University of Sheffield

Speaker Diarization System for the DIHARD Challenge

Mangesh Hambarde

Supervisor: Dr. Thomas Hain

A report submitted in fulfilment of the requirements
for the degree of MSc Computer Science with Speech and Language Processing

in the

Department of Computer Science

September 9, 2019

Declaration

All sentences or passages quoted in this report from other people's work have been specifically acknowledged by clear cross-referencing to author, work and page(s). Any illustrations that are not the work of the author of this report have been used with the explicit permission of the originator and are specifically acknowledged. I understand that failure to do this amounts to plagiarism and will be considered grounds for failure in this project and the degree examination as a whole.

Name: ____________________________________

Signature: ________________________________

Date: _____________________________________

Abstract

Speaker Diarization is commonly known as the task of finding out “who spoken when?” in an audio recording. It is an important field because it is a crucial preprocessing step for many other areas in speech technology. One of the key challenges faced in this field is how to deal with domain variation - the speaker, channel and environment variability that exists in the speech signal. DIHARD is a challenge that has been created by the diarization community to boost research so this problem can be solved. The main aim of the project is to build a complete speaker diarization system in Kaldi that works within the rules of the 2019 DIHARD challenge. Several different system configurations are explored in the project. The best system involved concatenating two different speaker embeddings into a single embedding. This resulted in a DER of 24.64%, almost 2% less compared to the baseline at 26.58%.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Speaker Diarization

Speaker diarization is commonly known as the task of finding out "who spoke when?" in an audio recording with an unknown amount of speakers. It aims to split the recording into segments according to their speaker identity. These segments can also be overlapping. It acts as an important upstream preprocessing step for several tasks in speech processing. For example, it can be used in automatic transcription services to find all spoken segments and the speaker identities for them. The segments can then be passed to an automatic speech recognition (ASR) system to recognize the words. Furthermore, this also allows speaker-adapted ASR models to be used in order to improve the accuracy of ASR.

With increase in computing power, speech processing technologies have achieved incredible advances in the past decade that were not possible earlier. This has increased interest in automatic transcription technologies that can be used to automatically index the enormous amount of audio and video information that is generated in the modern world. This creates the creation of search engines that search audio files for information, just text documents. Examples of such queries can be: "which speakers tend to dominate a conversation?", "which speakers are most likely to interrupt others?" and "fetch all segments spoken by a particular speaker". Since speaker diarization is an important part in any transcription system, there is a great deal of research interest in the area.

Diarization is not an easy problem since the output is affected by several factors like the application domain (broadcast news, meetings, telephone audio, internet audio, restaurant speech, clinical recordings etc), types and quality of microphones used (boom, lapel, far-field), inter-channel synchronization problems, overlapping speech, etc. These days, most of the research focuses on the meeting speech domain, since most problems that exist in speech recognition are encountered in this domain. The meeting scenario is thus often termed as "speech recognition complete". But focusing on only one domain can lead to problems. It makes it hard to compare diarization systems that are trained for different domains. In the worst case it also causes overfitting to the domain that they are trained on.

The DIHARD challenge was created to establish standard datasets for diarization that have a good amount of domain variability. Systems that are trained only for a single domain are expected to perform poorly on these datasets. This is where the word "hard" comes from in the name. The DIHARD datasets span several domains of speech like broadcast, meeting, telephone, restaurant, courtroom, YouTube speech etc.

1.2 Motivation and Objectives

Creating a diarization system for the DIHARD challenge can be a rewarding experience since it gives a chance to learn about state-of-the-art speaker diarization techniques. Thus the main aim of the project is to build such a system using the Kaldi toolkit [1] that works within the rules of the 2019 DIHARD challenge. Systems from last year's challenge (2018) can be used as a reference. The focus is to explore different possible configurations to get the best performing system.

1.3 Report Outline

In this chapter we briefly described the concepts in speaker diarization. Chapter 2 covers more detail on the various steps of a speaker diarization system and the Kaldi toolkit. Chapter 3 describes the structure and rules of the DIHARD challenge and the datasets involved. Chapter 4 explains how the baseline system supplied with the 2019 DIHARD challenge works. Chapter 5 describes different experiments that were performed throughout the project along with results. Chapter 6 concludes the report.

Chapter 2

Stages of Speaker Diarization

This chapter discusses the general steps of a speaker diarization system and some of the recent techniques that are commonly used these days. In all diarization systems, the basic approach is to first extract the segments containing speech from the audio, further divide the segments into subsegments consisting of only one speaker, represent each subsegment using an embedding, and cluster the subsegments together so that each cluster represents unique speaker.

2.1 Speech Enhancement

Any speech system needs to address the problem of environmental noise in audio recordings. It is important to remove as much noise as possible, but also speaker information loss should be kept to a minimum. If done right, this stage results in increased performance in subsequent stages. If not, artefacts in denoised speech reduce diarization performance. In recent years, deep learning methods have partially solved the problem of artefacts, but generalization ability in varied domains is still a problem. This stage is not mandatory, but can be helpful in certain conditions.

2.2 Feature Extraction

The first important step in any speech system is to represent the speech recording using a sequence of feature vectors. This representation is much more compact as it only uses a few thousand parameters for every second of audio, compared to 44,100 samples in a raw waveform sampled at 44.1 KHz. The most commonly used features are Mel Frequency Cepstral Coefficients (MFCCs). Speech production can be thought of as filtering of the sound produced by the vocal cords by the shape of the human vocal tract. The shape is influenced by the positions of the tongue, teeth, lips, velum etc and determines what sound comes out. Since the shape of the vocal tract is manifested in the envelope of the short time power spectrum of the speech signal, the job of the MFCCs can be thought of as to represent this envelope accurately.

As a first step, overlapping frames are extracted from the digitized signal using a sliding window. FFT is applied on each frame to get the short time power spectrum of the frame.

2.3 Speech Activity Detection

Speech activity detection (SAD) extracts the segments in the audio that contain speech, and discards the rest. This is important because speaker diarization is only concerned with assigning speaker identities to speech segments, and does not do anything with non-speech segments. It is also possible to consider all the non-speech segments to be coming from a hypothetical new speaker which has its own cluster after clustering, but that does not result in good performance as the amount of variability possible in non-speech sounds is too high. Therefore, doing a separate SAD step before diarization is the standard method.

There are two goals of a good SAD system - keep missed speech to a minimum, and keep false alarm speech to a minimum. The first causes under-clustering and might cause speech segments being ignored or detect too few speakers, while the second pollutes clusters and degrades the diarization output. If the diarization system is used as a frontend for ASR, these errors cause word deletion and word insertion errors respectively.

Since this is a binary classification problem, it can be solved by using simple thresholding on the frame energy level. The MFCC features have the log energy of the frame as the first coefficient which can be used for this. The Kaldi program `compute-vad` uses energy thresholding to do SAD.

Statistical model-based approaches are much more popular and are trained with lots of diverse external data. Typically Long Short Term Memory networks (LSTMs) or Time Delay Neural Networks (TDNNs) are used for this task.

2.4 Segmentation

The goal of the segmentation task is to further divide the speech segments found after the SAD step into smaller segments such that there are no speaker turns within any of these subsegments. Speaker turns are defined to be the points in the audio where the set of talking speakers changes. For example within a speech segment, the point where spkr1 changes into spkr2 (spkr1 finished talking and spkr2 started immediately) would be a speaker turn because the set of talking speakers changes from ¡spkr1¿ to ¡spkr2¿. Similarly, if spkr2 starts talking without waiting for spkr1 to end (causing an overlap) the set changes from ¡spkr1¿ to ¡spkr1,spkr2¿, which is also a speaker turn.

There are basically two ways to do segmentation. The first way is to try and automatically detect speaker turns and divide the segments at these points. The classical aproach for doing this uses a sliding window, and compares consecutive windows. The comparision decides whether the two windows are better accounted by two separate models (different speaker sets) or single model (same speaker set) using an emperically determined threshold. Many distance metrics exist for this decision, for example the δ Bayesian Information Criterion (BIC) metric.

The second way is to divide the segments uniformly into very small subsegments (1-2 seconds) so that it is unlikely that the set of speakers changes within that segment.

It is not clear which of these ways yields better results. Uniform segmentation approaches are reported to work better for x-vectors [2].

2.5 Clustering

As the most important step of the diarization process, clustering works on the whole audio recording and groups together segments that belong to the same speaker. In the ideal case, all the segments belong to a speaker exist in the same cluster, and the number of clusters is equal to the actual number of speakers. The clustering process needs a distance-like similarity measure to exist between pairs of segments. Each segment can be represented by a point in vector space or a statistical model. This representation acts as the speaker representation for the segment, where the similarity between any two representations is lower if the segments have speech from the same set of speakers.

Since the speaker verification and speaker recognition fields also use speaker models to capture speaker information, these models are adopted for clustering for diarization. The best performing models are outlined below.

2.5.1 Speaker Representation

Gaussian Mixture Models

Gaussian Mixture Models (GMMs) are generative models that can be used for modeling multivariate data. In GMMs, the probability of a data point is given by the weighted combination of the probabilities from multivariate Gaussian distributions having their own mean and covariance matrices. The goal is to train a GMM to represent each segment. For this, the feature vectors belonging to a segment can simply be pooled together and the GMM parameters can be learned using the Expectation-Maximization (EM) algorithm. But this is a problem because the number of feature vectors available from the segment would likely be insufficient to obtain a good estimate of the GMM parameters. This is because the number of GMMs is usually in the order of 512, 1024 or even 2048. To overcome this problem, a Universal Background Model (UBM) is trained using a large amount speech from the general population. This UBM is later adapted to each target segment using a Maximum Apriori (MAP) adaptation, resulting in an adapted GMM for each segment. Now that we have a GMM to represent each segment, we can use different statistical similarity measures that can act as a distance metric that can be used for clustering. The Kullback-Leibler (KL) divergence is a measure that estimates the distance between two random distributions. The cross likelihood ratio is given by the following.

$$CLR(S1, S2) = \log \frac{P(S1|M1)}{P(S1|M2)} + \log \frac{P(S2|M1)}{P(S2|M2)}$$

Where S_1 and S_2 are the segments that are being compared, and M_1 and M_2 are their corresponding GMMs. It can be seen that if the segments come from the same speaker, the denominators increase, and the distance decreases.

Later experiments found that only the means in the adapted GMMs carry most of the useful speaker information, the mixture weights and covariance matrices have too much variability to be of any use. Hence the means of GMMs were concatenated into a single vector called a GMM supervector and simpler distance measures like cosine distance and Mahalanobis distance were used as distance metrics for clustering.

i-vectors

I-vectors were introduced as a reduced dimension representation of the GMM supervector using factor analysis.

$$m_s = m_u + Tw_s$$

m_s and m_u are the adapted supervector for a segment S and the UBM supervector, respectively. w_s is the i-vector of the segment s. T is the “total variability matrix” which projects the supervector down to the i-vector representation. T is estimated from the training data using the EM algorithm.

x-vectors

X-vectors were introduced in [3], where a DNN is trained to discriminate between speakers and map variable-length utterances to fixed dimensional embeddings called x-vectors. Unlike the i-vector training, where the projection matrix T is learned in an unsupervised way, DNNs require speaker labels to train.

2.5.2 Agglomerative Clustering

2.5.3 Distance metrics

BIC

PLDA

2.6 Evaluation

Diarization Error Rate

Jaccard Error Rate

2.7 Kaldi toolkit

Chapter 3

DIHARD challenge setup

3.1 Task definition

3.2 Evaluation Tracks

3.3 Scoring

3.4 Datasets

Chapter 4

Baseline setup

4.1 Overview

There are three software baselines provided by the DIHARD II organizers, each for the parts of speech enhancement, speech activity detection and diarization. The speech enhancement baseline and the speech activity detection are meant to be used together in the case of system-generated SAD (tracks 2 and 4), but since we only work with reference SAD, we do not need them. Thus we will only describe the diarization baseline in the following sections.

The diarization baseline is based on the best performing submission [4] from John Hopkins University (JHU) in the previous year's DIHARD challenge (DIHARD I). There are 4 Kaldi recipes, each for an evaluation track, but we will focus only on the recipe for Track 1 since we only work with single channel audio and gold speech segmentation.

4.2 Baseline directory structure

The baseline repository is localted at `https://github.com/iiscleap/DIHARD_2019_baseline_alltracks` and has the following directory structure. Some of the irrrelevant files have been removed.

```
DIHARD_2019_baseline_alltracks/
|-- data
|   |-- final.raw
|   |-- max_chunk_size
|   |-- min_chunk_size
|   |-- plda_track1
|   |-- plda_track2
|   |-- plda_track3
|   |-- plda_track4
|-- README.md
|-- recipes
|   |-- track1
|   |-- track2
|   |-- track2_den
|   |-- track3
|   |-- track4
|   `-- track4_den
|-- scripts
|   |-- alltracksrun.sh
|   |-- flac_to_wav.sh
|   |-- make_data_dir.py
|   |-- md_eval.pl
|   |-- prepare_feats.sh
|   |-- prep_eg_dir.sh
|   `-- split_rttm.py
`-- tools
    |-- env.sh
    |-- install_dscore.sh
    |-- install_kaldi.sh
}
```

The `data` directory has pre-trained models (in Kaldi binary format) and some configuration parameters - `final.raw` is the neural network x-vector extractor, and the `plda_*` files are the PLDA backends for the 4 tracks. The `recipes` directory has the `run.sh` files for all 4 recipes, we only care about `track1`. The scripts directory has extra scripts that are needed on top of the `egs/dihard_2018` Kaldi recipe - `alltracksrun.sh` is the main diarization script, `make_data_dir.py` makes the Kaldi data directory from the DIHARD datasets (creating files like wav.scp, segments, utt2spk etc), `prep_eg_dir.sh` copies the extra files from this repository to the `egs/dihard_2018` directory, `md_eval.pl` [5] is a diarization evaluation script that was developed by NIST, and others are self-explanatory. The `tools` directory holds scripts to install Kaldi and dscore [6], which are installed in the same directory.

The baseline code modifies and reuses the `egs/dihard_2018` recipe that was checked into Kaldi by the researchers at JHU. It does this by copying over new scripts and data that is needed to the `egs/dihard_2018` directory, `cd`'ing to that directory and running the recipe from there.

We modify and add scripts in this repository so we can easily run experiments with different parameters. The `run.sh` script is modified to allow easily changing parameters to run different experiments.

4.3 Initial segmentation

The initial segmentation step is done by `make_data_dir.py`. It deals with separating speech and non-speech segments from the recording files using the reference SAD which is provided in the form of HTK label files (.lab). Each audio recording has one label file. The label file has one line for each speech segment with the format `<start-timestamp><end-timestamp>speech`.

```
0.000 3.513 speech
4.698 7.133 speech
7.377 12.826 speech
13.284 16.797 speech
17.312 21.201 speech
...
```

This results in a bunch of segments which are known to be containing only speech. These are treated as "utterances" in Kaldi terminology and act as keys in the `utt2spk`, `feats.scp` and `segments` files. These files reside in two Kaldi "data directories", one for each dev and eval.

4.4 Features

The baseline then extracts 30 dimensional MFCC features for each of the every 10 ms using a 25 ms window. It uses the standard `steps/make_mfcc.sh` Kaldi script for this. The MFCC configuration used `mfcc.conf` is given below.

```
--sample-frequency=16000
--frame-length=25 # the default is 25
--low-freq=20 # the default.
--high-freq=7600 # Nyquist (8k in this case).
--num-mel-bins=30
--num-ceps=30
--snip-edges=false
```

Later, cepstral mean and variance normalization (CMVN) with a 3 second sliding window is applied using the `apply-cmvn-sliding` Kaldi tool.

4.5 Subsegmentation

After MFCC features are ready, the utterances are uniformly divided into smaller 1.5 second subsegments with a 0.75 second overlap. This creates new Kaldi data directories (one each for dev and eval sets) with newer keys corresponding to each subsegment. An x-vector is extracted from each of these subsegments in the next step using the Kaldi binary `nnet3-xvector-compute`.

4.6 Speaker representation

The baseline extracts an 512-dimensional x-vector from each subsegment using a neural network x-vector extractor. The extractor is trained on the datasets VoxCeleb I and II, along with added augmentation. Utterances smaller than 400 frames and speakers less than 8 utterances are discarded. Since the VoxCeleb dataset does not come with gold speech segmentation, the program `compute-vad` is used with the following configuration to classify each frame into speech or non-speech.

```
--vad-energy-threshold=5.5
--vad-energy-mean-scale=0.5
--vad-proportion-threshold=0.12
--vad-frames-context=2
```

It uses simple energy-based thresholding to generate a speech segmentation. Finally there are 1,277,503 utterances spoken by 7,351 speakers that can be used for training. Although the actual number is much more because of augmentation.

The augmentation is done by additive noise (noise, music, babble) using the MUSAN dataset and reverberation using the RIR dataset. The augmentation is done because it was determined in [3] that x-vectors exploit large quantities of training data much better than i-vectors, and show a significant increase in performance.

4.7 Scoring

For scoring two x-vectors, a PLDA backend is used as a distance metric. To train the PLDA backend, x-vectors are extracted from a random subset (size 128k) of the VoxCeleb dataset. To adapt the extracted x-vectors to the DIHARD domain, they are whitened with a whitening transform learned from the DIHARD development set. The PLDA model is trained using the x-vectors and the `ivector-compute-plda` Kaldi binary.

Each pair of x-vectors within a recording is then scored using the PLDA backend by reusing `score_plda.sh` from `egs/callhome_diarization`. These scores are stored as an affinity matrix for each recording.

4.8 Clustering

The x-vectors are then clustered using agglomerative hierarchical clustering (AHC) and a parameter sweep is done on the dev set to find the threshold that maximises the DER on the dev set. This threshold is then used for clustering the x-vectors of the eval set. The `agglomerative-cluster` Kaldi binary is used for clustering.

4.9 Diarization output

The clustering output is used to generate RTTMs using the script `make_rttm.py` from `egs/callhome_diarization`. The RTTMs give a flat segmentation of the recordings with no overlap. Since the x-vectors were extracted from segments that were overlapping, care needs to be taken when two adjacent segments are assigned to a different speaker. The script places the speaker boundary midway between the end of the first segment and the start of the second segment.

Chapter 5

Experiments and Results

5.1 Baseline results

The baseline results shown in Table 5.1 have been computed by running the 2019 baseline on the datasets from the previous year's DIHARD challenge (2018). This is because it was not possible to register for DIHARD 2019 before the deadline, and thus access to 2019 datasets was denied.

Luckily the 2018 datasets were released as a part of the June LDC newsletter. Since the basic problem statement of the challenge remains the same as last year, the last year's datasets can still be used without any trouble. But this unfortunately means that it is no longer possible to verify the computed baseline scores with official scores, because the official scores only exist for the 2019 datasets. The only useful hint was found on the webpage at [7], which mentions a rough score of 20.71 on the 2018 development set.

Dataset	DER (%)	Speaker error (%)	JER (%)
dev	19.96	11.62	53.92
eval	26.58	17.05	59.44

Table 5.1: Baseline scores.

The results already seem pretty good, considering that the system is not very complex. The best JHU system in DIHARD 2018 had an eval DER of 23.73%, and that included doing Variational Bayes refinement as an extra step.

It is also important to mention that Missed Speech and False Alarm are not mentioned in any of the following experiments, because they are always constant (since we work with reference SAD only). Missed speech is always 8.34% for the dev set and 9.52% for the eval set. False alarm is always zero for both, as expected. Missed speech should also be zero from the same reason, but it is not because the reference segmentation, which is in the form of HTK label files, have been created by merging overlapping segments. This causes the amount of speech in the label files to be 8.34% and 9.52% less than the RTTM files.

5.2 Using Existing Pre-trained Models

The first set of experiments was to see how certain pre-trained models freely available on the Internet perform.

5.2.1 Kaldi VoxCeleb x-vector model

This model was downloaded from the Kaldi models webpage [8], and closely follows the recipe in [3]. The recipe used for training is available in Kaldi at `egs/voxceleb/v2`. Similar to the baseline recipe, this model is also trained on a combination of VoxCeleb I and II along with augmentation. Both produce 512-dimensional x-vectors. There are two differences though:

- The PLDA backend included in this model is trained on the whole training data, which consists of 1,276,888 utterances. The baseline PLDA backend is trained on a subset where each segment is at least 3 seconds long.
- The PLDA backend here is trained on 200 dimensional x-vectors, which are produced after LDA. The baseline PLDA backend is trained directly on 512-dimensional x-vectors.

Dataset	DER (%)	Speaker error (%)	JER (%)
dev	22.86	14.52	51.74
eval	26.42	16.89	55.55

Table 5.2: Scores with Kaldi VoxCeleb x-vector model.

This has already produced a small improvement over the baseline. This could be possibly due to larger amount of training data used to train the PLDA backend, but more likely is due to indeterminism.

5.2.2 Kaldi VoxCeleb i-vector model

This model was also downloaded from the same Kaldi models webpage as the previous recipe, and closely follows the recipe in [3]. The recipe used for training is available in Kaldi at `egs/voxceleb/v1`. Similar to the x-vector recipe, the model is also trained on a combination of VoxCeleb I and II, but without augmentation. The UBM is trained on 2048 gaussians using all the training utterances. The i-vector extractor is trained using the longest 100k utterances and produces 400-dimensional i-vectors. I-vectors are extracted for all the training utterances, reduced to 200 dimensions using LDA, and then used to train a PLDA backend.

Dataset	DER (%)	Speaker error (%)	JER (%)
dev	26.18	17.83	59.81
eval	32.03	22.51	65.22

Table 5.3: Scores with Kaldi VoxCeleb i-vector model.

Clearly, this is much worse than the x-vector model trained on the same VoxCeleb data without augmentation. Augmentation is not used because [3] talks about i-vectors not being able to use additional data effectively, unlike x-vectors.

5.3 Training Custom Models

Training our own models was considered because that would allow models to be trained on in-domain data (the DIHARD development set). The amount of data available in the dev set is relatively small, only 19 hours, so we do not expect great results. The recipes in `egs/voxceleb` were used as a starting point for both i-vector extractor and x-vector extractor training.

5.3.1 Training with DIHARD development set

The following results in Table 5.4 were obtained by training an x-vector model on the DIHARD development set. The reference RTTM files for the dev set were used to generate 28241 training utterances from 221 speakers. The recipe imposes a minimum feature length of 400 frames and a minimum 8 utterances per speaker, so after filtering only 2726 utterances from 90 speakers were used to train with. This meant that the neural network had 90 output nodes. The embedding layer had 512 dimension. The PLDA backend was trained from x-vectors extracted from the whole dev set and reduced to 200 dimensions using LDA.

Dataset	DER (%)	Speaker error (%)	JER (%)
dev	41.35	33.00	74.60
eval	43.16	33.64	75.96

Table 5.4: Scores with x-vector model trained on DIHARD dev.

These seem to be pretty bad, but they align with the known fact that x-vectors perform poorly on small amounts of data. As an additional experiment, augmentation was applied to the dev set. The augmentation was done similar to what the baseline does: 4 variants of the training set were created (reverb, noise, babble, music) and added to the original set, multiplying the number of utterances by 5. This resulted in 141205 utterances from 221 speakers, reduced to 13630 utterances from 90 speakers after filtering. This increased amount of training data resulted in a small increase in performance, as given in Table 5.5.

Dataset	DER (%)	Speaker error (%)	JER (%)
dev	35.54	27.20	76.56
eval	39.43	29.91	78.65

Table 5.5: Scores with x-vector model trained on DIHARD dev + augmentation.

The result of training x-vector extractor on in-domain data did not increase the perfomance beyond the baseline, despite adding augmentation.

Next, an i-vector model was trained on the dev set. Surprisingly, it performed much better, as given in the Table 5.6.

Dataset	DER (%)	Speaker error (%)	JER (%)
dev	25.22	xx.xx	58.58
eval	34.61	xx.xx	65.69

Table 5.6: Scores with i-vector model trained on DIHARD dev.

This shows that the i-vector model without augmentation performed better than the x-vector model with augmentation, given the amount of data is small.

The i-vector training on the the non-augmented dev set took 17 hours on a 32-core machine, with near 100% CPU usage all the time. The i-vector training was not attempted with an augmented dev set, which would be 5 times bigger.

5.3.2 Training with combination of Voxceleb and DIHARD development set

For the next set of experiments the amount of training data was increased by adding data from VoxCeleb I. VoxCeleb II was not used because it is 7 times bigger than VoxCeleb I, making the training set too big, especially for i-vector training. There are 153,516 utterances from 1,251 speakers in VoxCeleb I, so this increases the total amount of training data significantly. All the parameters of the training remained similar.

The results of the x-vector model trained on the combination of VoxCeleb I and DIHARD dev set are given in Table 5.7.

Dataset	DER (%)	Speaker error (%)	JER (%)
dev	23.45	15.11	56.89
eval	29.44	19.92	61.37

Table 5.7: Scores with x-vector model trained on combination of VoxCeleb I and DIHARD dev.

The results of the i-vector model trained on the combination of VoxCeleb I and DIHARD dev set are given in Table 5.8.

Dataset	DER (%)	Speaker error (%)	JER (%)
dev	25.15	16.81	56.78
eval	31.61	22.08	60.74

Table 5.8: Scores with i-vector model trained on combination of VoxCeleb I and DIHARD dev.

5.3.3 Lessons Learnt From Training

Unlike i-vector training which is unsupervised, x-vector training needs speaker labels since the neural network is trained to discriminate between the speakers [9].

5.4 Feature concatenation

5.5 Tuning Hyperparameters

5.5.1 Vector Dimensionality

5.5.2 Segment Length and Overlap

5.5.3 Clustering Threshold

5.5.4 Number of UBM Gaussians

5.6 Discussion of results

5.6.1 By amount of speaker data

5.6.2 By recording

5.6.3 By utterance duration

5.6.4 By number of speakers

Chapter 6

Conclusions

Bibliography

[1] D. Povey, A. Ghoshal, G. Boulianne, L. Burget, O. Glembek, N. Goel, M. Hannemann, P. Motlicek, Y. Qian, P. Schwarz *et al.*, "The kaldi speech recognition toolkit," in *IEEE 2011 workshop on automatic speech recognition and understanding*, no. CONF. IEEE Signal Processing Society, 2011.

[2] J. Patino, H. Delgado, R. Yin, H. Bredin, C. Barras, and N. W. Evans, "Odessa at albayzin speaker diarization challenge 2018." in *IberSPEECH*, 2018, pp. 211–215.

[3] D. Snyder, D. Garcia-Romero, G. Sell, D. Povey, and S. Khudanpur, "X-vectors: Robust dnn embeddings for speaker recognition," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2018, pp. 5329–5333.

[4] G. Sell, D. Snyder, A. McCree, D. Garcia-Romero, J. Villalba, M. Maciejewski, V. Manohar, N. Dehak, D. Povey, S. Watanabe *et al.*, "Diarization is hard: Some experiences and lessons learned for the jhu team in the inaugural dihard challenge," in *Proc. Interspeech*, 2018, pp. 2808–2812.

[5] "md-eval.pl." [Online]. Available: https://web.archive.org/web/20061001115045/http://www.nist.gov/speech/tests/rt/rt2006/spring/code/md-eval-v21.pl

[6] N. Ryant, "dscore." [Online]. Available: https://github.com/nryant/dscore

[7] "Dihard ii unofficial repository." [Online]. Available: http://archive.is/Ha8x3

[8] "Kaldi models webpage." [Online]. Available: http://kaldi-asr.org/models.html

[9] T. Stafylakis, J. Rohdin, O. Plchot, P. Mizera, and L. Burget, "Self-supervised speaker embeddings," *arXiv preprint arXiv:1904.03486*, 2019.