Zurich University
of Applied Sciences

School of
Engineering
InIT Institute of Applied
Information Technology

Bachelor Thesis (Computer Science)

Benchmarking of Classical and Deep Learning Speaker Clustering Approaches

Authors	Jan Sonderegger Patrick Walter
Supervisor	Thilo Stadelmann
Date	07.06.2019

Zusammenfassung

Speaker-Clustering beschreibt das Gruppieren von mehreren gesprochenen Segmenten zu verschiedenen Sprechern, ohne die Anzahl der Sprecher oder die einzelnen Sprecher selbst zu kennen. In den vergangenen Jahren wurden dafür am Institut für angewandte Informationstechnologie der ZHAW mehrere Ansätze basierend auf Deep Learning entwickelt. Die erzielten Resultate dieser Ansätze waren vielversprechend, jedoch haben sich kleinere Unterschiede im Evaluationsprozess der jeweiligen Ansätze eingeschlichen. Aus diesem Grund waren die Resultate nicht komplett miteinander vergleichbar.

Das Ziel dieser Arbeit ist, die drei Ansätze "Luvo", "Pairwise_Kldiv" und "Pairwise_Lstm" zu vergleichen. Um dies zu tun, müssen die Ansätze unter gleichen Voraussetzungen evaluiert werden um den vielversprechendsten Ansatz bestimmen zu können. Zusätzlich sollen die Deep Learning Ansätze mit klassischen Speaker-Clustering-Ansätzen verglichen werden, um die erzielten Resultate besser einordnen zu können.

Um dieses Ziel zu erreichen haben wir ein Benchmarking-System mit einem verbesserten Experimentaufbau entwickelt. Damit können wir vergleichbare und sinnvolle Resultate garantieren, indem wir bestehende Unterschiede der Ansätze eliminieren. Zusätzlich haben wir ein GMM- und I-Vector-Ansatz in das System eingebaut. Die Resultate dieser beiden klassischen Speaker-Clustering-Ansätze dienen als Referenzwerte, welche es zu übertreffen gilt. Um aussagekräftige Resultate zu bekommen, haben wir vier Metriken eingeführt aus welchen sich verschiedene Rückschlüsse aus den Resultaten ziehen lassen.

Die Arbeit konnte die Deep Learning Ansätze nicht abschliessend vergleichen. Die durchgeführten Experimente deuten an, dass die Ansätze optimiert werden müssen, um ihr Potential im neuen Experimentaufbau ausschöpfen zu können. Ausserdem hat sich gezeigt, das der Pairwise_Lstm Ansatz optimiert wurde, in dem die Inputdaten in kleinere Segmente unterteilt werden. Dies könnte auch für Luvo und Pairwise_Kldiv eine lohnenswerte Verbesserung sein und müsste für einen aussagekräftigen Vergleich untersucht werden.

Abstract

Speaker clustering describes the task of grouping multiple spoken samples by the speakers without previously knowing the speakers or even how many different speakers there are. In the past few years multiple approaches based on deep learning have been developed at the ZHAW Institute of Applied Information Technology (InIT). The results of these approaches were promising but there were small differences in the evaluation processes. Therefore the results were not fully comparable between each other.

The goal of this work is to benchmark three deep learning approaches called Luvo, Pairwise_Kldiv and Pairwise_Lstm. To do this, the approaches have to be evaluated under the same circumstances in order to find the most promising approach. Additionally, it should be possible to compare the deep learning approaches to classical speaker clustering approaches to better classify the achieved results.

To achieve this goal we built a benchmarking system with an improved experimental setup to guarantee comparable and meaningful results by eliminating the differences in the evaluation processes. Additionally, we implemented a GMM and I-Vector approach into the benchmarking system. The results of these two classical speaker clustering approaches are used as the baseline the deep learning approaches should outperform. To obtain meaningful results we defined four metrics that each provide a different insight regarding the quality of the results.

This work could not definitely benchmark the different deep learning approaches. The conducted experiments indicate that the deep learning approaches have to be optimised to reach their full potential in our experimental setup. Furthermore, it was discovered that the Pairwise_Lstm approach has been optimised by dividing the input into smaller segments. This could also be a possible improvement for Luvo and Pairwise_Kldiv and must be investigated for these approaches for a meaningful benchmarking.

Preface

After our project work within the field of Reinforcement Learning we decided that, while we could have further expanded our knowledge in this subject, we wanted to explore a different field of Machine Learning. We imagined that we could take our expertise about already used technologies into a new project but also wanted to extend our knowledge to other AI subjects. We found all of the above in this Bachelor thesis where we could learn a lot about the speaker clustering task. We expanded already existing code and read about different approaches that were worked out in previous Bachelor theses by other students. This required that we dive deep into the whole subject as we needed to understand every approach to a certain degree. We saw this as a fitting challenge and were eager to work on this Bachelor thesis.

Before we started working with the already provided code base we received an updated version from Claude Lehmann and Christian Lauener. In their Bachelor thesis they were working on integrating a new data set into the project and already fixed some bugs in the code base. Especially in the beginning they helped us to get the code working and further exchange with them had a positive influence on our project.

Despite this help we had some problems to get the code working. Some parts of the code did not work from the beginning and the code required much of our attention while we also studied the fundamentals of the speaker clustering task and the already developed deep learning approaches.

Luckily, afterwards the work on the code base was pleasant. While we were trying to implement our features to benchmark the different approaches we also tried to implement them as clean as possible and fixed various bugs and code smells as we saw them. In the end we were satisfied with our updated version of the code and are sure that it is now easier to understand for future teams that work on this task. The benchmarking system should help future teams to get results that provide more information about the performance of an approach and also serve as a sound environment for all approaches.

This project was an interesting experience where we learned a lot about state of the art as well as experimental approaches and speaker clustering in general. We are happy to end our Bachelor studies with such an interesting work. In the end we want to thank our supervisor Dr. Thilo Stadelmann. He helped us to better understand the speaker clustering task, gave us additional resources and information when we needed them and motivated us to create the best possible version of this work. We also want to thank Gian Spörndli and Urs Sonderegger for proofreading this work.

Zurich University
of Applied Sciences

School of
Engineering

DECLARATION OF ORIGINALITY

Bachelor’s Thesis at the School of Engineering

By submitting this Bachelor’s thesis, the undersigned student confirms that this thesis is his/her own work and was written without the help of a third party. (Group works: the performance of the other group members are not considered as third party).

The student declares that all sources in the text (including Internet pages) and appendices have been correctly disclosed. This means that there has been no plagiarism, i.e. no sections of the Bachelor thesis have been partially or wholly taken from other texts and represented as the student’s own work or included without being correctly referenced.

Any misconduct will be dealt with according to paragraphs 39 and 40 of the General Academic Regulations for Bachelor’s and Master’s Degree courses at the Zurich University of Applied Sciences (Rahmenprüfungsordnung ZHAW (RPO)) and subject to the provisions for disciplinary action stipulated in the University regulations.

City, Date:	Signature:
....	....
	....
	....

The original signed and dated document (no copies) must be included after the title sheet in the ZHAW version of all Bachelor thesis submitted.

Contents

1. Introduction

Speaker recognition comes in three different flavours. Firstly, there is speaker identification, where a system can decide if a sample belongs to a specific, known person or not. Secondly, with speaker clustering a system decides for multiple spoken samples which sample belongs to which speaker without knowing the speakers in advance. Finally, speaker diarization means that a system can automatically split a recording into slices and identify which slices were spoken by whom, even if no speakers were previously known to the system. Over the years, different approaches have been developed for all of these problems.

In 1995, speaker recognition was approached by using Gaussian mixture models (GMM). The idea of this approach is to approximate the characteristics of all speakers during training. When given a spoken segment of a speaker, the system would calculate which characteristics best fit to the given input and identify which speaker it belongs to. With this idea, the system was quite successful at identifying speakers.

Later, in 2010, a new way of representing a speaker, the I-Vector, has been introduced. This system was based on a more mathematically sophisticated model to represent a speakers characteristics. This approach has proven to be successful and became state of the art for speaker recognition.

With deep learning gaining popularity by solving more and more complex problems with neural networks, it was soon also applied to speaker recognition. The basic idea there is, that neural networks can learn to distinguish speakers without any previous knowledge about speakers or their voices. These neural networks have opened new doors to develop speaker recognition systems. Thus, the Institute of Applied Information Technology (InIT) at ZHAW has developed different deep learning approaches in the past years, some of which have shown promising results.

1.1. Motivation

In previously conducted Bachelor theses at the Institute of Applied Information Technology (InIT), different deep learning approaches for speaker clustering have been published. While these methods showed promising results, they are not fully comparable to each other due to minor differences in their evaluation process.

This work aims to create a common experimental setup and enhance the quality of all methods in order to be able to make the results comparable. To achieve this goal, we want to improve the evaluation process, introduce new metrics and build an I-Vector and a GMM system in order to classify the achieved results. With these changes, we seek to provide a benchmarking system to evaluate the speaker clustering approaches and find the most promising approach to do further research.

1.2. Problem Formulation

The purpose of this thesis is to benchmark different deep learning approaches for speaker clustering. Which of the existing speaker clustering approaches performs best? This is the question this thesis tries to answer. To find this answer we need to build a framework to provide a conceptually solid foundation to compare the existing speaker clustering approaches. The framework should also provide meaningful metrics to measure the performance and former state of the art approaches as reference.

2. Related Work

Reynolds and Rose [RR95] applied Gaussian Mixture Models on the speaker identification task. It showed good results for that time and was the state of the art approach for speaker recognition for a long time. While it is outdated by now it serves as a good reference point for performance comparison.

The I-Vector introduced by [DKD$^+$10] was a new way to represent speakers by modelling the variability of the speakers. The idea of the approach was to model the variability of all speaker in a single matrix and extract feature vectors out of this matrix based on a spoken sample that was fed into the system. This has proved to be successful and became the new state of the art approach for speaker recognition.

Lukic and Vogt [LVDS16] then applied a deep learning approach using a Convolutional Neural Network (CNN) to speaker clustering and speaker identification tasks in their thesis at ZHAW. The idea of this approach was to interpret internal components of such a neural network as "fingerprints" for the speakers. The neural network was trained to identify a number of speakers. During this task the internal components automatically become "unique fingerprints" of the speakers. In their paper, they stated that they have achieved results comparable to state of the art systems of that time.

A year later, in 2017 Lukic and Vogt extended this approach in [LVDS17]. Again, they used internal components of a CNN as fingerprints. This time they specifically trained the neural network to produce unique fingerprints for each speaker. With this, they have surpassed the results of their previous approach.

Subsequently, in 2018 Glinski-Haefeli and Gerber [SGHGD18] took this approach and substituted the CNN for a Long Short-Term Memory Network (LSTM). The LSTMs are specialised for sequential data, therefore it made sense to apply it to speaker recognition since speech is naturally sequential. Again they used internal components of the neural network as fingerprints and again they trained the network specifically to produce unique fingerprints for each speaker.

During their project work at ZHAW, Niederer and Heusser [NH17] merged the three deep learning approaches described above into one software suite. The suite offers a standardised interface for each approach, a clean separation of specific and common software components and an extended documentation. In their work they also prove that the performance of all the approaches was not influenced by their changes.

Hibraj et al. [HVSP18] used a clustering algorithm called Dominant Sets for the speaker clustering task. They used the same data set for their paper as [LVDS16], [LVDS17] and [SGHGD18], and achieved comparable results with a CNN.

3. Fundamentals

This chapter provides an overview over the most important technologies used for speaker clustering, especially within the scope of this work. Where needed, we provide further resources for readers keen on understanding the technology in greater detail.

3.1. Spectrograms

Audio files can be converted into spectrograms. Spectrograms visualise the intensity of a frequency band over time. The colours indicate how much a frequency range is emphasized. A spectogram of an audio file is visualized in Figure 3.1.

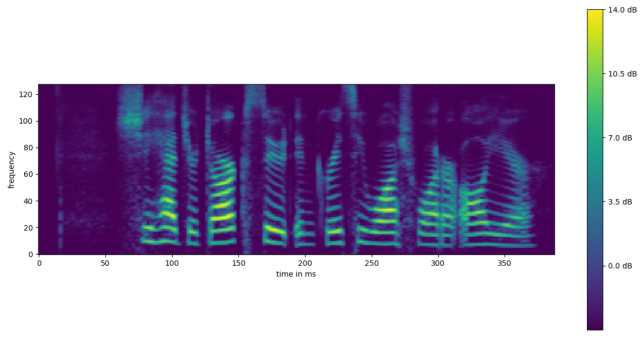

Figure 3.1.: Spectrogram of a recorded speaker. (Original by [GE17])

Although frequencies are usually given in Hertz, the frequency scale of spectrograms is often given in the Mel scale. This scale is inspired by the human perceptible range of frequencies. Therefore, such spectrograms are often calles Mel-Spectrogram or Mel-Frequency spectrograms.

3.2. Mel-Frequency Cepstral Coefficients

Another form of representation of audio are the Mel-Frequency Cepstral Coefficients (MFCC). MFCCs have frequently been used for speaker recognition for some time [Log00, Chapter 2]. MFCCs are created in five steps visualised in Figure 3.2.

First, the audio is split into small frames of fixed intervals. Then, the Discrete Fourier Transformation is calculated for each frame in step two. In the third step, only the logarithm of the amplitude of the signal is kept, phase information is not of great use. The result is then smoothed and transformed to Mel-Frequencies, before applying Discrete Cosine Transform to decorrelate the components gained in step four [Log00].

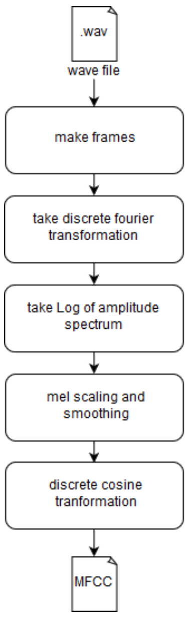
.wav
wave file
make frames
take discrete fourier transformation
take Log of amplitude spectrum
mel scaling and smoothing
discrete cosine tranformation
MFCC

Figure 3.2.: Creation of MFCC features. (Adapted from [Log00])

3.3. Embeddings

To be able to do speaker clustering, the recordings of a speaker have to be transformed into so called embeddings. An embedding can be interpreted as a feature vector that contains all the important information of a speaker and serves as a new representation of the input. These embeddings are created by using neural networks as a tool to extract these important features from the input data [STS+18].

3.4. Clustering

The embeddings created by the deep learning approaches are used as the input for the clustering. Before the actual clustering the embeddings created from sentences of the same speaker are concatenated to a so-called utterance. The algorithm then clusters the different utterances. The clustering used in all deep learning approaches from ZHAW is hierarchical agglomerative clustering with complete linkage and the cosine metric [NH17]. In the beginning every cluster contains only one utterance. The hierarchical agglomerative clustering works by connecting the nearest clusters until all clusters are linked to a single cluster. Then we can cut the linkage at a certain threshold to get a specific number of clusters. An example result of such a clustering can be seen in Figure 3.3.

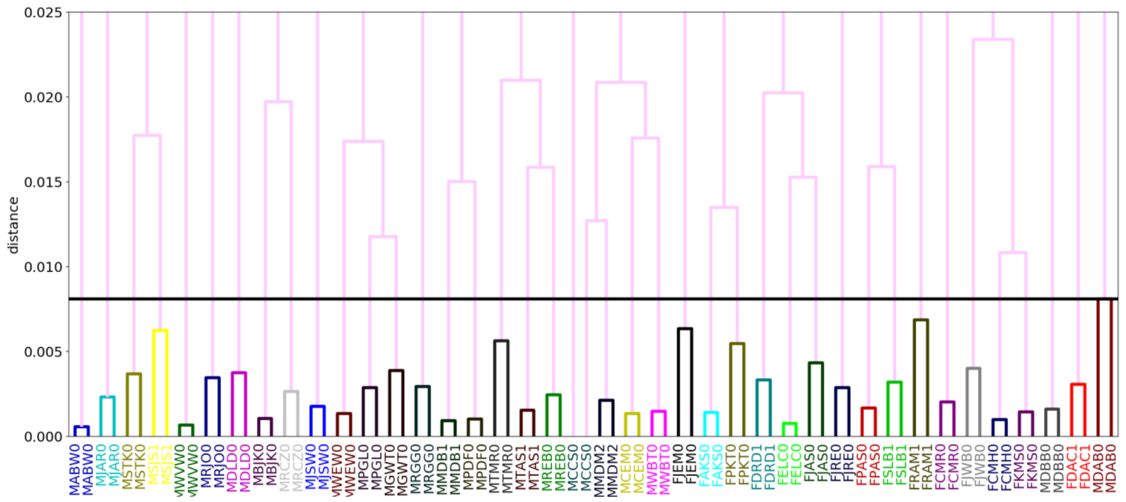

Figure 3.3.: The result of a embeddings clustering displayed in a dendrogram. The horizontal black line is the threshold on which we cut the linkage. (Original by [SGHGD18])

To decide which clusters to link we have to define how to measure the distance between clusters. Because we use complete linkage, we decide by searching the smallest maximum distance between two clusters. The maximum distance is defined as the largest distance between two data points in these two clusters. This can be seen in Figure 3.4. The associated equation is Equation 3.1.

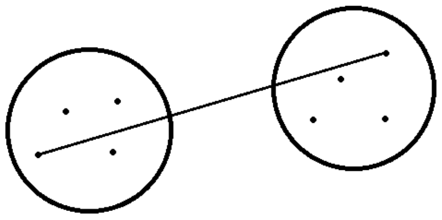

Figure 3.4.: A visualization of the maximum distance between two clusters.

$$d(u, v) = max(dist(u[i], v[j])) \tag{3.1}$$

The cosine distance is used to calculate the distance between two data points in Equation 3.1. It is defined in Equation 3.2.

$$Cosine\ distance = 1 - \frac{u \cdot v}{\|u\|_2 \, \|v\|_2} \tag{3.2}$$

3.5. Gaussian Mixture Models

This section describes the motivation of using Gaussian Mixture Models for speaker recognition tasks and explains the general concepts of this approach. For more detailed information and mathematical background, we refer to the following resources:

Recommended Resources:

- GMM for speaker clustering: *Robust Text-Independent Speaker Identification Using Gaussian Mixture Speaker Models* by [RR95]
- EM algorithm blogpost: *The EM Algorithm Explained* by [Bi19]
  - https://medium.com/@chloebee/the-em-algorithm-explained-52182dbb19d9

The Idea of Gaussian Mixture Models in the Field of Speaker Recognition
Gaussian Mixture Models (GMM) are weighted sums of Gaussian densities. Since every Gaussian density is described by the mean value and the covariance, a GMM can be described by mean vectors, covariance matrices and the mixture weights of all densities. For speaker recognition, one GMM will represent the characteristics of one speaker [RR95, Section II B]. Reynolds and Rose describe two motivations for modeling speakers with GMMs: Firstly, one can intuitively think of a GMM as an underlying set of acoustic classes. By adjusting mean, covariance and weights, the GMM represents the configuration of these classes that make up a speakers voice. The second motivation is that GMMs can smoothly approximate other densities. This ability is used in speaker recognition [RR95, Section II C].

Training a GMM
A GMM is trained by finding the optimal parameters (mean-, covariance values and weights) to approximate a given target. To obtain the values, the expectation-maximization (EM) algorithm is used. Initially, the GMM is randomly initialised. In the E-Step of the EM algorithm, the likelihood of a data point being produced by each Gaussian density is calculated. Given this likelihood, the EM algorithm then calculates the new parameters of all densities in the M-Step. This can be repeated until a threshold for the likelihood is reached.

For speaker recognition, this process is required for every speaker to approximate the speakers utterance represented by MFCC. To identify a speaker, the likelihood of the MFCC representation being produced by each GMM is calculated. The best GMM resulting in a probability higher than a threshold, identifies the utterance being produced by the corresponding speaker [RR95].

3.6. I-Vector

With the following section, we introduce the basic concepts of I-Vectors. The goal is to cover the key ideas and how an I-Vector system is used for speaker recognition. Again we refer to the following resources for more in-depth information on the approach:

Recommended Resources:

- I-Vector concept: *Front-End Factor Analysis for Speaker Verification* by [DKD$^+$10]
- Training process: *An I-Vector based Approach to Training Data Clustering for Improved Speech Recognition* by [ZXYH11]

The Idea of I-Vectors in the Field of Speaker Recognition
In their 2010 paper, [DKD$^+$10] proposed a new way to represent speakers. The main idea of this representation is to model the variability of speakers in order to be able to recognize them. Before this paper, a speakers utterance was usually represented by a supervector s that contained all the features of that utterance:

$$s = m + Vy + Ux + Dz \tag{3.3}$$

where m is a speaker independent component, Vy is the speaker dependent component, Ux is the channel dependant component and Dz is a residual. V, D are matrices that define the speaker subspace, U is a matrix that defines the channel-subspace. x, y and z contain the speaker and channel dependant factors that define a given utterance [DKD$^+$10, Chapter II].

An experiment that showed that the channel dependant components also contain information about speakers motivated a new representation for the samples supervector s:

$$s = m + Tw \tag{3.4}$$

where, as in Equation 3.3, m is the speaker independent component but Tw represents the total variability of speaker and channel effects. Here, T is a matrix that defines the total variability space of a speaker and w is a vector that contains the total factors of a speaker. This vector was referred to as *identity vector*, short *I-Vector* [DKD$^+$10, Chapter III].

Training an I-Vector System
The speaker independent component m in Equation 3.4 is modeled as a Universal Background Model (UBM) [ZXYH11, Section 2.1]. A UBM is a GMM as described in Section 3.5 and is also trained the same way. The UBM usually contains a large number of Gaussian distributions and is trained on a large number of speakers so it is not overly tuned to specific speakers [Rey97, Section 2.2]. With the UBM trained, the total variability matrix can be trained next. An EM-like algorithm can be used to estimate the matrix (vgl. [ZXYH11]:

Initialising the matrix T

Initialise each component T_{ij} of the matrix randomly.

The E-Step

Calculate the estimation of the I-Vector w using the current values of T for an utterance in the training set.

The M-step

Calculate the new values to update T

With the trained UBM and total variability matrix, the model can calculate I-Vectors of a given utterance. These I-Vectors are then used as embeddings to cluster the speakers.

3.7. Deep Learning Approaches

In this section, we will introduce the different deep learning approaches that were developed at the InIT at ZHAW. There are three different approaches, namely "Luvo", "Pairwise_Kldiv" and "Pairwise_Lstm" as they are called. The approaches were developed in this order, each one year apart from the next. Since these approaches have been the product of past theses, we will not go into detail about the approaches nor their underlying technologies. A fundamental understanding of neural networks is recommended for this section.

Recommended Ressources:

- CNN: Chapter 1 in *Neural Networks and Deep Learning* by [Nie15]
- LSTM: Article *Long Short-term Memory* in Neural Computation by [HS97]
- LSTM Blogpost: Understanding-LSTMs [Ola15]
  - http://colah.github.io/posts/2015-08-Understanding-LSTMs/
- KL-Divergence: Paper *Neural network-based clustering using pairwise constraints* by [HK15]

3.7.1. Luvo

In their 2016 paper, Lukic and Vogt [LVDS16] describe their method called "Luvo" that applies a Convolutional Neural Network (CNN) to a speaker recognition task. CNNs have been proven to perform well when recognizing and analyzing images. Therefore the fundamental idea is to convert the spoken samples into spectrograms that can be fed into a CNN.

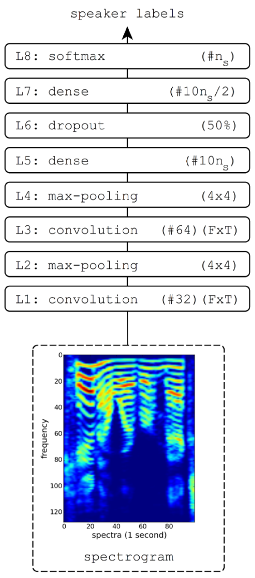

Figure 3.5.: The different layers of the Convolutional Neural Network used for Luvo. (Original by [LVDS16].)

As shown in Figure 3.5, the CNN consists of multiple layers. Each layer consist of multiple nodes, so called neurons. The first layer of the network consists of FxT neurons where F is the frequency and T the time axis of a single spectrogram. A spectrogram displays 128 frequency elements and is divided into segments of one second each. This leads to an input dimension of $128x100$. There are multiple large layers on top, the hidden layers, that are used to analyze the spectrogram and predict which speaker it belongs to. For this purpose, each layer will forward its values to his immediate successor. Finally, the last layer, the softmax layer, consists of as many nodes as there are speakers in the training set. This layer represents the prediction of the neural network. Every node in this layer can be thought of as a light bulb belonging to a specific speaker. The neural network switches on the light bulb belonging to the speaker it has identified. While the predictions will be random at the beginning, the neural network will optimize the neurons in the hidden layers over time to improve the predictions.
Since the last layer can only classify the number of speakers in the training set, this is not sufficient to cluster unknown speakers that have not occurred during training. To cluster speakers, the last layer is omitted and the second last layer L7 in figure 3.5 is used for this. One can argue that if the last layer produces different outputs for every speakers, the values in the hidden layers must be unique to a certain degree as well. Therefore, the values of one of the hidden layers can be interpreted as a fingerprint for a speaker. In this case the values of L7 are used as fingerprints. These fingerprints are used as the embeddings for the clustering.

3.7.2. Pairwise_Kldiv

Lukic and Vogt have extended the Luvo approach in 2017 when they applied pairwise constraints to train the network. They took the Luvo approach but changed the networks goal for the training. With the new approach named Pairwise_Kldiv, they compare embeddings pairwise, training the network to produce similar embeddings for segments of the same speaker and inherently different embeddings for segments from different speakers. To measure the difference between the pairs, the Kullback-Leibler divergence was used. This divergence indicates how far two embeddings are statistically apart.

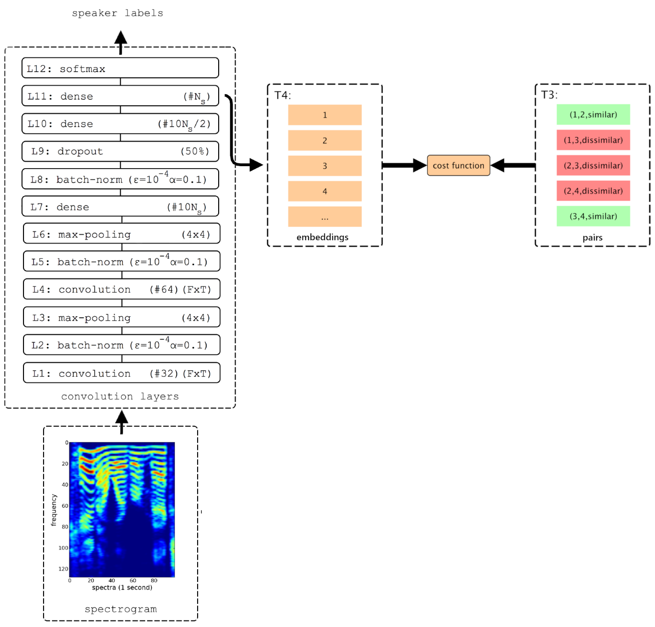

Figure 3.6.: Visual representation of how Pairwise_Kldiv trains. (Adapted from [LVDS17].)

As before, one second long snippets of spectrograms are fed into the neural network and processed across the layers. Instead of learning to identify speakers, the embeddings at second last layer, which will be used for clustering, are compared at training stage as shown in Figure 3.6. To asses the performance of the network, the network takes a batch of 100 segments and calculates the embeddings of these segments. Out of the 100 embeddings all possible pairwise combinations are formed and these pairs are compared to each other. The neural network receives a good score if dissimilar speakers have different embeddings or similar speaker have nearly the same embeddings. This ensures that the neural network is training the task it is designed to do, which is creating good embeddings. The embeddings are then again used for clustering the speakers.

3.7.3. Pairwise_Lstm

CNNs are by design not well suited to process sequential data like speech. By using CNNs to analyse segments of spectrograms, the output is only based on the one segment that was fed into the CNN. Recurrent Neural Networks (RNN) on the other hand are built to be able to process sequential data and therefore well suited for speaker recognition. For this reason [SGHGD18] have substituted the CNN for a long short-term memory (LSTM) network, which is one type of a RNN.

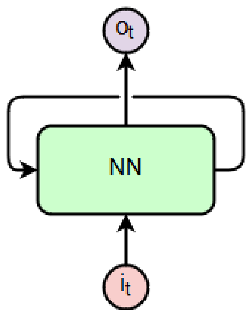

Figure 3.7.: RNN visualised as loop. (Adapted from [Ola15].)

Compared to the CNN, this type of neural network can remember information on previous inputs by reconnecting it into the network again. This is visualised in Figure 3.7. Although this figure might suggests that only information from the immediate predecessor step is looped into the neural network again, the network can take older information into account as well. With the ability of taking long term information into account, an LSTM must be able to decide which of the past information should influence the current result. This made separate control mechanisms necessary.

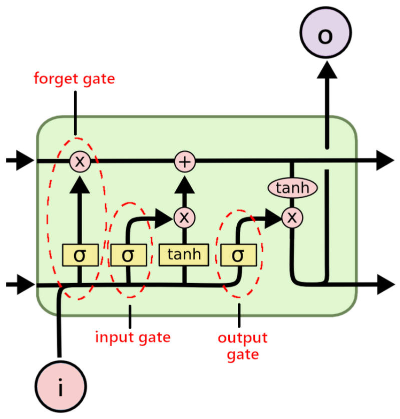

Figure 3.8.: Internal Gates of an LSTM. (Adapted from [Ola15].)

To control which forwarded information to take into account, three so-called gates are implemented as shown in 3.8:

The input gate
This gate controls what information of the new input is used.

The forget gate
This gate controls what previous information should be kept or forgotten.

The output gate
This gate controls what information will be forwarded for the next steps.

The gates implement a so-called sigmoid function which assigns values between 0 and 1 to the information. This indicates how much of the information is let through the gate, 0 being nothing and 1 being everything. The optimal behaviour of these gates is also learned during training.

In training, the segments of a spectrogram of a spoken sample are again fed into the neural network as before. This type of neural network is able to take previous segments into account as described above. Apart from that, training works similar to the training of Pairwise_Kldiv: To asses the performance during training, the pairwise KL divergence is used identically as in Pairwise_Kldiv. The embeddings of the hidden layer are compared pairwise to each other and scored as described in the section on Pairwise_Kldiv.
However, one key difference of this approach is that the spectrograms are no longer split into segments of one second length but 400 milliseconds. In this work, [SGHGD18] have experimented with different segment sizes to find the optimal segment size. They prove in their work that a segment size of 400ms improves the performance of this approach.

4. Concepts

One goal of this work is to build a system containing several speaker clustering approaches, data sets, metrics and clustering algorithms in order to be able to benchmark the deep learning approaches. In this chapter, we describe the architecture and concepts of the system we built for this thesis.

4.1. Architecture

The inner structure of the benchmarking system can be explained best by dividing it into layers.

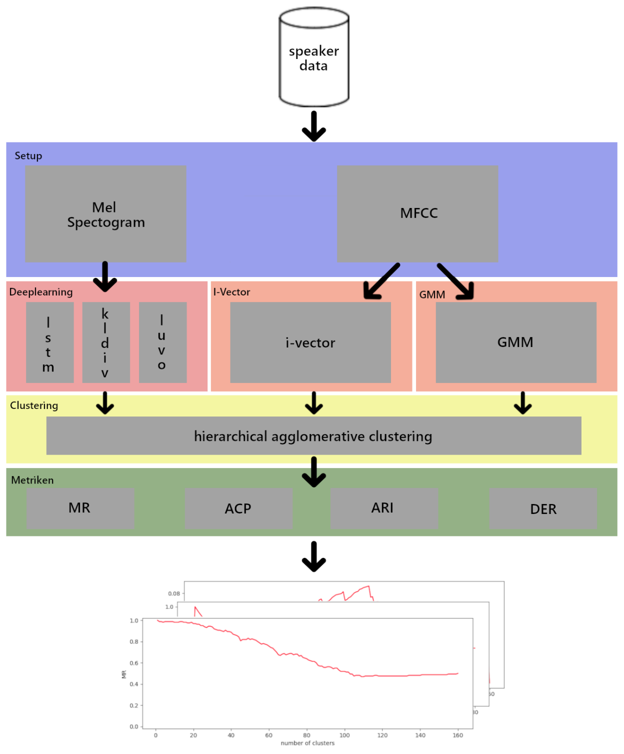

Figure 4.1.: The basic architecture of the system visualised

In their work, Niederer and Heusser [NH17] have already build a system that contains the three independently developed approaches. Their work has already introduced a basic structure by extracting the setup and clustering process from each of the approaches and implementing independent software components for these tasks (see blue and yellow layer in Figure 4.1). To evaluate result, their work has also provided a metric to measure the performance. Furthermore, they also introduced interfaces that allow to easily extend the system with additional speaker clustering approaches.

Our work uses this system as a foundation and extends it.[1] Most notably we introduce two more speaker clustering approaches into the system (see orange boxes in Figure 4.1) which has required to convert the speaker data into MFCCs (blue layer in Figure 4.1). We provide three new metrics to be able to analyse performance in more detail (green layer in Figure 4.1). [2]

In summary, the system provides features to convert the speaker data into the required formats, the environment to train the approaches in the system, the clustering algorithm and metrics to evaluate the performance of the approaches, and finally the functionality to plot the results.

4.2. Experimental Environment

In order to eliminate differences in the training and evaluation process we suggest a new experimental environment in this section. We describe what data set we use and how we suggest to split the data set for training and testing to guarantee a conceptually sound process.

4.2.1. TIMIT Data Set

As mentioned above, the system does convert speaker data into different formats to train different approaches. The system is designed to be open for different speaker data. However, as of now the system has only been used with the TIMIT data set. The TIMIT corpus is specifically designed for development and evaluation of automatic speech recognition systems. It contains 438 male and 192 female speakers which is a total of 630 speakers. Each speaker reads ten phonetically rich sentences in one of eight major dialects of American English. The acoustic quality is exceptional because the recordings were done in a sound recording studio. Typically, one part of the data set will be used to train the approaches while the other part will be used to evaluate the trained model [Con].

4.2.2. Training, Validation and Test Set

When starting work on a machine learning project one has to think about how to split the data. To emulate the real world, a machine learning system should not be evaluated on data that it has already seen during training. Therefore, the data has to be split into three different sets, the training set, the validation set and the test set.

[1]During this process we have discovered that their system does not work anymore with the dependencies provided in [NH17, Section 3.8]. We provide an updated list in Appendix A

[2]During the course of this work we also implemented an additional clustering algorithm called Dominant Sets. Due to time shortage, we have not been able to explore the full potential of Dominant Set clustering in this thesis. We discuss this topic in Appendix D.

The training set is the set on which the algorithm trains. It is usually the set that contains the largest amount of data as normally the accuracy of a model increases the more data it has to learn from.

The validation set is used to frequently evaluate a trained model and then tune the parameter of the algorithm. The algorithm does not learn on the data in this set and has therefore never seen it but the results of the evaluation still influences the training process.

The test set contains the data on which the algorithm is tested on after it was trained and evaluated several times. It is also completely separate from train- and validation-set. It provides the final result of the trained model.

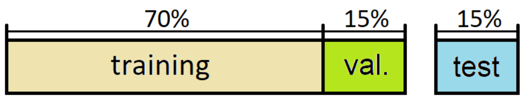

Figure 4.2.: Visualization of a possible split of data into training, validation and test set.

The three types data sets are visualized in Figure 4.2. The reason why a validation and test set are needed is that when using only train- and test-set this might lead to peeking, which means that information about the test set influenced the learning algorithm. This might result into good performance on the test-set but when using the model on new data the performance might be worse [RN10, Section 18.2 and 18.4].

4.2.3. Experimental Setup

In all previous theses, there was one separate test set containing 40 speakers as described in [SF09]. The remaining 590 out of the total 630 speakers in the TIMIT data set were used as training set for Luvo. A second training set of 100 speakers (containing 50 male speakers and 50 female speakers) was introduced in [LVDS16] and subsequently used to train Pairwise_Kldiv and Pairwise_Lstm. For our benchmarking, we require one training set to be used for all approaches to guarantee a common experimental setup. Furthermore, in one of their experiments, [LVDS17] used a test set containing 80 speakers. These set includes the 40 speakers from the original test set plus 40 additional speakers. Later in [SGHGD18] another test set with 60 speakers is used as well. Since there are only 630 speakers in total, it is obvious that both those test sets must contain speakers that are also part of the training set containing 590 speakers which should be avoided as described in Section 4.2.2.[3] Additionally, the fact that only one set was used for both testing during development and as final test might lead to peeking. Due to these problems, we suggest a new set of data sets as follows:

Most importantly, the original test set containing 40 speakers must be kept unchanged as a test set to retain comparability of the result. As a first step, we decided for all the

[3]In the experiments conducted by [NH17] (see page 17-19) it can be observed that the MR increases for Pairwise_Kldiv and Pairwise_Lstm when the larger test sets are used which intuitively makes sense. For humans it is also harder to differentiate between 80 speakers than 40. In contrast, the MR of the Luvo approach decreases. This is most certainly due to the fact that the approach misclassified more speakers of the 40 unknown speakers than the 20 or 40 additional speakers in the larger test sets it has already seen during training. Therefore, the MR decreases for the larger test sets.

	training	validation	test
number of speakers	470	80	80
distribution of male/female speakers	stratified	stratified	stratified
number of utterances (experimental setup)	-	2 utterances per speaker containing 8+2 sentences	2 utterances containing 8+2 sentences
number of utterances (short utterance setup)	-	10 utterances per speaker containing 1 sentence each	10 utterances per speaker containing 1 sentence each
compatibility	-	-	contains original test set

Table 4.1.: Table of the different data sets needed for the experimental setup.

other sets, that they should be stratified. This means, the ratio between male and female speakers should be equal to the ratio of the TIMIT set overall. The ratio of the TIMIT set is 70% male and 30% female. This prevents that a set contains only a small percentage of female speakers which would possibly lead to gender biased results.

Secondly, we introduced a validation set. This validation set can be used while still developing the algorithms or tweaking their parameters. We reserved 80 speakers for this set, because the idea of anticipating the need for larger sets for testing as [LVDS16] already did, seemed sensible. The 80 speakers are divided into a validation set containing 40 speaker, 60 speakers and all 80 speakers. Another 80 speakers were reserved for the test set. The 80 speakers are split into a test set containing only the 40 speakers originally used as test set since [LVDS16], a test set containing 60 speakers and a set of all 80 speakers. Finally, the remaining 470 speakers (630 minus 80 speakers for the validation set and minus 80 speakers for the test set) are used as training set. This set is also stratified.

This setup of different data sets allows to develop and improve algorithms without peeking and without speakers being used in both test and training set.

Thirdly, we also introduced the short utterance setup. Previously the deep learning approaches only predicted the clustering for two utterances per speaker. One of this utterances consists of eight sentences from one speaker while the other utterance consists of two sentences. In the newly introduced short utterance setup every utterance consists of one sentence from a speaker. This allows us to analyze the results for more clusters and also how well the clustering performs when a speaker speaks for a shorter period of time.

5. Implementation

In this section, we discuss the changes and additional features we have implemented into the existing system.

5.1. Metrics

To evaluate the different approaches we use four metrics. The Misclassification Rate (MR), the Adjusted RAND Index (ARI), the Average Cluster Purity (ACP) and the Diarization Error Rate (DER). Using different metrics provides additional information about the performance of the approaches. While the MR represents how many samples are incorrectly clustered the ACP and the ARI indicate how good the samples are grouped/separated. Finally, the DER gives us similar information as the MR but also includes how long the incorrectly clustered sample is. It is also used as a reference point for future projects that revolve around the speaker diarization task.

5.1.1. Misclassification Rate (MR)

The MR measures how many utterances are not mapped to their corresponding cluster and is defined as follows by Kotti et al. [KMK08]:

$$MR = \frac{1}{N} \sum_{j=1}^{N_{cl}} e_j \tag{5.1}$$

Where N is the total number of utterances to cluster, N_{cl} is the number of found clusters and e_j are the utterances of a speaker j that are assigned to the wrong cluster. According to [SGHGD18] the correct cluster for any speaker is the one that fulfills the following two conditions:

1. The cluster is the one that contains most utterances of the speaker.
2. The cluster contains no other speaker that has more utterances in this cluster.

The best possible value for the MR is 0 when all utterances are perfectly assigned. If all utterances are wrongly assigned the MR is 1.

The MR is the only metric that was used in the three deep learning approaches and is therefore the only metric that can be used to compare to previous results. According to [SGHGD18] the MR was defined differently in [LVDS16] [LVDS17] but the results in [NH17] use the same definition as [SGHGD18]. Therefore we compare our results with the results from [NH17].

5.1.2. Average Cluster Purity (ACP)

The ACP measures how pure the clusters are. A cluster is pure if it only contains utterances of a single speaker. The ACP ranges from 0 to 1 where 1 is the best possible value where all clusters are completely pure [HVSP18].

$$ACP = \frac{1}{N} \sum_{i=1}^{N_c} p_i \cdot n_i \tag{5.2}$$

$$p_i = \sum_{j=1}^{N_s} n_{ij}^2 / n_i^2 \tag{5.3}$$

Where N is the total number of utterances, N_c is the total number of clusters, n_i is the size of cluster i, p_i is the purity of cluster i, N_s is the total number of speakers and n_{ij} is the number of utterances in cluster i spoken by speaker j [HVSP18].

5.1.3. Adjusted RAND Index (ARI)

The ARI measures how close the predicted clustering is compared to the ground truth. The ARI is calculated by using a contingency table where X is the predicted clustering and Y is the ground truth. n_{ij} is the number of entries that co-occur in X_i and Y_j, a_i is the sum of co-occurrences in the predicted cluster X_i, b_i is the sum of co-occurrences in the true cluster Y_i and N is the total number of utterances [YR01].

	Y_1	Y_2	$\cdots$	Y_j	$Sums$
X_1	n_{11}	n_{12}	$\cdots$	n_{1j}	a_1
X_2	n_{21}	n_{22}	$\cdots$	n_{2j}	a_2
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
X_i	n_{i1}	n_{i2}	$\cdots$	n_{ij}	a_i
$Sums$	b_1	b_2	$\cdots$	b_j	$N = N$

(5.4)

Using the contingency table the ARI is defined as follows:

$$ARI = \frac{index - expected\ index}{maximum\ index - expected\ index} \tag{5.5}$$

$$index = \sum_{ij} \binom{n_{ij}}{2} \tag{5.6}$$

$$expected\ index = \frac{\sum_i \binom{a_i}{2} \sum_j \binom{b_j}{2}}{\binom{N}{2}} \tag{5.7}$$

$$maximum\ index = \frac{1}{2} \Big[\sum_i \binom{a_i}{2} + \sum_j \binom{b_j}{2} \Big] \tag{5.8}$$

5.1.4. Diarization Error Rate (DER)

The Diarization Error Rate is the main metric to compare speaker diarization performance and has been introduced by NIST in [NIS00]. In contrast to the other metrics, the DER can not only be calculated from the predicted clusters and the true clusters because it needs additional time information. Therefore, the reference and the hypothesis are used as the input for the DER. The reference is the correct information about when which speaker speaks. The hypothesis is the prediction when which speaker speaks [Gal13].

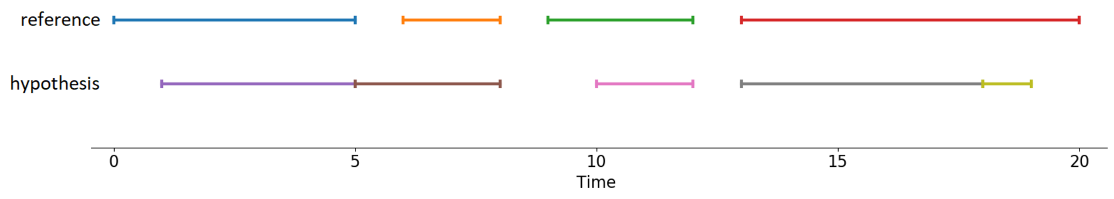

Figure 5.1.: An example reference and hypothesis. It shows that an hypothesis does not have to contain the same amount of speech segments, that sometimes a speaker is detected even if there is no speech in the reference and that the length of a speech is not always detected correctly.

In a first step a mapping between the speaker identities in the reference and the speaker tags in the hypothesis is created. The implementation used in this thesis calculates the mapping using the Hungarian method. The following definition was given by [Bru05]:

Step 1. Subtract the smallest entry in each row from all the entries of its row.

Step 2. Subtract the smallest entry in each column from all the entries of its column.

Step 3. Draw lines through appropriate rows and columns so that all the zero entries of the cost matrix are covered and the minimum number of such lines is used.

Step 4. Test for optimality:

i If the minimum number of covering lines is n, an optimal assignment of zeros is possible and we are finished.

ii If the minimum number of covering lines is less than n, an optimal assignment of zeros is not yet possible. In that case, proceed to Step 5.

Step 5. Determine the smallest entry not covered by any line. Subtract this entry from each uncovered row, and then add it to each covered column. Return to Step 3.

In a second step the DER is calculated from this mapping. As displayed in Figure 5.1 there can be several errors in the hypothesis. Those errors influence the DER and are defined in three error rates. These error rates are:

- The confusion error: When a wrong speaker is detected for a speech. The correct speaker is the one that has the most speech time in the hypothesis for the speech segments from a certain speaker in the reference.
- The miss error: When speech is detected in the hypothesis but there is none in the reference.

- The false alarm error: When speech is detected in the reference but there is none in the hypothesis.

The DER is then calculated from these error rates as follows:

$$DER = \frac{confusion\ error + miss\ error + false\ alarm\ error}{total\ reference\ speech\ time} \tag{5.9}$$

Because the task in this work is not a speaker diarization task we will never have a miss error or false alarm error. An example input as used in this work is displayed in Figure 5.2.

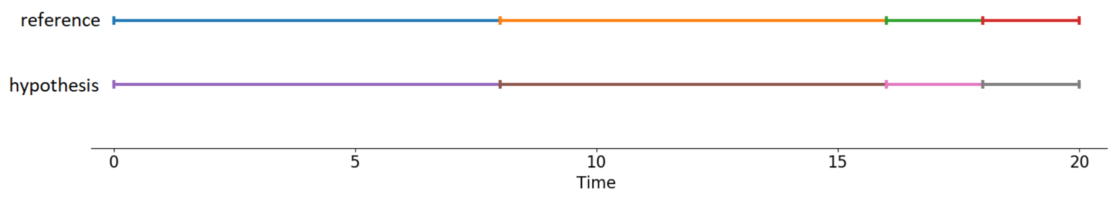

Figure 5.2.: An example reference and hypothesis as used in this work.

5.2. Comparison with the State of the Art

All implementations described above aim towards better results for the methods developed at the InIT. By implementing established state of the art speaker recognition systems into our benchmarking system, we want to provide references to be able to better classify the performance of the deep learning approaches.

5.2.1. GMM-System

To compare the performance of the deep learning approaches with a GMM approach, we built a system that estimates the characteristics of all speakers in the training set, and scores how well each speaker in the training set fits to each of the characteristic. We used the Gaussian Mixture Model implementation of the sklearn library by [PVG$^+$11].

Estimating the characteristics works according to the process described in Section 3.5: We create a GMM for every sentence in the training set which approximates the characteristics of the sentence. All the GMMs are stored in a collection which we refer to as the model. When testing, we calculate how well each GMM fits each sentence in the test set. This calculation is done by using the log-likelihood (see documentation by [PVG$^+$11]). This basically means we calculate how likely the sentence was spoken by each speaker contained in our model. Of course, none of the speakers in our test set are contained in the model. The idea is, that we analyse how similar the unknown speaker is compared to all our known speakers. This results in a vector of as many likelihoods as there are speaker in our model. This can be imagined as the speakers profile based on the sentence in the training set. Those vectors of all the sentences in the test set are then used as embeddings for clustering.

Experiment
To confirm that the implementation works, we have conducted the following experiment:

training set:
speakers_470_stratified

test set:
speakers_40_clustering_vs_reynolds

parameters:
mixturecount=128

We have trained the GMM algorithm on the 470 speakers in our training set. The parameter ”mixturecount” is used to configure how many Gaussian densities each GMM should consist of. We have experimented with different values, the experiment shown here with 128 densities performed the best. Unfortunately, this algorithm requires a lot of memory. We did not have enough resources available to run the experiment with more than 128 densities per GMM. For a comparable result, we have then tested the algorithm on the original test set containing 40 speakers. This has led to the following results:

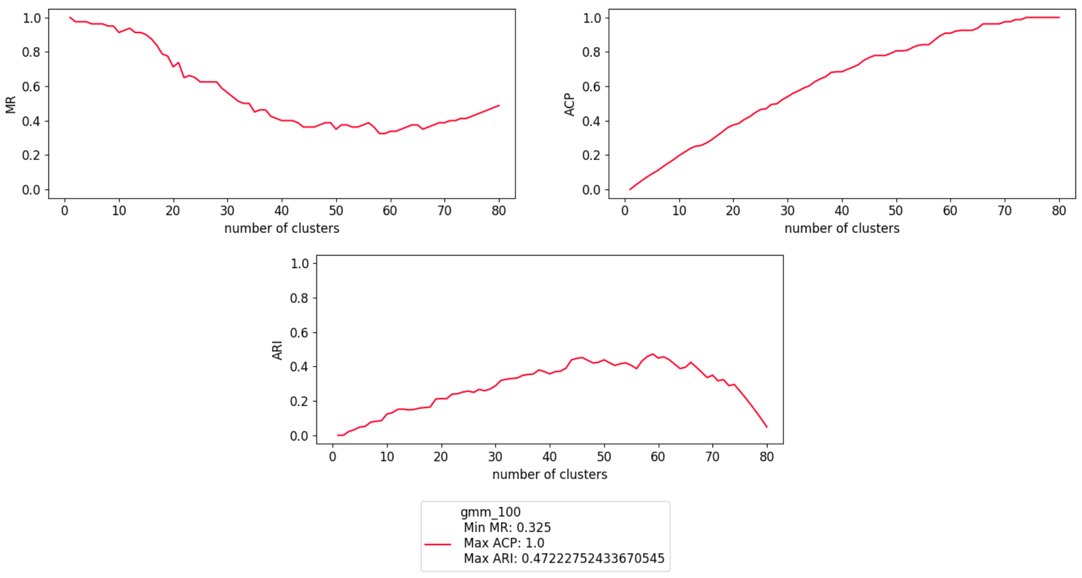

Figure 5.3.: Plot of MR, ACP and ARI per cluster size for GMM.

The MR clearly shows that this algorithm does not perform very well for speaker clustering. The ARI shows that the best clustering compared to the ground truth is reached with around 40 speakers. However, the actual ARI score is still relatively low. Finally, the ACP does increase steadily which seems promising at first. But this is due to the fact that there are two utterances per speaker, so in total 80 utterances to cluster. With the number of clusters increasing towards 80 clusters, there will be less utterances in a single cluster until eventually, there is one cluster for each utterance and therefore purity is 1. So overall, the performance of the GMM algorithm is mediocre. Since GMMs have mostly been used for speaker identification, this is not very surprising. Comparing an unknown speakers utterance to the characteristics of many other speakers intuitively cannot be an optimal model to cluster speakers. Nevertheless, it can still be used as a comparison to classify results of other algorithms.

5.2.2. I-Vector System

Since I-Vector systems have been performing well when introduced by [DKD+10], we have integrated I-Vectors to be able to further asses the performance of the deep learning approach.

To build the I-Vector system we used the I-Vector framework of the sidekit library by [LLM16]. The system provides all the building blocks of an I-Vector system as discussed in Section 3.6. The library provides the functionality to convert the TIMIT data set into MFCCs. We then train the UBM on our training set. With this, we then train the total variability matrix. Both structures and their training algorithms are provided by the library as well. During testing, we extract the I-Vectors of each sentence in the test set from our model and use it as embeddings for our clustering algorithm.

Experiment
To confirm that the implementation works, we have conducted the following experiment:

training set:
speakers_470_stratified

test set:
speakers_40_clustering_vs_reynolds

parameters:
distrib_nb=2048
rank_TV = 400
tv_iteration = 10
vector_size=400

The I-Vector algorithm is also trained on 470 speakers and tested on the original test set of 40 speakers. There are a few more parameters that can be modified compared to the GMM:

- **distrib_nb** is the number of Gaussian densities used for the UBM.
- **rank_TV** is the size of the total variability matrix.
- **tv_iterations** specifies how many iterations of the EM-Algorithm should be done to learn the total variability matrix.
- **vector_size** is the size of the I-Vector.

With this setup we achieved the following results:

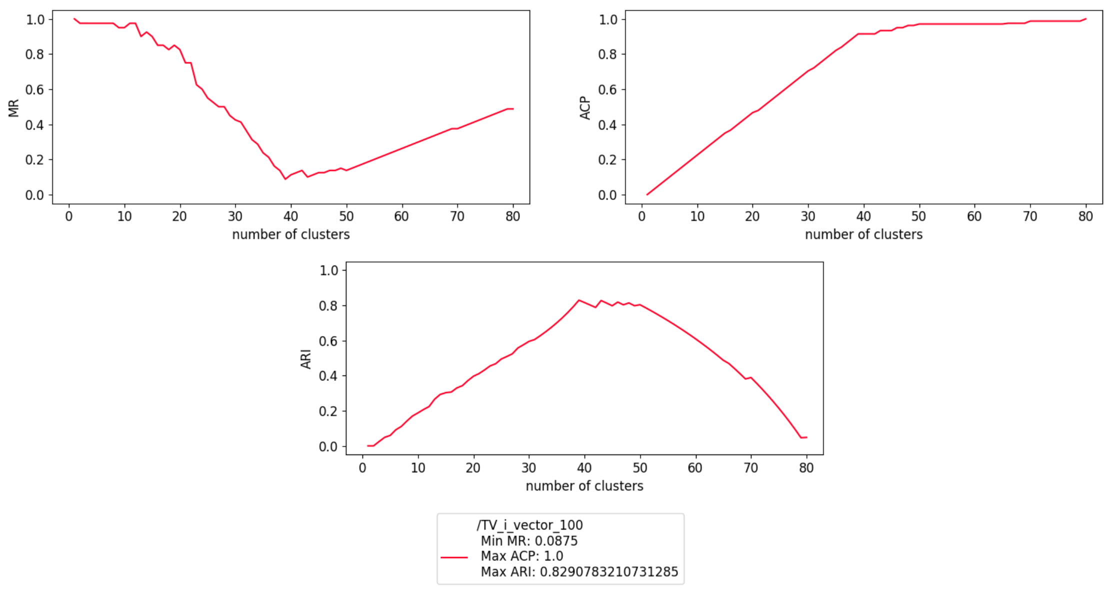

Figure 5.4.: Plot of MR, ACP and ARI per cluster size for I-Vector.

Figure 5.4 shows that the I-Vector approach reaches a MR of 0.0875. This is about as good as the Luvo approach when trained on 590 speakers according to [NH17]. The Average Cluster Purity (ACP) at 40 clusters is about 0.9 which is also a good score. Finally, the Adjusted RAND Index indicates that the solution with about 40 clusters is pretty accurate with a score of 0.829. In conclusion, this experiment shows that this I-Vector implementation performs as expected and therefore serves as a good comparison for the deep learning approaches.

6. Experiments

This section describes the experiments we conducted to test the improvements we added to the system and to benchmark the existing approaches.

6.1. Experiment 1: Testing the New Setup

As described in Section 4.2.3, we implemented an improved experimental setup with 470 speakers as training set, up to 80 speakers as evaluation set and 80 speakers as test set. After we implemented the new metrics into the system, we ran an experiment with our suggested setup to test it.

6.1.1. Setup and Expected Result

training set:
speakers_470_stratified

test set:
speakers_40_clustering_vs_reynolds

algorithms:
Luvo
Pairwise_Lstm

The authors of all the deep learning papers have suggested that the size of the dense layers in their neural networks should be adapted to fit the size of the training set (see [LVDS16, Chapter 2, Figure 1], [LVDS17, Section 2.2, Figure 2] and [SGHGD18, Section 2.1, Figure 1]). Therefore, we have adjusted the size of the respective layers. The other parameters were kept as suggested by the respective papers.

The goal of this experiment is to prove that the algorithms still work after our changes. Since we have changed the size of the dense layers of the networks, we anticipate that slight differences in performance might be possible. Other than that we expect the result to be within range of the results in [NH17]

6.1.2. Results and Discussion

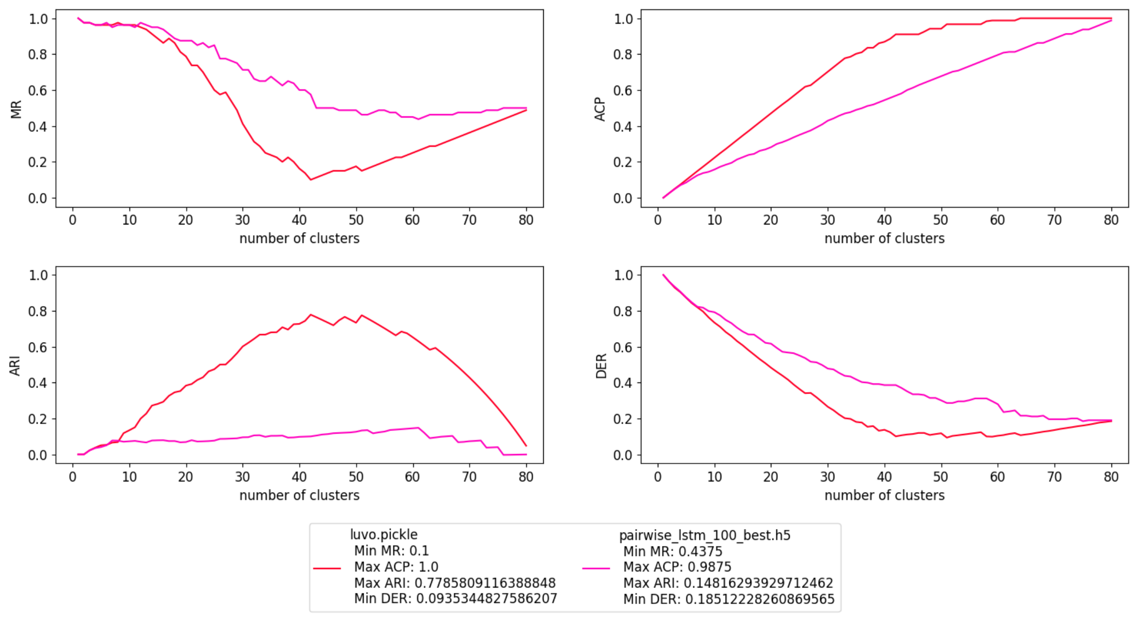

Figure 6.1.: Plot of MR, ACP, ARI and DER per cluster size for Luvo and Pairwise_Lstm with the new setup.

The results of the Luvo algorithm shown in Figure 6.1 are identical as described by [NH17]. Since we did only changed the network slightly and gave the algorithm a smaller training set as before, the MR achieved seems reasonable. The ACP rapidly increases up to around 40 speakers and stabilizes afterwards. This shows that we achieve a relatively pure solution at 40 speakers. The slow increase between 40 and 80 clusters is because less utterances will be placed in a single cluster which naturally leads to a better purity score.
The results of the Pairwise_Lstm on the other hand are surprisingly poor with a MR of 43% as opposed to the average MR score of 0.7% achieved by [NH17] in their experiments. Such a significant drop in performance might be caused by an error during the implementation of the new features. However, since no changes to any training process of any algorithm were necessary during the implementation of the features, we do believe that the reason might also be the adaption of the network layers: Since in the previous work, [SGHGD18] have only experimented with a training set of 100 speaker, the parameters are probably not optimal for the neural network we adjusted to train on 470 speakers. Nevertheless, in our opinion it is more sensible to conduct an experiment to proof the integrity of our system first before exploring this possibility.
At this point we decided to first test our system before running Pairwise_Kldiv in this setting because we would have to wait multiple days for Paiwise_Kldiv to train. We conducted this experiment later and after three days the Linux Screen of our experiment froze. We had to restart the experiment. Unfortunately, the experiment did not finish by the time this thesis was due.

6.2. Experiment 2: Sanity Check

Since the result of the first experiment could potentially be caused by a bug in the system, we test our system under the same conditions as described in the respective papers. If we have not made a coding mistake, the results of this experiment should be close to identical with the results of [NH17] and a bug can be ruled out as the explanation for the poor results in the first experiments.

6.2.1. Setup and Expected Result

training set:
speakers_100_50w_50m_not_reynolds for Pairwise_Kldiv and Pairwise_Lstm
speakers_590_clustering_without_raynolds for Luvo

test set:
speakers_40_clustering_vs_reynolds

algorithms:
Luvo
Pairwise_Kldiv
Pairwise_Lstm

The size of the network layer for all the algorithms have been reverted back and the same parameters are used for this experiment as in the papers. Since the MR has been changed, we can not take the results in [LVDS16] and [LVDS17] as reference. However, [NH17] have conducted an experiment using the new implementation of the MR to proof that their system works correctly. Therefore we use this experiment as a reference point.

6.2.2. Results and Discussion

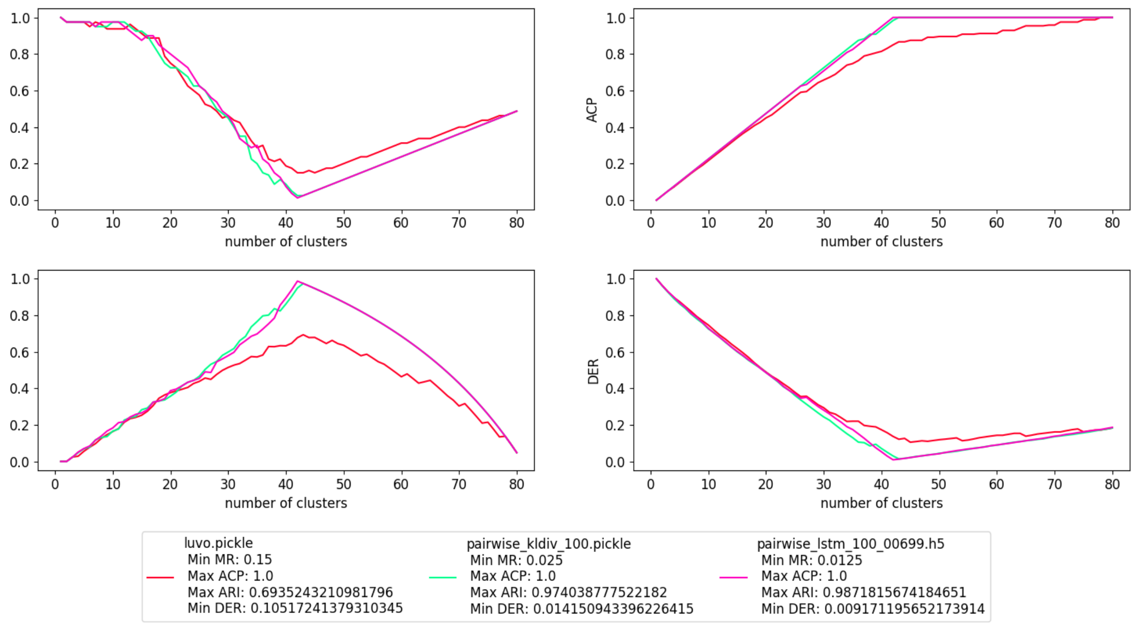

Figure 6.2.: Plot of MR, ACP, ARI and DER per cluster size for Luvo, Pairwise_Kldiv and Pairwise_Lstm with the original parameters.

The results are within a sensible range compared to the results of [NH17].[1] This indicates that there was no mistake implemented into the system. If this was the case, the results of this experiment should have clearly differed from the results in [NH17].
Therefore, the result in Experiment 1 can only be explained by the changes we made to the network. The papers of Pairwise_Kldiv [LVDS17] and Pairwise_Lstm [SGHGD18] focus on a training set containing 100 speakers only, therefore we suspect that the parameters resulting from their work are not ideal for adapted networks and there is still room for improvement. To support this hypothesis, we conduct an additional experiment.

[1]During this experiment, we have also realized that for Luvo, the length of the spectrogram segments used as input has been changed from 1s (as specified in [LVDS16]) to 500ms. We have compared the results of using 500ms long segments to the results retained by using 1s long segments in a separate experiment which can be found in Appendix C. Based on these results we have conducted all further experiments with a segment size of 500ms as well.

6.3. Experiment 3: New Setup with Original Networks

This experiment tries to show that the algorithms could potentially also perform well using our experimental setup of 470 speakers for training. For benchmarking it would make sense to compare all approaches using the same training data to not create a different setup that could potentially distort a comparison. This is the motivation of this experiment.

6.3.1. Setup and Expected Result

training set:
speakers_470_stratified

test set:
speakers_40_clustering_vs_reynolds

algorithms:
Luvo
Pairwise_Kldiv
Pairwise_Lstm

For all algorithms we use the parameters given in the respective papers. Both the network of Pairwise_Kldiv and Pairwise_Lstm have been readjusted as if 100 speakers are used as training data. The network of Luvo has been changed back to 590 speakers.

We have already seen that these combinations of parameters and networks perform well when trained on 100 or 590 speakers respectively in Experiment 2. Intuitively, the performance of the algorithms should increase, especially for Pairwise_Kldiv and Pairwise_Lstm since there are more training opportunities when using 470 speakers for training. On the other hand, according to the papers we do not use ideal neural networks for the 470 speaker in our training data. This might lead to a poor performance as in Experiment 1. In summary, we expect that the result of Pairwise_Kldiv and Pairwise_Lstm should be close to original results by [NH17]. Both algorithms learn by comparing the embeddings they produce pairwise. The adjustments of the neural network only change the dimension of the created embeddings and we believe that this should not have a significant influence on the performance of the algorithms. The Luvo algorithm learns by classifying the speakers in the training set. As discussed previously, the neural network needs as many nodes as there are speakers in the training set on the last layer to indicate what its prediction is. Since the original neural network has been designed for classifying 590 speakers, the last layer does also have enough nodes to classify the 470 speaker in our training set. Hence, we anticipate that not adjusting the network should not have a significant influence on the performance for Luvo as well. In Experiment 2, the Luvo algorithm was trained in this setting on 590 speakers. Expectations are that the performance stays more or less stable when the training opportunities are decreased to 470 speakers.

6.3.2. Results and Discussion

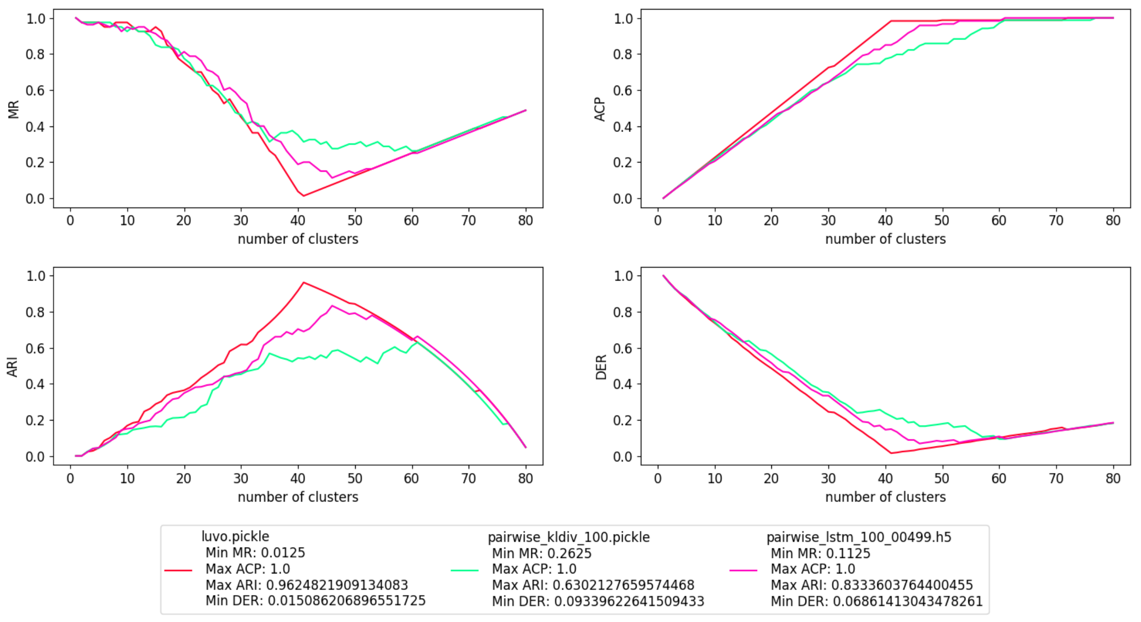

Figure 6.3.: Plot of MR, ACP, ARI and DER per cluster size for Luvo, Pairwise_Kldiv and Pairwise_Lstm with the new training set and the original parameters.

The results of the experiment have been interesting for two reasons:
Firstly, Luvo has performed particularly well, reaching a MR of 0.0125 or 1.25%. This MR is as low as the best result of Pairwise_Lstm when trained on 100 speakers. This result is surprising because in Experiment 2, the network has been trained on 590 speakers and has only achieved a MR of 15%. In their paper on Luvo, [LVDS16] only used 8 sentences per speaker whereas we have all 10 available sentences of the 470 speakers in our training set. In total there are about the same number of sentences in the training set: $590 * 8 = 4720$ sentences in their training set and $470 * 10 = 4700$ sentences in our training set. But our training set contains 2 additional sentences per speaker, giving the network more data per speaker to train. We believe that this might be a reason for the significantly better MR.
Secondly, Pairwise_Lstm and Pairwise_Kldiv have improved to a MR of 0.1125 and 0.2625 respectively. Compared to the results in Experiment 2, this is still a relatively poor result. We see two potential reasons for this: Firstly, the papers clearly suggest that the network should be adjusted to the number of speakers in the training data. Experiment 1 has proven that only changing the network but not the parameters has a negative influence on the performance. However, there might be a lot of potential in searching for the optimal parameters for the adjusted network as well. Experiment 2 shows that good performances can be achieved if the size of the training set, the parameters and the network are well adjusted to one other.
Secondly, we can observe that only the two algorithms that use pairwise KL divergence for learning perform poorly. Upon further investigation, we have found that the neural network calculates the embeddings of a batch of 100 randomly chosen segments and calculates the pairwise KL divergence of all the possible pairs of segments. Based on this calculation it improves its behaviour before the procedure is repeated. This is done 1000 times during the training. In both Pairwise_Kldiv and Pairwise_Lstm the batch size was set to 100 in the source code. To select the segments to add to a batch, one spectrogram

is randomly selected. Out of the spectrogram, a random segment of 1s for Pairwise_Kldiv or 400ms for Pairwise_Lstm is selected and then added to the batch. Since a fixed number of segments are selected randomly and we increase the number of speakers to choose from, the chances of having multiple segments from the same speakers decrease. The following calculation supports this intuition:

There are 10 spectrograms for each speaker to choose from. For 100 speakers, this gives us 1000 spectrograms in total. The chances that a spectrogram of a speaker x is chosen is therefore $\frac{1}{1000} * 10 = \frac{1}{100}$. For 470 speakers it is $\frac{1}{4700} * 10 = \frac{1}{470}$ accordingly. When 100 spectrograms are randomly chosen, we can calculate the probability of a speaker being selected at least twice as $1 - (\frac{n-1}{n})^k + \frac{1}{n} * (\frac{n-1}{n})^{k-1}$ with n = number of speakers to choose from and k = how many times we choose randomly. The first term of the equation is the probability of a speaker never being selected and the second term is the probability of a speaker being selected exactly once. For 100 speakers, this gives us 63.0270% chance of speaker x being selected twice or more. With 470 speaker to choose from, the chance is 19.0115%.

Therefore, there will be less pairs where both segments belong to the same speaker when using 470 speakers. As a result, the network will often learn that embeddings of segments from different speakers must look different but it will hardly ever learn that embeddings of the same speaker should look similar. This might be the reason why both algorithms that use the pairwise KL divergence do not perform well in this experiment.

In conclusion, this experiment has indicated that the algorithms can potentially reach similar results with this experimental setting as in the setup described in the respective paper. Luvo seems to be able to even exceed its previous results in this setting. This would be a key requirement for a meaningful benchmarking of the algorithms since they would all be trained under the same circumstances. However, this experiment also indicates that some adjustments to the parameters might be necessary to reach the best possible performances of the algorithms.

6.4. Experiment 4: Short Utterances

With this experiment we test the newly implemented short utterance mode. This is needed to check the performance of the algorithms when only very short spoken sentences are available.

6.4.1. Setup and Expected Result

training set:
speakers_470_stratified

test set:
speakers_40_clustering_vs_reynolds

algorithms:

Luvo
Pairwise_Kldiv
Pairwise_Lstm

Because we could get better results in our setup when training the networks with the original configuration we further used the same setup as described in Experiment 3. We switched to the short utterance setup defined in Section 4.2.3 where each utterance only consists of one sentence per speaker.

Our expectation is that the performance of all three algorithms should drop as the clustering task gets significantly harder. There are more utterances to cluster, therefore there are also more utterances that can be clustered incorrectly. Additionally every utterance is shorter which means that it gets harder to cluster. This can intuitively be compared to how it is also harder for humans to detect who speaks if they only hear a small part of a speech instead of the full speech.

6.4.2. Results and Discussion

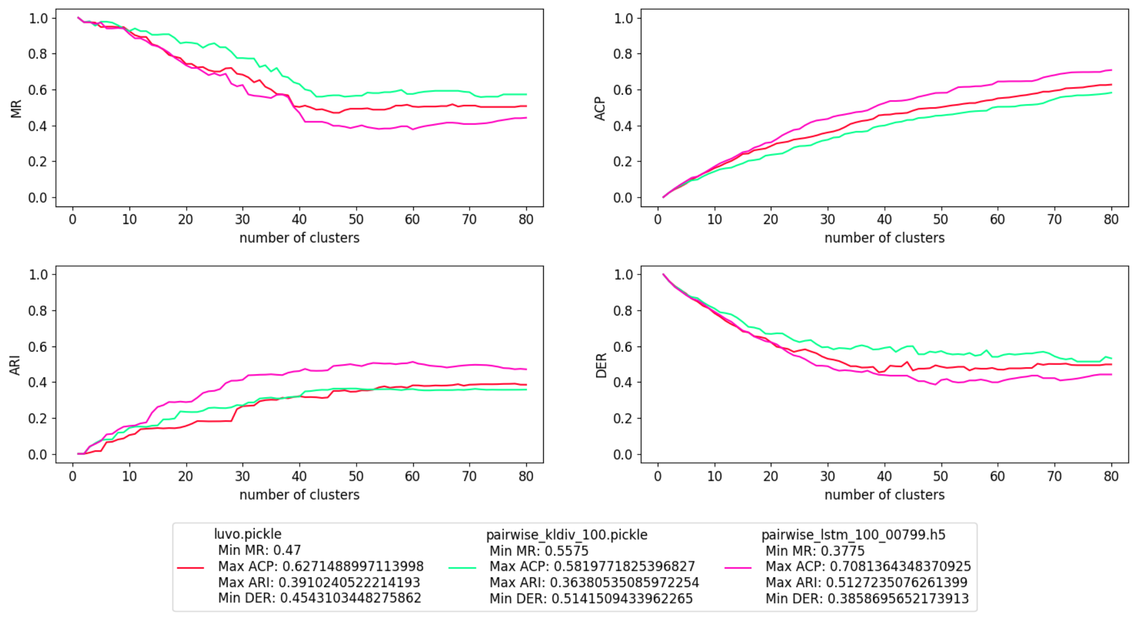

Figure 6.4.: Plot of MR, ACP, ARI and DER per cluster size for Luvo, Pairwise_Kldiv and Pairwise_Lstm in the short utterance setup.

The results of the experiment in Figure 6.4 show that all algorithms performed worse than in the previous experiment. Especially the Luvo algorithm which achieved outstanding results in Experiment 3 is now much worse as it achieved a MR of 1,25% on the longer utterances and now only achieves a MR of 47%. The Luvo algorithm is also outperformed by the Pairwise_Lstm algorithm. This shows that the Luvo network is the one that has the most problems clustering speakers when only providing short utterances. Because the two pairwise deep learning algorithms both drop similarly and the Luvo algorithm drops so significantly we can assume that the pairwise characteristic is more stable for different utterance sizes.

7. Conclusion

With this work, we are not able to definitively answer the question which deep learning approach performs best for speaker clustering. We have introduced a framework in which the approaches can be benchmarked and have proven in our experiments that the approaches still behave correctly after our changes. We have also introduced a conceptually sound experimental setup to train and evaluate the approaches under the same circumstances. Additionally, we have introduced two legacy approaches for meaningful comparisons to be able to classify the results achieved of the newly developed approaches. The experiments we conducted have shown that in order to benchmark the approaches in our setup, the approaches must first be adjusted to the new setup.

Finding Ideal Parameter Values
It is intuitively clear that using more training data should lead to better results. The experiments show that the performance of Pairwise_Lstm and Pairwise_Kldiv declined when increasing the training data to 470 speakers. However, the experiments also showed that the performance could be increased by searching for the ideal parameters to use with the adapted neural network for 470 speakers. This is also true for Luvo, although it has already exceeded the previously best performance in Experiment 3. To benchmark the approaches, the ideal parameters for the different algorithms must be found for our suggested setup.

Adjusting the Batch Size for Pairwise KL Divergence
In our last experiment, we have also shown that when trained on 470 speakers, both Pairwise_Kldiv and Pairwise_Lstm might not be trained enough to produce similar embeddings for segments from the same speaker. This could potentially have a negative influence on the performance of the approaches in our suggested experimental setup. This hypothesis could not be confirmed nor refuted within the time of this work, but should be explored as well.

Experimenting with Different Segment Sizes
The analysis of the three approaches in Section 3.7 has shown that for Pairwise_Lstm [SGHGD18] have experimented with different segment sizes to find the ideal size. For benchmarking, the ideal segment sizes for both Pairwise_Kldiv and Luvo should also be explored. Otherwise the approaches are not compared under the same circumstances. The experiment in Appendix C indicates that for the Luvo approach, this will most likely lead to better results.

8. Future Work

Since this work could not definitively benchmark the three deep learning approaches, there are some aspects to research in the future:

Further Experiment with the Current Setup
As discussed in the previous section, there might be a lot of potential in searching for the ideal parameters to train the three approaches with 470 speakers. The findings of these experiments would be vital to asses which approach is best equipped for speaker clustering.

Experiment with Segment Sizes
It was also discussed in the previous section that finding the ideal segment sizes for Luvo and Pairwise_Kldiv would be an essential contribution for benchmarking the approaches properly. In their work, [SGHGD18] have already done this for Pairwise_Lstm. Conducting experiments on the same time interval would be a promising next step to take.

Experiment with Segment Sizes
In the early stages of our work, we have found that Luvo and Pairwise_Kldiv have been build using the library theano which was discontinued in 2017. Both approaches did not work in the beginning due to bugs in the theano version that was used. If the approaches are to be used in the future, we would suggest to use an alternative library. Pairwise_Kldiv and Luvo take by far the longest to train with approximately 6 or 3 days respectively. Using a different library might also potentially lead to shorter training times which would be beneficial for future work on these approaches.

9. Acronyms

ACP	Average Cluster Purity
AI	Artificial Intelligence
ARI	Adjusted RAND Index
CNN	Convolutional Neural Network
DER	Diarization Error Rate
GMM	Gaussian Mixture Model
InIT	Institute of Applied Information Technology
KL divergence	Kullback-Leibler divergence
LSTM	Long-Short Term Memory
MFCC	Mel-Frequency Cepstral Coefficients
MR	Misclassification Rate
RNN	Recurrent Neural Network
UBM	Universal Background Model
ZHAW	Zürcher Hochschule für Angewandte Wissenschaften

Bibliography

[Bi19] Chloe Bi. The em algorithm explained. `https://medium.com/@chloebee/the-em-algorithm-explained-52182dbb19d9`, 2019. (Online; accessed 28-May-2019).

[Bru05] Derek Bruff. The assignment problem and the hungarian method. `http://www.math.harvard.edu/archive/20_spring_05/handouts/assignment_overheads.pdf`, 2005. (Online; accessed 26-April-2019).

[Con] Linguistic Data Consortium. Timit acoustic-phonetic continuous speech corpus. `https://catalog.ldc.upenn.edu/LDC93S1`. (Online; accessed 28-May-2019).

[DKD$^+$10] Najim Dehak, Patrick J. Kenny, Réda Dehak, Pierre Dumouchel, and Pierre Ouellet. Front-end factor analysis for speaker verification, 2010.

[Gal13] Olivier Galibert. Methodologies for the evaluation of speaker diarization and automatic speech recognition in the presence of overlapping speech, 2013.

[GE17] Timon Gygax and Jörg Egli. Speaker identification mit metric embeddings, 2017.

[HK15] Yen-Chang Hsu and Zsolt Kira. Neural network-based clustering using pairwise constraints. *CoRR*, abs/1511.06321, 2015.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[HVSP18] Feliks Hibraj, Sebastiano Vascon, Thilo Stadelmann, and Marcello Pelillo. Speaker clustering using dominant sets, 2018.

[KMK08] Margarita Kotti, Vassiliki Moschou, and Constantine Kotropoulos. Speaker segmentation and clustering, 2008.

[LLM16] Anthony Larcher, Kong Aik Lee, and Sylvain Meignier. An extensible speaker identification sidekit in python. International Conference on Audio Speech and Signal Processing (ICASSP), 2016. (Online; accessed 13-May-2019).

[Log00] Beth Logan. Mel frequency cepstral coefficients for music modeling. In *ISMIR*, 2000.

[LVDS16] Yanick Lukic, Carlo Vogt, Oliver Durr, and Thilo Stadelmann. Speaker identification and clustering using convolutional neural networks, 2016.

[LVDS17] Yanick X. Lukic, Carlo Vogt, Oliver Durr, and Thilo Stadelmann. Learning embeddings for speaker clustering based on voice equality, 2017.

[NH17] Savin Niederer and Benjamin Heusser. Deep learning für speaker clustering, 2017.

[Nie15] Michael A. Nielsen. *Neural Networks and Deep Learning.* Determination Press, 2015.

[NIS00] NIST. 2000 speaker recognition evaluation - evaluation plan. `https://www.nist.gov/sites/default/files/documents/2017/09/26/spk-2000-plan-v1.0.htm_.pdf`, 2000. (Online; accessed 13-May-2019).

[Ola15] Christopher Olah. Understanding lstm networks. `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`, 2015. (Online; accessed 28-May-2019).

[PP03] Massimiliano Pavan and Marcello Pelillo. Dominant sets and hierarchical clustering, 2003.

[PVG$^+$11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[Rey97] Douglas A. Reynolds. Comparison of background normalization methods for text-independentspeaker verification. `https://pdfs.semanticscholar.org/f5ad/e2e149b2d4bc0a4c679207b2bf858692af7a.pdf`, 1997. (Online; accessed 26-April-2019).

[RN10] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach.* Pearson, third edition edition, 2010. ISBN: 978-0-13-604259-4.

[RR95] Douglas A. Reynolds and Richard C. Rose. Robust text-independent speaker identification using gaussian mixture speaker models. `https://pdfs.semanticscholar.org/381a/00152b08160b0802c34eb66aa44317911089.pdf`, 1995. (Online; accessed 26-April-2019).

[SF09] Thilo Stadelmann and Bernd Freisleben. Unfolding speaker clustering potential: a biomimetic approach. *Proceedings of the 17th ACM international conference on Multimedia. ACM*, 2009.

[SGHGD18] Thilo Stadelmann, Sebastian Glinski-Haefeli, Patrick Gerber, and Oliver Dürr. Capturing suprasegmental features of a voice with rnns for improved speaker clustering, 2018.

[STS$^+$18] Thilo Stadelmann, Vasily Tolkachev, Beate Sick, Jan Stampfli, and Oliver Dürr. Beyond imagenet - deep learning in industrial practice. *Applied Data Science - Lessons Learned for the Data-Driven Business*, 2018.

[YR01] Ka Yee Yeung and Walter L. Ruzzo. Details of the adjusted rand index and clustering algorithms supplement to the paper "an empirical study on principal component analysis for clustering gene expression data" (to appear in bioinformatics), 2001.

[ZXYH11] Yu Zhang, Jian Xu, Zhi-Jie Yan, and Qiang Huo. An i-vector based approach to training data clustering for improved speech recognition. `https://pdfs.semanticscholar.org/fc31/99273223da59ccd4edbadecd864605f66139.pdf`, 2011. (Online; accessed 13-May-2019).

A. Dependency Changes

In their work, [NH17] have listed the dependencies needed to run their system. However, the system could not be run anymore with the list provided. Furthermore, after the system was running we experienced memory leaks for Luvo and Pairwise_Kldiv. This problem was due to a problem with the theano version used previously. Therefore, we provide an updated list of dependencies to use for our system:

library name	version
numpy	1.14.5
scipy	0.19.1
theano	1.0.4
scikit-learn	0.19.1
matplotlib	2.1.0
keras	2.0.6
librosa	0.5.1
lasagne	Commit a61b76f
nolearn	Commit 2ef346c
pandas	0.20.3
munkres	1.0.7
pyannote.metrics	1.8.1

B. Readme

```
# ZHAW Deep Voice
## General Information
The code is split into two branches. the reason for this
is that the I-Vector dependency is not compatible with
the deep learning dependencies (namely the sidekit
dependency does not work with theano 1.0.4. sidekit works
 with theano 0.9.0 but this version has a memory leak)

The name of the two branches are:
 - master
 - i_vector

## Using the code
### Docker Container
There are four different Dockerfiles that can be used to
create a local Docker Container:
 - Dockerfile: Contains all dependencies for running the
 deep learning approaches and GMM on the GPU cluster
 - Dockerfile-cpu: Contains all dependencies for running
 the deep learning approaches and GMM on a local computer
 - Dockerfile_ivector: Contains all dependencies for
 running I-Vector on the GPU cluster
 - Dockerfile-cpu_ivector: Contains all dependencies for
 running I-Vector on a local computer

The Docker container can be built with the batch files in
 the root directory when using Windows. They may have to
be adapted so another Container name or another
Dockerimage file is used for creating the Container.

The Docker Container can then be integrated into PyCharm
on a local computer or uploaded to Dockerhub to be
useable on the GPU cluster.

### Config file
The config file is used to set the configuration before
running an experiment. It is located in /common/config.
cfg

### Initial Setup
```

```
Before ZHAW deep voice can be used the initial setup has
to be executed to create all necessary speaker files.
First the TIMIT data set has to be located in /common/
data/training/TIMIT. In the config file the setup option
has to be set to true. Then the controller.py has to be
executed. It then creates all speaker files in /common/
data/training/speaker_pickles. This will take some time
and needs a lot of memory (16GB should be enough)

### Running an experiment
To run an experiment, simply edit the config file and
execute controller.py.

## Folder structure
On the root level there are two different folders that
contain code.

 - The common dictionary contains all code that is used
 by multiple networks
 - The networks dictionary contains a dictionary for
 every network that contains specific code only used in
 one network.

The output directory is located at /common/data/
experiments. It contains three sub directories:
 - nets: Contains the trained models
 - results: Contains the results from testing a model
 - plots: Contains the plots created from the results
```

C. Luvo Segment Size Comparison

We realized while conducting our experiments that the segment size of the Luvo algorithm has been set to 500ms. In [LVDS16] which originally implemented the Luvo algorithm it is never stated that better results are achieved when using a lower segment size. [NH17] was the only work that further changed the parameters of the Luvo algorithm. While they stated that they used the best parameters for every algorithm we could not find an evidence that the Luvo algorithm performs better with a segment size of 500ms. The purpose of this experiment is therefore to find out if the Luvo algorithm performs better with a segment size of 500ms.

C.1. Setup and Expected Result

training set:
speakers_590_clustering_without_raynolds

test set:
speakers_40_clustering_vs_reynolds

algorithms:
Luvo

The Luvo algorithm has been trained with a segment size of 1s and with a segment size of 500ms. We use the same training and test data set from [NH17] so we can compare the results.

Since [NH17] stated that they used the best parameters for every algorithm we expect that the Luvo algorithm with a segment size of 500ms shows better results. We also expect that the results are similar to the one in [NH17].

C.2. Results and Discussion

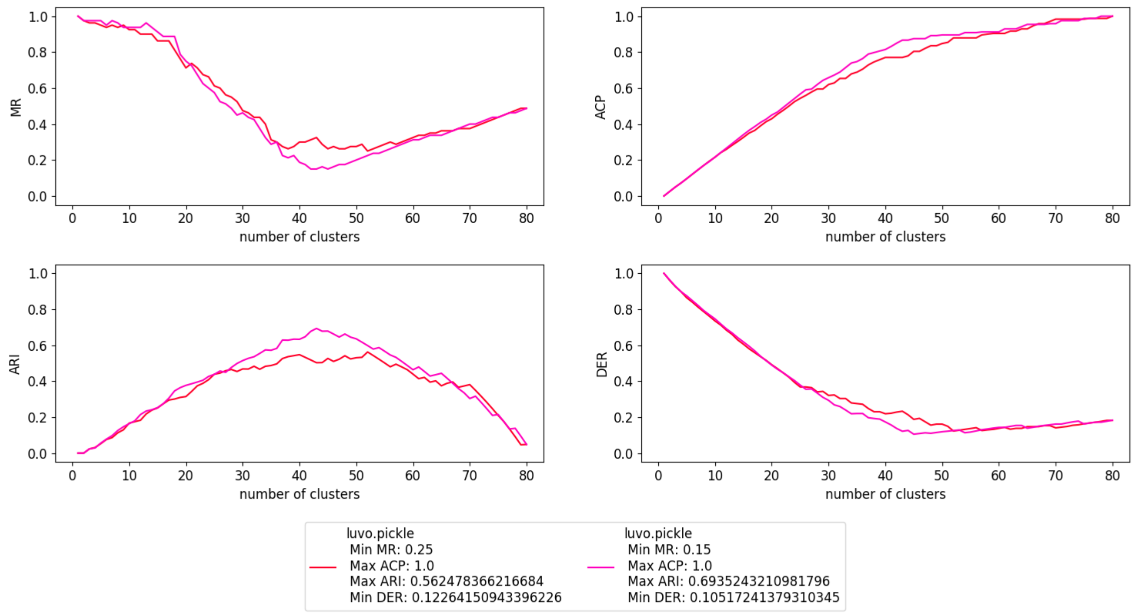

Figure C.1.: Plot of MR, ACP, ARI and DER per cluster size for the two Luvo algorithms. The red line is the result of the Luvo algorithm with a segment size of 1s, the purple line is the result of the Luvo algorithm with a segment size of 500ms.

The plots in Figure C.1 show that the Luvo algorithm with a segment size of 500ms outperforms the Luvo algorithm with a segment size of 1s. Also the results suggest that Heusser and Niederer [NH17] have been using a segment size of 500ms. Therefore we also continue to use the segment size of 500ms to be comparable to [NH17] and also because they deliver the better results.

D. Additional Clustering Algorithm

While working on this Bachelor thesis the idea of implementing an additional clustering algorithm arised. This would have given us additional information about how good the embeddings produced by the model are. With only one clustering algorithm, it is not clear if a bad result is caused by bad embeddings or if the embeddings just do not work with this specific clustering algorithm. If the same embeddings applied to a different clustering algorithm still lead to bad result, we can be almost certain that it is due to bad embeddings.

D.1. Dominant Sets

[HVSP18] showed in their work that applying Dominant Sets to speaker clustering outperforms state of the art clustering techniques in this domain. In their work they also performed the clustering on embeddings that resulted from the TIMIT data set. Therefore this clustering algorithm is a great choice to additionally implement into our code base.

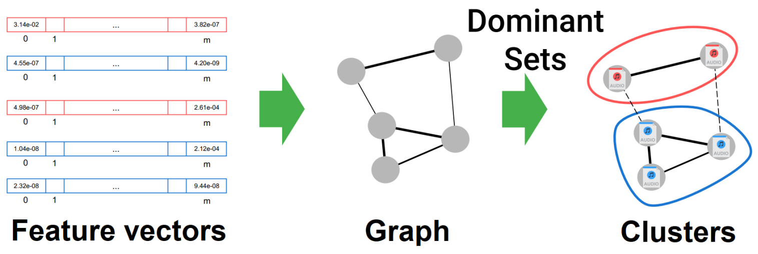

Figure D.1.: The basic idea behind Dominants Set Clustering from feature vectors to the clusters (Adapted from [HVSP18])

As displayed in Figure D.1 the basic idea behind Dominant Set Clustering is that it generalizes the clustering problem to a problem finding a maximal clique to a edge-weighted graph. The data set is represented as a undirected graph with no self loops. The nodes are the data points represented by feature vectors, the edges are relationships between two nodes and the edge weight is the similarity between two linked nodes. The goal of the clustering is now to cluster data points where the weights of the edges in a cluster is as large as possible while the weights of edges between clusters is minimized [PP03].

D.1.1. Setup and Expected Results

training set:
speakers_470_stratified

test set:
speakers_40_clustering_vs_reynolds

algorithms:
Luvo
Pairwise_Kldiv
Pairwise_Lstm

The networks used are the same as in Experiment 3. We also chose the three parameters 0.15, 0.10 and 0.20 as the cutoff value θ. $\theta = 0.15$ delivered the best results in our short test and was the best parameter in [HVSP18] for the CNN that was trained on the TIMIT data set (which is called CNN-T in [HVSP18]). We also chose nearby values to see how stable the algorithm in our case is. For ϵ we chose 1e-06 which is the default value for the parameter and also delivered the best results for the CNN-T in [HVSP18].

Our expectation is that the results should be comparable or better than the ones from Experiment 3. We also expect that the results should not vary by a large amount because in [HVSP18] the authors measured the stability of the algorithm which was really stable in their conducted experiments.

D.1.2. Results and Discussion

		θ		
		0.10	0.15	0.20
Luvo	MR	0.325	0.2375	0.625
	ACP	0.7158	0.7833	0.4224
	ARI	0.5376	0.6358	0.1598
	DER	0.2284	0.1651	0.5
Pairwise_Kldiv	MR	0.0375	0.025	0.025
	ACP	0.9583	0.975	0.975
	ARI	0.9125	0.9494	0.9494
	DER	0.0481	0.0264	0.0264
Pairwise_Lstm	MR	0.0	0.0	0.0
	ACP	1.0	1.0	1.0
	ARI	1.0	1.0	1.0
	DER	0.0	0.0	0.0

Table D.1.: Results of MR, ACP, ARI and DER per cutoff value θ for Luvo, Pairwise_Kldiv and Pairwise_Lstm using Dominant Sets.

The results of the experiment in Table D.1 show that the Pairwise_Kldiv and the Pairwise_Lstm algorithm have much better results than in Experiment 3. The Pairwise_Lstm algorithm even has the best results from all previously conducted experiments and manages to have a perfect prediction for all three chosen cutoff values. We can see that the parameters do not heavily influence these two algorithms as the results only vary by small amounts. We assume that only extreme parameter values will have a meaningful influence on the results of the algorithms as stated in [HVSP18]. In contrast the Luvo algorithm performs really poorly and also the results vary by a huge amount between the three cutoff values. This is surprising as the Luvo algorithm outperforms the other two algorithms in Experiment 3. Due to time constraints we could not further investigate the Dominant Sets approach but the results from this experiment show that further investigating into this approach can deliver interesting new insights.