Abstract

*pyannote.audio* is an open-source toolkit written in Python for speaker diarization. Version 2.1 introduces a major overhaul of *pyannote.audio* default speaker diarization pipeline, made of three main stages: speaker segmentation applied to a short sliding window, neural speaker embedding of each (local) speakers, and (global) agglomerative clustering. One of the main objectives of the toolkit is to democratize speaker diarization. Therefore, on top of a pretrained speaker diarization pipeline that gives good results out of the box, we also provide a recipe that practitioners can follow to improve its performance on their own (manually annotated) dataset. It has been used for various challenges and reached 1st place at Ego4D 2022, 1st place at Albayzin 2022, and 6th place at VoxSRC 2022.

**Index Terms:** speaker diarization, open source, toolkit

1. Introduction

*pyannote.audio* is an open-source toolkit written in Python for speaker diarization. Version 2.1 introduces a major overhaul of *pyannote.audio* default speaker diarization pipeline (with respect to branch 1.x) and is very similar in spirit to the line of work developed by Kinoshita at NTT [1, 2] that “integrates clustering-based and end-to-end neural network-based diarization approaches into one framework”. Hence, the proposed approach is composed of three main stages: speaker segmentation applied to a (local) sliding window, neural speaker embedding of each (local) speakers, and (global) agglomerative clustering. Section 2 goes into details about the proposed approach but we highlight here the main differences with [1, 2].

First, local neural speaker diarization is applied to much shorter overlapping windows (5s with a 500ms step) than the original one (30s with 30s step, i.e. no overlap), making the whole task much easier to solve:

- the upper bound on the number of speakers is smaller and the training sequences are shorter, hence reducing the computational and memory cost of training such networks;
- the use of strongly overlapping windows can be seen as test time augmentation, leading to better speaker segmentation and denser (hence easier to cluster) speaker embeddings.

Second, one of the main advantages of the joint (diarization + embedding) approach used in [1, 2] lies in embeddings that are both overlap-aware and computed from longer audio (hence more reliable). Despite relying on two separate networks applied in cascade (first segmentation, then embedding), we claim in Section 2.2 that our speaker embeddings enjoy the same properties. Training speaker embedding networks is notoriously data-hungry and it is not always possible for practitioners to gather a training dataset that both contains a large set of conversations as well as speaker labels which are consistent across conversations. Therefore, we claim that using two different networks makes the whole approach easier to adapt to a particular dataset. On top of a pretrained speaker diarization pipeline that gives good results out of the box, we also provide a set of recipes that practitioners can choose from, depending on the size of their (manually annotated) dataset.

2. Principle

Figure 1 depicts the manual speaker diarization of a 30s conversation between two speakers that we will use throughout the paper for illustration purposes.

![Figure 1: Expected speaker diarization output of the sample conversation used throughout this paper.](image)

2.1. Local neural speaker segmentation

The first step consists in applying the end-to-end neural speaker segmentation model introduced in [3] using a sliding window of 5s with a step of 500ms. Figure 2 illustrates the output of this stage on the 30s sample whose manual annotation is depicted in Figure 1.

![Figure 2: Local neural speaker segmentation. For each step of the 5s window and each one of \(K_{\text{max}} = 3\) speakers, the segmentation model outputs the probability of the speaker being active every 16ms. For readability, we use a sliding step of 2s and plot (otherwise overlapping) windows on three rows, but the actual practical step is 500ms.](image)

At this point, there is no guarantee that the same (local) speaker is consistently assigned to the same (global) speaker index. Since the speaker segmentation model has been trained in a permutation-invariant manner and is limited to at most \(K_{\text{max}} = 3\) active speakers, a particular speaker might be assigned two different indices in two different windows as can be observed in Figure 2 around \(t = 16s\).

A binarization step is further applied using a threshold \(θ ∈ [0, 1]\), which constitutes the first hyper-parameter of the

\[
θ = \frac{1}{K_{\text{max}}} \sum_{k=1}^{K_{\text{max}}} P(k) \quad \text{with} \quad P(k) = \begin{cases} 1 & \text{if} \ k \text{is active at step } t \text{ and the last time a } k \text{ was active was at step } t \text{ with } t < t - \text{step} \text{ of } \text{segmentation model} \text{ step size} \text{ or } K_{\text{max}} \text{ active speakers}, \\ 0 & \text{otherwise.} \end{cases}
\]
proposed speaker diarization pipeline. The effect of this binarization step on the 30s audio sample is depicted in Figure 3.

2.2. Local speaker embedding

The second step consists in extracting $K_w$ speaker embeddings for each window $w$: exactly one embedding per speaker who is active within the window $w$. Therefore, the number of speaker embeddings may vary depending on the window $w$. For instance, window $w_{ij} = [t \to t + 5]$ may contain $K_w = 0$ speaker like in $w_{ij} = [0 \to 5]$ (because no speaker ever passes the segmentation threshold $\theta$), $K_w = 1$ speaker like in $w_{22}$, or $K_w = 2$ speakers like in $w_{16}$ (surrounded in red in Figure 3).

As depicted by the gray overlay in Figure 4, speakers may overlap partially within the considered window. To account for this possibility, the embedding of speaker $k$ is computed from the concatenation of audio samples during which (1.) speaker $k$ is active and (2.) other speakers $k' \neq k$ are inactive. This is similar in spirit to overlap-aware speaker embeddings in [4].

Compared to the standard approach that consists in extracting exactly one speaker embedding using a short (typically 1 or 2 seconds) periodic window [5], the proposed speaker embeddings are expected to be more reliable for two main reasons:

- they are extracted from audio excerpts that only contain speech samples from one single speaker while the standard approach may extract speaker embeddings from a mixture of speakers (and non-speech);
- they are extracted from potentially longer audio excerpts (up to 5s in case a speaker speaks during the whole window $w$) while the standard approach is limited to 1 or 2 seconds.

The main drawback of this approach is that it depends on the upstream speaker segmentation model whose errors could lead to degraded speaker embeddings.

2.3. Global agglomerative clustering

The third step consists in clustering the resulting set of speaker embeddings in order to assign each local speaker to a global cluster, as depicted by colors in Figure 5.

Although spectral clustering [6] and variational Bayesian hidden Markov models [5] have been the preferred clustering techniques in recent speaker diarization literature [7], the proposed pipeline relies on a classical agglomerative hierarchical clustering with centroid linkage (also known as the UPGMC algorithm) for two main reasons:

- the latter only introduces a second hyper-parameter (the distance threshold $\delta$ used as stopping criterion of the agglomerative clustering process) while both spectral clustering and variational Bayesian hidden Markov models rely on at least a couple of hyper-parameters;
- while variational Bayesian hidden Markov models (and, to a lesser extent spectral clustering$^1$) expects that speaker embeddings are ordered chronologically with a strict periodicity (e.g. one embedding every second), the speaker embedding process introduced in Sections 2.1 and 2.2 cannot guarantee these properties because a variable number (zero, one, or more) of speaker embeddings may be extracted every 500ms (or whichever step is used by the 5s sliding window).

The choice of centroid linkage over variants (such as average, single, complete, or Ward linkage) derives from the fact that the former consistently outperforms the latter on every single validation sets later discussed in Section 3 (the runner-up being the more common average linkage).

2.4. Final aggregation

The fourth and final step aims at aggregating the clustered local speaker segmentation into an actual speaker diarization

$^1$when combined with the critical step of Gaussian blur refinement of the Laplacian matrix used for auto-tuning [6]
output. Depicted in Figure 6, it can be summarized as follows (from top to bottom):

1. estimating the instantaneous (i.e. for each frame $f$) number of speakers $K(f)$, by summing the binary local speaker segmentation obtained in Section 2.1 and Figure 3 and averaging over overlapping windows (rounding to closest integer);
2. estimating the instantaneous score of each cluster by summing the clustered local speaker segmentation obtained in Section 2.3 and Figure 5 over overlapping windows;
3. selecting the $K(f)$ (given by step 1) clusters with highest instantaneous score (step 2) and converting from discrete frame indices to the temporal domain;
4. filling within-speaker gaps shorter than a (usually short) predefined duration $\Delta$.

The last step is optional as the value of $\Delta$ usually depends more on the instructions given to the pool of human annotators than to the data itself. For instance, DIHARD evaluation plan says that “small pauses [shorter than] 200 ms by a speaker are not considered to be segmentation breaks and should be bridged into a single continuous segment” [8]; VoxConverse guidelines say that “[s]peech segments are split when pauses are greater than [250 ms]” [9]; the Albayzin 2022 evaluation plan goes even further by requesting that “[c]onsecutive segments of the same speaker with a silent [sic] of less that [sic] 2 seconds […] are considered as a single segment” [10].

3. Reproducible benchmark

Despite the availability of several benchmarking initiatives (such as DIHARD, VoxSRC, or Albayzin challenges, whose organizers are heartily thanked), it remains very difficult to gauge the many speaker diarization approaches proposed by the research community, for various reasons:

- Despite a growing number of freely available datasets such as AISHELL-4 [11], Albayzin/RTVE [10], AliMeeting [12], AMI [13], VoxConverse [9], Ego4D [14], or This American Life [15], some papers only report results on a limited set of datasets either behind paywalls (such as CALL-HOME [16], DIHARD [17], or REPERE [18]), on purely synthetic datasets, or even private in-house datasets – effectively preventing others (and newcomers in particular) from comparing their approach to the so-called state-of-the-art.
- Two papers reporting results on the same dataset often use different experimental protocols without even noticing. For instance, they might use a slightly different test set, different versions of the gold standard [5], different configuration of the reported diarization error rate (e.g. with or without forgiveness collars), or different assumption about the unrealistic availability of an oracle voice activity detector.

We apologize for the tone of the above rant. The objective was to convince the reader that they should at the very least share the actual output of their proposed approaches (ideally in de facto standard RTTM format) to solve (part of) those problems. We go one step further in Figure 7 by providing open source code to produce those RTTMs and therefore allow evaluating the approach on any other (possibly private) dataset.

The leftmost part of Table 1 summarizes the performance of these few lines of code. There, the processing is fully automatic (no oracle voice activity detection, no oracle number of speakers, no fine-tuning of the internal models nor tuning of the pipeline hyper-parameters to each specific dataset) with the least forgiving diarization error rate (DER) setup (no forgiveness collar, evaluation of overlapped speech). Unless stated otherwise in the first column, we report results on the official test sets of 11 benchmarks. We claim state-of-the-art performance on AISHELL-4 [11], AMI headset mix and array/channels [13], REPERE phase2 [18], Albayzin RTVE 2022 [10], Ego4D [14], and This American Life [15]. Precomputed RTTMs are available for download at hf.co/pyannote/speaker-diarization/tree/v0.1.1.

Using one Nvidia Tesla V100 SXM2 GPU (for neural inference described in Sections 2.1 and 2.2) and one Intel Cascade Lake 6248 CPU (for the clustering and aggregation described in Sections 2.3 and 2.4), the proposed pipeline is 40 times faster than real time, with most of the time spent in the speaker embedding extraction step. In particular, all experiments reported in Table 1 relies on the implementation of ECAPA-TDNN [22] available in SpeechBrain [23] because it was found to outperform three open-source alternatives. For instance, on VoxConverse v0.3, the fine-tuned pipeline reaches DER $= 14.9\%$ with the xvector implementation available in pyannote.audio [20], 12.0% with NeMo’s TitaNet [24], 10.8% with RawNet3 [25], and 10.7% with SpeechBrain’s ECAPA-TDNN.

4. Recipe

While the leftmost part of Table 1 reports performance of the pretrained speaker diarization pipeline (with default hyper-parameters and default internal models), this section provides a recipe to adapt it to a particular target domain and (hopefully) get better performance. Depending on the number and duration of labeled conversations, the practitioner may either focus on optimizing hyper-parameters ($\theta, \delta$, and $\Delta$, introduced in Sections 2.1, 2.3, and 2.4 respectively), additionally fine-tune the internal speaker segmentation model, or both. Fine-tuning speaker embedding might also be an option in case even more data is available for a particular domain but this is out of the scope of both this paper and pyannote.audio (since we rely on external libraries for this model).

4.1. Optimizing pipeline hyper-parameters

In case a small development set of labeled conversations is available, optimizing pipeline hyper-parameters (with the few lines of code in Figure 8) may lead to significant performance improvement. Datasets listed in Table 1 are split into two groups: in-domain datasets whose training subsets have been used to train the underlying segmentation model (available at  

```python
# install pyannote.audio
pip install pyannote-audio==2.1.1
```

```python
# load pretrained pipeline from pyannote.audio import Pipeline pipeline = Pipeline.from_pretrained( "pyannote/speaker-diarization")
```

```python
# apply pipeline and dump RTTM
diarization = pipeline("audio.wav")
with open("audio.rttm", "w") as f:
diarization.write_rttm(f)
```

Figure 7: From zero to RTTMs with pyannote.audio
### Table 1: Performance of the (default, optimized, and fine-tuned) pipelines on 11 different benchmarks. The grey background marks the best results for each dataset as well as those less than 5% worse relatively. DER stands for diarization error rate, which is the sum of two terms: CONF for speaker confusion rate, and FA+MISS for false alarm and missed detection rates. We also report the scale of development (for optimizing hyper-parameters, in number of files) and training sets (for fine-tuning the segmentation model, in number of hours) as well as the best baseline we could find in the literature as of February 2023. No comparable baseline was found for VoxConverse (either because of slightly different test sets or different metric configuration) and Albayzin/RTVE2022 (because the test set labels have only been released very recently).

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Default pipeline</th>
<th>Dev. + optimized hyper-parameters</th>
<th>Train + finetuned segmentation model</th>
<th>Baseline</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DER% FA+MISS% CONF%</td>
<td>DER% FA+MISS% CONF%</td>
<td>DER% FA+MISS% CONF%</td>
<td>DER% Source</td>
</tr>
<tr>
<td>DIHARD 3 fall</td>
<td>[17]</td>
<td>[17]</td>
<td>[17]</td>
<td>[17]</td>
</tr>
<tr>
<td>REPERE phase 2</td>
<td>[18]</td>
<td>[18]</td>
<td>[18]</td>
<td>[18]</td>
</tr>
<tr>
<td>VoxConverse v0.3</td>
<td>[9]</td>
<td>[9]</td>
<td>[9]</td>
<td>[9]</td>
</tr>
</tbody>
</table>

Figure 8: Optimizing hyper-parameters with pyannote.pipeline

```python
# load dataset with pyannote.database from pyannote.database import get_protocol dataset = get_protocol(...) dev_files = list(dataset.development())

# optimize hyper-parameters with pyannote.pipeline from pyannote.pipeline import Optimizer optimizer = Optimizer(pipeline) optimizer.tune(dev_files)

# apply optimized pipeline diarization = optimizer.best_pipeline("audio.wav")
```

main (which is equivalent to not filling any intra-speaker gaps). This is to be compared with the following optimal values for $\Delta$ when the pipeline relies on the pretrained speaker segmentation model: 10ms for AISHELL-4, 400ms for REPERE and VoxConverse, 500ms for AliMeeting, 1.5s for Albayzin, or even 2s for The American Life. In other words, fine-tuning the segmentation not only improves the performance but also reduces the dimensionality of the hyper-parameter search space, from 3 ($\delta$, $\theta$, and $\Delta$) to only 2 hyper-parameters ($\delta$ and $\theta$).

## 5. Conclusion

We introduced version 2.1 of pyannote.audio open source speaker diarization pipeline, evaluated its performance on a large collection of benchmarking datasets, and described a recipe that practitioners can follow to make the most of their own labeled data and adapt the pretrained pipeline to their particular use case. This recipe has been used to reach 6th place at VoxSRC 2022 challenge, 1st place at Ego4D 2022 challenge, and 1st place at Albayzin 2022 challenge. The source code, pretrained models and expected outputs are openly shared with the community at github.com/pyannote/pyannote-audio.

This work was granted access to the HPC resources of IDRIS under the allocations AD011012177R1 and AD011012177R2 made by GENCI.
6. References


