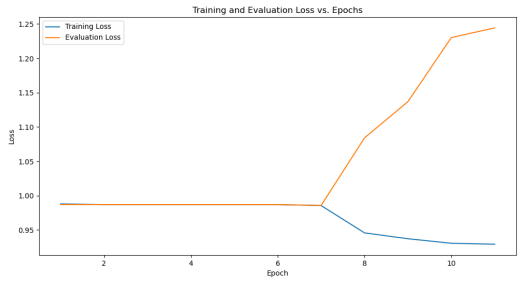


Figure 5.7: Loss with wrong Mel settings

The white squares in figure 5.6 indicate a combination where no data is available because the evaluation loss during training would either go up or plateaued much too early. This means that the these models were unable to learn any data and therefore did not undergo testing. This behavior can be seen in figure 5.7. With this result, we can conclude that the number of FFTs as well as the hop-length not only have a significant impact on the performance of the system but can also lead to non-functional systems. That is why it is crucial to set the correct values for the window size. So the results for a 30ms window size is a hop-length of 40 and 128 FFTs, as it yielded the best result by far.

5.2 Different clustering Methods

The clustering process is one of the most important steps to generate the correct number of speakers and cluster the d-vectors[5] to form correct speech segments. That is why we will take a closer look at the 4 most common clustering methods for this kind of data. One more important factor for the clustering algorithms is that they need to be able to work without knowing the number of clusters in advance. Therefore, we will take a look at Spectral Clustering[6], DBSCAN[9], offline K-Means[8, 35] and Agglomerative Clustering´[10]. For further information about the different algorithms, please refer to chapter 2 Material and Methods.

To compare the different clustering algorithms, we use the metrics discussed in chapter 2 plus a metric we introduce here, to evaluate how well each algorithm detects the number of speakers in the audio file, that we will call Relative Number of Speakers Error (RNSE). This metric is based on the relative difference [48] to the ground truth compared to the worst performing system. It is similar to the metric used by *In-domain Adaptation Solutions for the RTVE 2018 Diarization Challenge*[49] but with a slight change to make it relative to the worst performing system. Here's how it works: First, we calculate the average difference between the actual number of speakers (ground truth, N_{GT}) and the detected number of speakers (N_D) for each system. This difference, ΔN, is given by:

$$\Delta N = \frac{1}{T} \sum_{t=1}^{T} |N_{GT}(t) - N_D(t)| \tag{5.1}$$

Where T is the total number of time segments. By taking the absolute value, we can make sure there are no negative numbers. Next, check all the systems to find the one with the largest average difference, which we call ΔN_{max}. This represents the worst-performing system and is set as the 100% error mark. The RNSE for each system i is calculated by comparing their average difference ΔN_i to ΔN_{max}:

$$\text{RNSE}_i = \frac{\Delta N_i}{\Delta N_{max}} \times 100\% \tag{5.2}$$

This way, one can see how each system performs relative to the worst one, making it easier to compare their effectiveness in detecting the number of speakers.[49]

For the testing, we followed the methods described at the beginning of the chapter and used the same reduced dataset version of the 3 hidden layer and 768 nodes per layer used in the Neural Net architecture experiment. This was done to reduce the amount of time the experiments took. Therefore, the overall DER[11] values are worse, as can be seen in figure 5.8, this is also partly true because here we want to see how well it performs on dynamic number of speaker detection and therefore the Vox Converse dataset[12] had a range of 2 to 21 speakers which is a wider span than the randomly selected evaluation data in the other experiments. Here we also included a few more metrics[11] that overall show a similar result to DER, therefore for the analysis we will only focus on DER and RNSE.

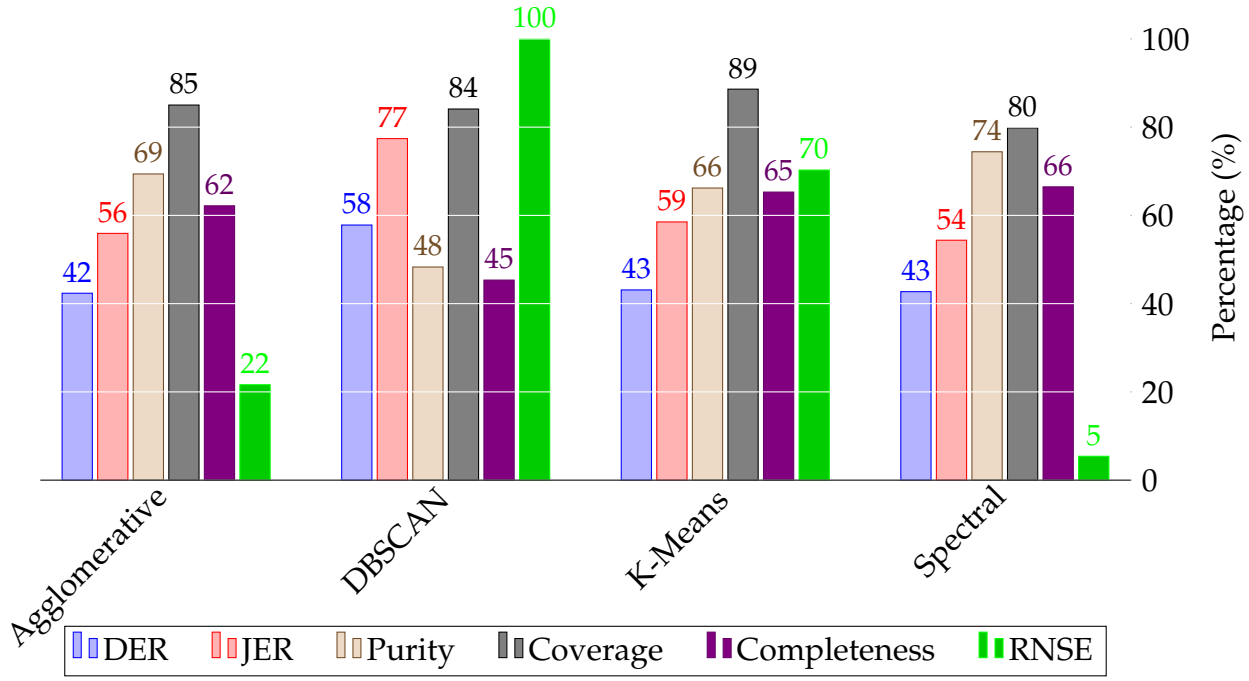


Figure 5.8: Clustering Algorithms Comparison

Overall, the data in figure 5.8 shows that DBSCAN performed the worst even though its abilities sounded promising, the amount of noise in the data causes it to combine clusters because there are a few outliers partially connecting them. This can be seen to an extent in figure 5.9. It is possible that the results could be improved by tuning the parameters better, but after tuning it to the best of our abilities, this was the best result.

The other three algorithms performed, within the margin of error, identically. But this is only true if we look at the overall DER result. When we take the difference in the number of speakers error into account, it paints a different picture. Namely, that k-means is significantly worse at finding the correct number of speakers, this can as well once again be seen in figure 5.9 where K-Means found 2 clusters instead of the 5 that Agglomerative

Clustering found. There we can also see the problem that spectral clustering, while good at identifying the correct number of clusters, is worse than Agglomerative Clustering as it found 6 clusters in figure 5.9.

Overall, Agglomerative clustering appears to have a slight edge over Spectral Clustering. Especially if we consider that Spectral clustering is so heavily depending on the correct parameter tuning. As for DBSCAN and K-Means Clustering, the d-vector data does not work well with their approach to generating clusters. One thing that might be interesting to consider, especially for DBSCAN is to use the same data refinement steps as spectral clustering uses, as described in 2.2.3.

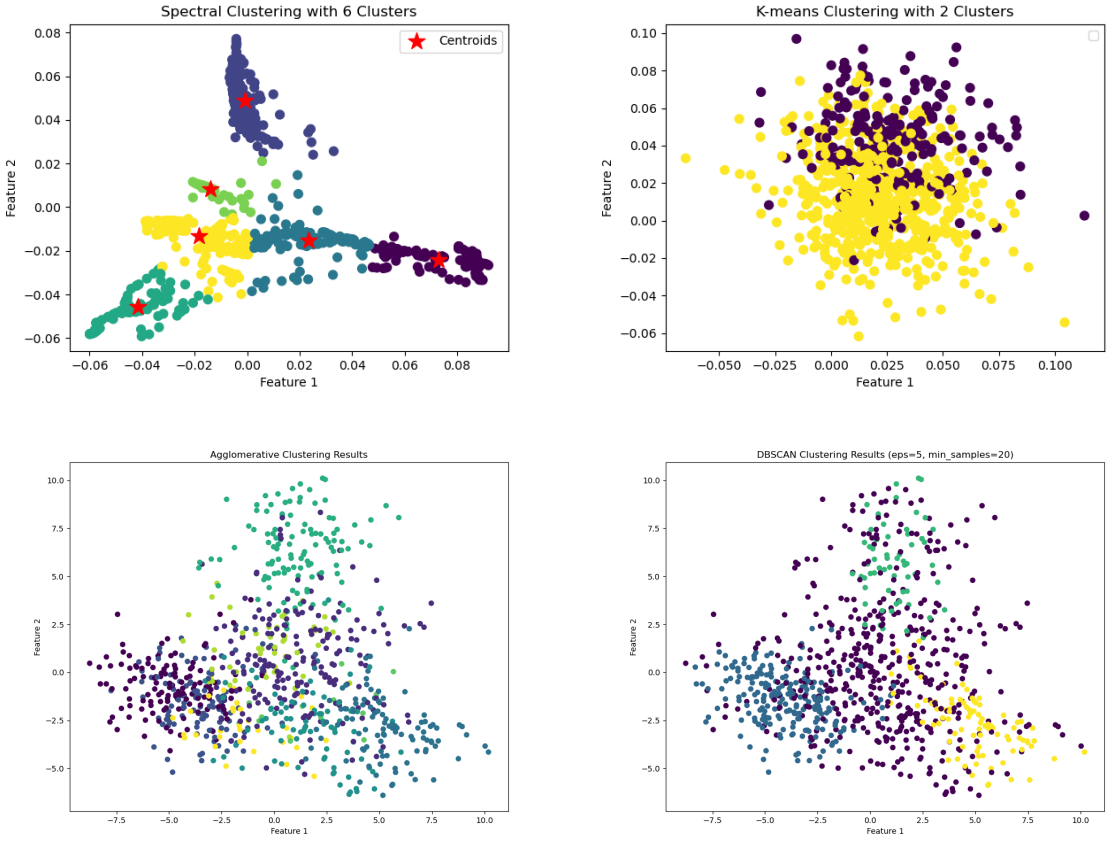


Figure 5.9: Result of four different Clustering Algorithms on the same Audio File

5.3 Different Neural Net Structures

BiLSTM vs LSTM

The idea of using BiLSTM[22] instead of LSTM[1] comes from their ability to understand more of the temporal space, as it can process data from the past and the future.[23]
For this experiment, we used 3 hidden layers and 768 nodes per layer. This means that both LSTM and BiLSTM networks have the same number of nodes. However, the BiLSTM network has twice the number of parameters, which in theory should give the network an advantage.[23] The results in table 5.1 show how each network performed on the same 10 random audio files in ascending order for the LSTM network and the average Diarization Error Rate in the last row.

DER of LSTM	DER of BiLSTM
10,15%	10,05%
10,29%	10,08%
15,22%	15,10%
16,68%	17,57%
20,39%	20,97%
26,44%	25,03%
26,74%	25,35%
27,03%	26,27%
38,96%	35,70%
42,88%	40,74%
23,48%	**22,69%**

Table 5.1: DER: BiLSTM vs LSTM

Even though the BiLSTM network should have a benefit, we can only see a minor, but measurable and very consistent, advantage to implementing BiLSTM over LSTM, in the data of table 5.1. It is also interesting to note that both performed quite similar per audio files, as both have a perfect ascending order. To conclude, the portion of precision gained isn't large enough to justify the addition of more parameters, thereby making the model larger and more computationally demanding.

Different Layer Size and Number of Layers

One of the great benefits of using fewer layers and fewer nodes per layer is that we can train the model much faster, and also running the speaker diarization with smaller models is much faster. But on the other side, we lose accuracy with more lightweight models. To get the best tradeoff between performance and accuracy, we will look at 9 different models with 2, 3 and 4 hidden layers as well as 192, 384 and 768 nodes per layer. The result of the experiment can be seen in table 5.2 where the name of each column is the number of layers and nodes per layer and each row is the DER of that model for one of the ten randomly selected audio files. Where the base model structure from "Speaker Diarization with LSTM"[15] with 3 hidden layers and 768 nodes per layer is sorted in ascending order.

Then in the last row, the average is calculated. This average of each model is visualized as a heatmap in figure 5.10.

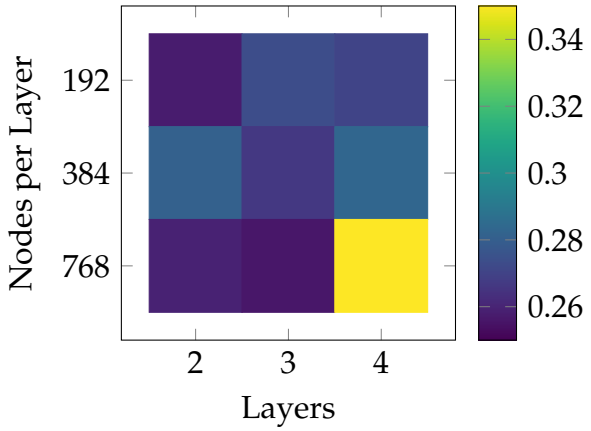


Figure 5.10: Heatmap: Number of Layers against Nodes per Layer

The best performance was made by the proposed configuration by Wang et al.[15] as can be seen in table 5.2 where the best average DER is in the column of 3 layers and 768 nodes per layer. However, the second-best performance, that is on the same level, within the margin of error, was the smallest one. Which has only one sixth of the nodes with 2 layers and 192 nodes per layer. It is important to see that a model which is too large, in this case with 4 layers and 768 nodes per layer, not only is a lot larger and therefore slower, for training and running. This is because it has to do more calculations. It also runs into the problem that it can learn a lot of the noise in the training data. Therefore, it performed unfortunately in the out of domain evaluation data[12]. While the 2 layer 192 nodes model was as fast as the 3 layer 768 nodes model, the large of the two is overall more reliable, as can be seen by the smaller variance in table 5.2. Overall, the data in figure 5.10 and table 5.2 show that 2 and 3 layer models performed pretty good. But with 4 hidden layers, we start to see that the models begin to perform worse. This is either because the models start to learn too much of the noise instead of the patterns of a speaker or because of the smaller dataset where not able to fully learn to differentiate speakers.

While for our experimentation the number of nodes per layer showed a smaller impact, too many nodes per layer can increase the risk of overfitting and with too few nodes per layer the model does not have enough capabilities for learning the patterns in the data.

Therefore, it is important to always design the Neural Net for the specific dataset and usage, as with a smaller model one might sacrifice accuracy but gain speed in training and running the model. So it boils down to the requirements as to what number of layers and nodes will yield the best results.

2 192	**2 384**	**2 768**	**3 192**	**3 384**	**3 768**	**4 192**	**4 384**	**4 768**
10,15%	10,15%	10,15%	10,15%	10,15%	10,15%	10,15%	10,15%	10,19%
10,29%	10,29%	10,29%	10,29%	10,29%	10,29%	10,29%	10,29%	10,29%
15,44%	15,96%	16,35%	14,99%	15,79%	15,25%	15,27%	15,40%	17,71%
19,71%	41,55%	20,20%	19,97%	19,59%	19,71%	20,02%	20,40%	53,47%
16,28%	16,18%	16,18%	16,28%	16,95%	23,86%	16,95%	16,95%	22,85%
27,03%	27,03%	27,03%	27,03%	27,03%	27,03%	27,03%	27,03%	27,03%
25,74%	26,76%	27,85%	29,45%	32,47%	32,07%	33,62%	33,51%	36,31%
45,67%	44,32%	49,30%	58,25%	44,35%	38,07%	55,35%	62,56%	64,75%
26,91%	30,29%	25,62%	31,05%	26,06%	38,96%	26,97%	26,65%	43,15%
60,30%	57,99%	56,33%	55,67%	63,60%	40,41%	54,44%	59,70%	66,17%
25,75%	**28,05%**	**25,93%**	**27,31%**	**26,63%**	**25,58%**	**27,01%**	**28,27%**	**35,19%**

Table 5.2: DER per Number of Layers and Nodes per Layer test data

6 Alternative Methods

Now that the implementation based on d-vectors[5] generated by LSTM-Neural-Networks[1] has been thoroughly evaluated and tested, even though there are still more aspects to experiment on, this chapter now takes a look at three alternate approaches for speaker diarization systems. We will present older, but proofed technologies like the i-vector[50] newer advances in the d-vector[5] based approach the x-vector[51] as well as take a look at a recent approach with a different idea the Graph Neural Network (GNN)[52].

6.1 I-Vector

While d-vectors have gained popularity in recent speaker diarization systems, the use of i-vectors remains a viable and well-established alternative. I-vectors, or identity vectors, were introduced as a compact representation of speaker characteristics, derived from a factor analysis framework [50]. This approach represents the speaker features in a relatively low-dimensional space, in the range of 100 to 600 dimensions. I-vectors are robust and show good performance due to their ability to capture long-term speaker features effectively.

One advantage of i-vectors over d-vectors is their ability to handle longer speech segments with less computational resources. Since they are designed to work well with utterances of varying lengths, they are particularly well suited for speaker diarization where speech segments can vary significantly in duration. Additionally, i-vectors generally perform better with less training data than d-vectors, which can be beneficial when working with limited datasets or resources for training. They are also quite well established and a lot of optimization has been performed to improve their performance even further.[18]

But there are also downsides to i-vectors[50] when compared to d-vectors. Firstly, i-vectors typically require a Universal Background Model (UBM) and a Total Variability Matrix, which both need large and diverse datasets[53]. This pre-training process can be computationally intensive and may not always generalize well to significantly different acoustic conditions. Secondly, i-vectors may not capture fine-grained temporal dynamics as effectively as d-vectors, which are often extracted from Neural Networks designed to model sequential data, like LSTMs. This can be a disadvantage where short-term speaker characteristics are important to find discrimination. Lastly, while i-vectors have good performance, recent advancements in deep learning helped d-vectors achieve state-of-the-art speaker diarization results in many speaker recognition tasks.[18]

6.2 X-Vector

X-vectors[51] are quite a recent development in the realm of speaker recognition. They can be considered a bridge between traditional i-vectors and newer deep learning-based methods such as d-vectors. They basically serve as a new generational development of the d-vector, as x-vectors use a Time Delay Neural Network (TDNN)[54] structure to capture both frame-level and segment-level speaker features. This strategy enables x-vectors to effectively model the temporal dynamics of speech while generating a fixed-dimensional embedding, regardless of the input utterance length. Therefore, trying to combine the best of both.

One advantage of x-vectors over d-vectors is their enhanced capability to capture long-range temporal speaker characteristics. The TDNN architecture enables the x-vectors to consider a broader context when extracting speaker features, potentially resulting in a more discriminative speaker representations. Moreover, x-vectors have shown an increased performance in for short utterances compared to i-vectors, making them well-suited for speaker diarization tasks with shorter speech segments. The x-vector also allows for data augmentation techniques during training, therefore making it more robust against bad acoustic conditions such as noise.[18]

Nevertheless, x-vectors do come with certain downsides. They also need a lot of training data, just like d-vectors. This also implies the need of large number of resources for training a model. While x-vectors have shown their capability to achieve state-of-the-art performance in numerous speaker recognition tasks, including diarization, they are more complex and resource intensive than d-vetor. However, this might improve over time as x-vectors see more optimizations in the future.[55]

6.3 Graph Neural Net (GNN)

Graph Neural Networks (GNNs) are becoming a good choice instead of LSTM-based d-vectors for speaker diarization tasks [52]. LSTM-based d-vectors process speech like a straight line, but GNNs make the diarization problem into a graph. In this graph, each node is usually a speech segment, and edges show how these segments connect. This way of using graphs helps to model complex relationships between speech parts, which might find more detailed speaker features and how they interact.

GNNs have some good points compared to LSTM-based d-vectors. They can model global dependencies better. LSTMs sometimes have trouble with long-range dependencies because they work in order, but GNNs can connect far apart speech segments if they are similar. This big picture view can make speaker clustering more accurate, especially when there are many speakers talking. Furthermore, GNNs work well with inputs of different lengths because the graph structure can easily handle speech segments that are short or long. The graph representation is flexible, so we can add extra information like metadata or context as features on nodes or edges, which might improve diarization.[18]

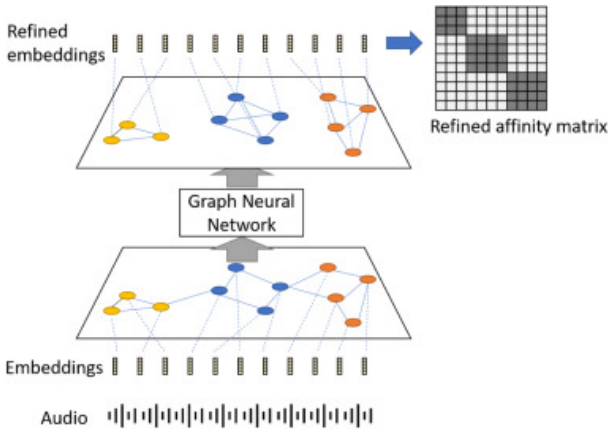


Figure 6.1: Graph Neural Net for Speaker Diarization[18]

But GNNs for speaker diarization also have some difficulties. Making the first graph is essential and can change how well it works. The best way to make the graph might be different for different diarization tasks or datasets. GNNs also need more complicated training than LSTM models, which means they need more computer power, and it's harder to find the best settings. Even though GNNs show good results in many speaker diarization situations, they might not always be better than well-tuned LSTM-based d-vector systems[18]. Because GNNs are newer, there is still a lot of research needed to definitively say if they outperform LSTM-Based systems.

7 Conclusion and Outlook

7.1 Conclusion

This thesis explored the optimization of Long Short-Term Memory (LSTM) Neural Networks for speaker diarization, focusing on three main research questions:

1. The impact of hyperparameters on system performance, particularly in spectral clustering and Mel filter bank calculations.
2. The effectiveness of different clustering methods in speaker diarization.
3. The influence of Neural Network structure on diarization accuracy.

Our findings demonstrate that:

1. Hyperparameters significantly affect performance:
   - Optimal spectral clustering results were achieved with $\sigma = 7$ and percentile = 85%.
   - Mel filter bank settings showed best results with a hop-length of 40ms and 128 FFTs.
2. Clustering method comparison:
   - Spectral clustering outperformed other methods in detecting the correct number of speakers.
   - Agglomerative clustering showed promise in overall performance metrics.
3. Neural network structure:
   - BiLSTM showed a slight advantage over LSTM, but the performance gain may not justify the increased computational cost.
   - A balance between model size and performance was found with two architectures:
     - 2 layers and 192 nodes: While small, able to handle speaker diarization task quite well, but had a larger variance.
     - 3 layers and 768 nodes: This architecture showed a slightly better and more robust performance.

These findings highlight the importance of hyperparameter tuning. The comparison of clustering methods suggests that a hybrid approach, combining the strengths of Spectral and Agglomerative Clustering, could be a promising direction for future research. Additionally,

our analysis of Neural Network structures provides valuable insights for balancing model complexity with performance.

Furthermore, the system's overall performance resulted in an average DER of 20% over the Vox Converse dataset[12]. his performance, while promising, falls short of the current state-of-the-art range of 6-9%[13, 14]. This makes sense because the system was originally proposed over seven years ago in 2017 and the speaker diarization is a very active field of research.

These results provide valuable insights into optimizing LSTM-based speaker diarization systems, offering a foundation for future improvements in the field.

7.2 Outlook

While this research aims to help improve the understanding of how to optimize LSTM-based speaker diarization, several avenues for future research and improvement remain:

- Short-Term improvements:
    1. **Enhanced Voice Activity Detection (VAD):** Implementing a more sophisticated VAD system could improve the quality of input data, potentially leading to better diarization results.
    2. **Hybrid Approaches:** Investigating combinations of different techniques, such as using spectral clustering refinement steps with Agglomerative Clustering, could yield improved results.
    3. **Cross-gender Performance Analysis:** Studying how models trained on one gender perform on another could provide insights into gender-specific speech characteristics and improve overall system robustness.
- Long-term research directions
    1. **Multi-language Support:** Extending the system to support multiple languages could broaden its applicability and robustness.
    2. **Advanced LSTM Variants:** Exploring more recent LSTM variants like sLSTM, mLSTM, or xLSTM could potentially improve performance further.[56]
    3. **Real-time Processing:** Adapting the system for real-time diarization could open up new applications in live transcription and meeting analysis.

By pursuing these directions, future research can build upon the foundations laid in this thesis, potentially leading to more accurate, efficient, and versatile speaker diarization systems.

8 References

[1] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9.8 (Nov. 1997), pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. eprint: https://direct.mit.edu/neco/article-pdf/9/8/1735/813796/neco.1997.9.8.1735.pdf. URL: https://doi.org/10.1162/neco.1997.9.8.1735 (cit. on pp. i, ii, 1, 2, 5, 6, 16, 22, 29, 32).

[2] Wikipedia. *Speaker diarisation — Wikipedia, The Free Encyclopedia*. http://en.wikipedia.org/w/index.php?title=Speaker%20diarisation&oldid=1195112526. [Online; accessed 01-August-2024]. 2024 (cit. on pp. i, ii, 1).

[3] Beth Logan. "Mel Frequency Cepstral Coefficients for Music Modeling". In: *Proc. 1st Int. Symposium Music Information Retrieval* (Nov. 2000) (cit. on pp. i, ii, 4).

[4] Laxmi Narayana M. and Sunil Kumar Kopparapu. "Choice of Mel Filter Bank in Computing MFCC of a Resampled Speech". In: *CoRR* abs/1410.6903 (2014). arXiv: 1410.6903. URL: http://arxiv.org/abs/1410.6903 (cit. on pp. i, ii, 2, 4, 22, 25).

[5] Ehsan Variani et al. "Deep neural networks for small footprint text-dependent speaker verification". In: *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2014, pp. 4052–4056. DOI: 10.1109/ICASSP.2014.6854363 (cit. on pp. i, ii, 1, 18, 26, 32).

[6] Huazhong Ning et al. "A spectral clustering approach to speaker diarization." In: *Interspeech*. Citeseer. 2006 (cit. on pp. i, ii, 2, 3, 7, 18, 22, 23, 24, 26).

[7] J. MacQueen. *Some methods for classification and analysis of multivariate observations*. English. Proc. 5th Berkeley Symp. Math. Stat. Probab., Univ. Calif. 1965/66, 1, 281-297 (1967). 1967 (cit. on pp. i, ii, 2, 3, 9).

[8] Purnima Bholowalia and Arvind Kumar. "EBK-means: A clustering technique based on elbow method and k-means in WSN". In: *International Journal of Computer Applications* 105.9 (2014) (cit. on pp. i, ii, 2, 9, 26).

[9] Dingsheng Deng. "DBSCAN Clustering Algorithm Based on Density". In: *2020 7th International Forum on Electrical Engineering and Automation (IFEEA)*. 2020, pp. 949–953. DOI: 10.1109/IFEEA51475.2020.00199 (cit. on pp. i, ii, 2, 3, 9, 10, 18, 26).

[10] Frank Nielsen. "Hierarchical Clustering". In: Feb. 2016, pp. 195–211. ISBN: 978-3-319-21902-8. DOI: 10.1007/978-3-319-21903-5_8 (cit. on pp. i, ii, 2, 10, 18, 26).

[11] Hervé Bredin. "pyannote. metrics: A Toolkit for Reproducible Evaluation, Diagnostic, and Error Analysis of Speaker Diarization Systems." In: *INTERSPEECH*. 2017, pp. 3587–3591 (cit. on pp. i, ii, 3, 10, 11, 19, 22, 23, 27).

[12] Joon Son Chung et al. "Spot the conversation: speaker diarisation in the wild". In: *INTERSPEECH*. 2020 (cit. on pp. i, ii, 3, 12, 22, 27, 30, 36).

[13] Reza (Pouya) Rostam. *Picovoice/speaker-diarization-benchmark*. Nov. 29, 2023. URL: https://github.com/Picovoice/speaker-diarization-benchmark (cit. on pp. i, ii, 22, 36).

[14] Jaesung Huh et al. *VoxSRC 2022: The Fourth VoxCeleb Speaker Recognition Challenge*. 2023. arXiv: 2302.10248 [cs.SD]. URL: https://arxiv.org/abs/2302.10248 (cit. on pp. i, ii, 22, 36).

[15] Quan Wang et al. "Speaker Diarization with LSTM". In: *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2018, pp. 5239–5243. DOI: 10.1109/ICASSP.2018.8462628 (cit. on pp. i, ii, 1, 3, 6, 12, 13, 18, 22, 25, 29, 30).

[16] Hanwu Sun et al. "Speaker diarization system for RT07 and RT09 meeting room audio". In: *2010 IEEE International Conference on Acoustics, Speech and Signal Processing*. 2010, pp. 4982–4985. DOI: 10.1109/ICASSP.2010.5495077 (cit. on p. 1).

[17] Sergey Salishev et al. "Voice Activity Detector (VAD) Based on Long-Term Mel Frequency Band Features". In: *Text, Speech, and Dialogue*. Ed. by Petr Sojka et al. Cham: Springer International Publishing, 2016, pp. 352–358. ISBN: 978-3-319-45510-5 (cit. on pp. 1, 4, 12, 13).

[18] Tae Park et al. "A review of speaker diarization: Recent advances with deep learning". In: *Computer Speech and Language* 72 (Nov. 2021), p. 101317. DOI: 10.1016/j.csl.2021.101317 (cit. on pp. 1, 32, 33, 34).

[19] Md Sahidullah et al. *The Speed Submission to DIHARD II: Contributions and Lessons Learned*. 2019. arXiv: 1911.02388 [eess.AS]. URL: https://arxiv.org/abs/1911.02388 (cit. on p. 1).

[20] Wikipedia. *Fast Fourier transform — Wikipedia, The Free Encyclopedia*. http://en.wikipedia.org/w/index.php?title=Fast%20Fourier%20transform&oldid=1236413825. [Online; accessed 02-August-2024]. 2024 (cit. on pp. 2, 25).

[21] Wikipedia. *Recurrent neural network — Wikipedia, The Free Encyclopedia*. http://en.wikipedia.org/w/index.php?title=Recurrent%20neural%20network&oldid=1238141724. [Online; accessed 02-August-2024]. 2024 (cit. on pp. 2, 5).

[22] Alex Graves, Santiago Fernández, and Jürgen Schmidhuber. "Bidirectional LSTM Networks for Improved Phoneme Classification and Recognition". In: *Artificial Neural Networks: Formal Models and Their Applications – ICANN 2005*. Ed. by Włodzisław Duch et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 799–804. ISBN: 978-3-540-28756-8 (cit. on pp. 2, 6, 22, 29).

[23] Sima Siami-Namini, Neda Tavakoli, and Akbar Siami Namin. "The performance of LSTM and BiLSTM in forecasting time series". In: *2019 IEEE International conference on big data (Big Data)*. IEEE. 2019, pp. 3285–3292 (cit. on pp. 2, 6, 29).

[24] A. Nagrani, J. S. Chung, and A. Zisserman. "VoxCeleb: a large-scale speaker identification dataset". In: *INTERSPEECH*. 2017 (cit. on pp. 3, 12, 22).

[25] Alexandra Canavan, David Graff, and George Zipperlen. *CALLHOME american english speech corpus ldc97s42*. 1997. DOI: 10.21415/T5KP54 (cit. on pp. 3, 12).

[26] Wikipedia. *Mel — Wikipedia, The Free Encyclopedia*. http://de.wikipedia.org/w/index.php?title=Mel&oldid=236843024. [Online; accessed 20-July-2024]. 2024 (cit. on p. 4).

[27] Wikipedia. *Voice activity detection — Wikipedia, The Free Encyclopedia*. http://en.wikipedia.org/w/index.php?title=Voice%20activity%20detection&oldid=1219468211. [Online; accessed 04-August-2024]. 2024 (cit. on p. 4).

[28] D.A. Reynolds and Richard Rose. "Robust text-independent speaker identification using Gaussian Mixture speaker models". In: *Speech and Audio Processing, IEEE Transactions on* 3 (Feb. 1995), pp. 72–83. DOI: 10.1109/89.365379 (cit. on p. 4).

[29] Wikipedia. *Neural network (machine learning) — Wikipedia, The Free Encyclopedia*. http://en.wikipedia.org/w/index.php?title=Neural%20network%20(machine%20learning)&oldid=1237652724. [Online; accessed 04-August-2024]. 2024 (cit. on p. 5).

[30] Wenpeng Yin et al. "Comparative study of CNN and RNN for natural language processing". In: *arXiv preprint arXiv:1702.01923* (2017) (cit. on p. 5).

[31] Kyunghyun Cho et al. "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation". In: *CoRR* abs/1406.1078 (2014). arXiv: 1406.1078. URL: http://arxiv.org/abs/1406.1078 (cit. on p. 5).

[32] Ci Lin, Tet Yeap, and Iluju Kiringa. "Stacked Bidirectional LSTM for Predicting Emission of Nitrous Oxide". In: *Proceedings of the Canadian Conference on Artificial Intelligence* (May 2022). https://caiac.pubpub.org/pub/5inot0i4 (cit. on p. 6).

[33] Li Wan et al. “Generalized end-to-end loss for speaker verification”. In: *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2018, pp. 4879–4883 (cit. on pp. 6, 7, 14, 16, 17).

[34] Wikipedia. *Cluster analysis — Wikipedia, The Free Encyclopedia*. http://en.wikipedia.org/w/index.php?title=Cluster%20analysis&oldid=1234188906. [Online; accessed 04-August-2024]. 2024 (cit. on p. 7).

[35] David Arthur and Sergei Vassilvitskii. “k-means++: the advantages of careful seeding”. In: *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*. SODA ’07. New Orleans, Louisiana: Society for Industrial and Applied Mathematics, 2007, pp. 1027–1035. ISBN: 9780898716245 (cit. on pp. 9, 18, 26).

[36] Wikipedia. *Dendrogram — Wikipedia, The Free Encyclopedia*. http://en.wikipedia.org/w/index.php?title=Dendrogram&oldid=1195077958. [Online; accessed 30-July-2024]. 2024 (cit. on p. 10).

[37] Neville Ryant et al. *Third DIHARD Challenge Evaluation Plan*. 2020. arXiv: 2006.05815 [eess.AS]. URL: https://arxiv.org/abs/2006.05815 (cit. on p. 11).

[38] Adam Paszke et al. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf (cit. on pp. 12, 13, 15, 16, 18).

[39] Brian McFee et al. *librosa/librosa: 0.10.2.post1*. Version 0.10.2.post1. May 2024. DOI: 10.5281/zenodo.11192913. URL: https://doi.org/10.5281/zenodo.11192913 (cit. on p. 12).

[40] Charles R. Harris et al. “Array programming with NumPy”. In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2 (cit. on pp. 12, 14, 18).

[41] J. D. Hunter. “Matplotlib: A 2D graphics environment”. In: *Computing in Science & Engineering* 9.3 (2007), pp. 90–95. DOI: 10.1109/MCSE.2007.55 (cit. on pp. 12, 18, 19).

[42] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830 (cit. on pp. 12, 18).

[43] Pauli Virtanen et al. “SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python”. In: *Nature Methods* 17 (2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2 (cit. on pp. 12, 18).

[44] Hervé Bredin. “pyannote.audio 2.1 speaker diarization pipeline: principle, benchmark, and recipe”. In: *Proc. INTERSPEECH 2023*. 2023 (cit. on pp. 12, 18, 19).

[45] Neville Ryant et al. "First DIHARD challenge evaluation plan". In: *tech. Rep.* Linguistic Data Consortium, University of Pennsylvania, 2018 (cit. on pp. 12, 18, 19).

[46] Zijun Zhang. "Improved Adam Optimizer for Deep Neural Networks". In: *2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS)*. 2018, pp. 1–2. DOI: 10.1109/IWQoS.2018.8624183 (cit. on p. 16).

[47] Lutz Prechelt. "Early stopping-but when?" In: *Neural Networks: Tricks of the trade*. Springer, 2002, pp. 55–69 (cit. on pp. 17, 23).

[48] Wikipedia. *Relative change — Wikipedia, The Free Encyclopedia*. http://en.wikipedia.org/w/index.php?title=Relative%20change&oldid=1232875376. [Online; accessed 01-August-2024]. 2024 (cit. on p. 26).

[49] Ignacio Viñals et al. *In-domain Adaptation Solutions for the RTVE 2018 Diarization Challenge*. Nov. 2018. DOI: 10.21437/IberSPEECH.2018-45 (cit. on p. 26).

[50] Najim Dehak et al. "Support vector machines versus fast scoring in the low-dimensional total variability space for speaker verification". In: *Tenth Annual conference of the international speech communication association*. 2009 (cit. on p. 32).

[51] David Snyder et al. "X-Vectors: Robust DNN Embeddings for Speaker Recognition". In: *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2018, pp. 5329–5333. DOI: 10.1109/ICASSP.2018.8461375 (cit. on pp. 32, 33).

[52] Jixuan Wang et al. "Speaker Diarization with Session-Level Speaker Embedding Refinement Using Graph Neural Networks". In: *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2020, pp. 7109–7113. DOI: 10.1109/ICASSP40776.2020.9054176 (cit. on pp. 32, 33).

[53] Daniel Povey, Stephen Chu, and Balakrishnan Varadarajan. *Universal background model based speech recognition*. May 2008. DOI: 10.1109/ICASSP.2008.4518671 (cit. on p. 32).

[54] A. Waibel et al. "Phoneme recognition using time-delay neural networks". In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 37.3 (1989), pp. 328–339. DOI: 10.1109/29.21701 (cit. on p. 33).

[55] Mireia Diez et al. "Optimizing Bayesian Hmm Based X-Vector Clustering for the Second Dihard Speech Diarization Challenge". In: *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2020, pp. 6519–6523. DOI: 10.1109/ICASSP40776.2020.9053982 (cit. on p. 33).

[56] Maximilian Beck et al. *xLSTM: Extended Long Short-Term Memory*. 2024. arXiv: 2405.04517 [cs.LG]. URL: https://arxiv.org/abs/2405.04517 (cit. on p. 36).

List of Figures

List of Tables