Speaker Diarization Report

Working Code

There is a `README.md` with a detailed explanation for using the codebase.

To install the environment

```
conda create --prefix ./augnito python=3.9
conda activate ./augnito
./augnito/bin/python3 -m pip install -r requirements.txt
```

To install and prepare the dataset

```
git clone https://github.com/babylonhealth/primock57
cd primock57/scripts/
bash mix_audio.sh
```

make the following changes in `textgrid_to_transcript.py` (inside `primock57/scripts/`)

```
# line 15, from this
return [f"{u['speaker']}: {strip_transcript_tags(u['text'])}"
            for u in combined_utterances]
# to
return [f"{u['from']}::{u['to']}::{u['speaker']}::{strip_tran
            for u in combined_utterances]
```

```
python3 textgrid_to_transcript.py --transcript_path ../transc
cd ../../ # go to home dir
python3 transcript_to_timestamps.py # to convert the joined_t
```

> **NOTE:** Doctor is SPEAKER_00 and patient is SPEAKER_01 in the timestamps

To prepare `pyannote` dataset

We need to create a `pyannote.dataset` using the above dataset for finetuning the model.

```
python3 make_pyan_database.py # will create necessary files i
```

Create a file `database.yml` in the home dir, and add the following

```
Databases:
  Primock: primock57/output/mixed_audio/{uri}.wav

Protocols:
  Primock:
    SpeakerDiarization:
      full:
        train:
            uri: pyan_db/train_list.txt
            annotation: pyan_db/rttms/train_{uri}.rttm
            annotated: pyan_db/uems/train_{uri}.uem
        development:
            uri: pyan_db/dev_list.txt
            annotation: pyan_db/rttms/dev_{uri}.rttm
            annotated: pyan_db/uems/dev_{uri}.uem
        test:
            uri: pyan_db/test_list.txt
            annotation: pyan_db/rttms/test_{uri}.rttm
            annotated: pyan_db/uems/test_{uri}.uem
```

To check if the `pyannote.dataset` was created properly

```
python3 check_protocol.py
```

Codes: scripts and notebooks

- `pyan_finetune.ipynb` has the code to finetune the SpeakerDiarization model using `pyannote/speaker-diarization-3.1` and `pyannote/segmentation-3.0`. It uses the created `database.yml` `pyannote.dataset` to do this.

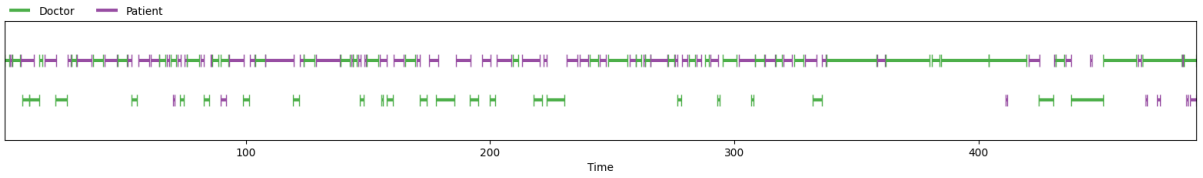
- `pyan_inf.ipynb` will run inference on the dataset using both the pretrained and finetuned models and create the output directories `timestamps_pyan_diar_pretrained` and `timestamps_pyan_diar_pretrained`.
- `nemo_inf.ipynb` will run inference on the dataset using pretrained `marblenet-vad`, `titanet_large-speaker_embedder` and creates the output directory `timestamps_nemo_pretrained`.
- `der.ipynb` has the code to calculate Diarization Error Rate (DER) for all the models used.
- `eda_pri.ipynb` has the code to perform the EDA on the dataset.
- `lightning_logs/version_0` has the checkpoint containing the fine-tuned `pyannote/speaker-diarization-3.1` model.

EDA (Exploratory Dataset Analysis)

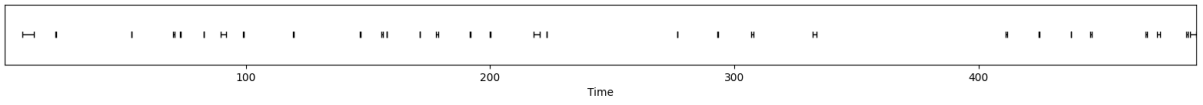
The dataset used is **primrock57** which consists of 57 mock medical primary care consultations held over 5 days by 7 Babylon clinicians and 57 Babylon employees acting as patients, using case cards with presenting complaints, symptoms, medical & general history etc.

We can use the helper scripts to get the combined audio and transcript. The transcripts contain the start, end, speaker id and text of several utterances.

I used `pyannote.core.annotation.Annotation` to visualize the transcripts.



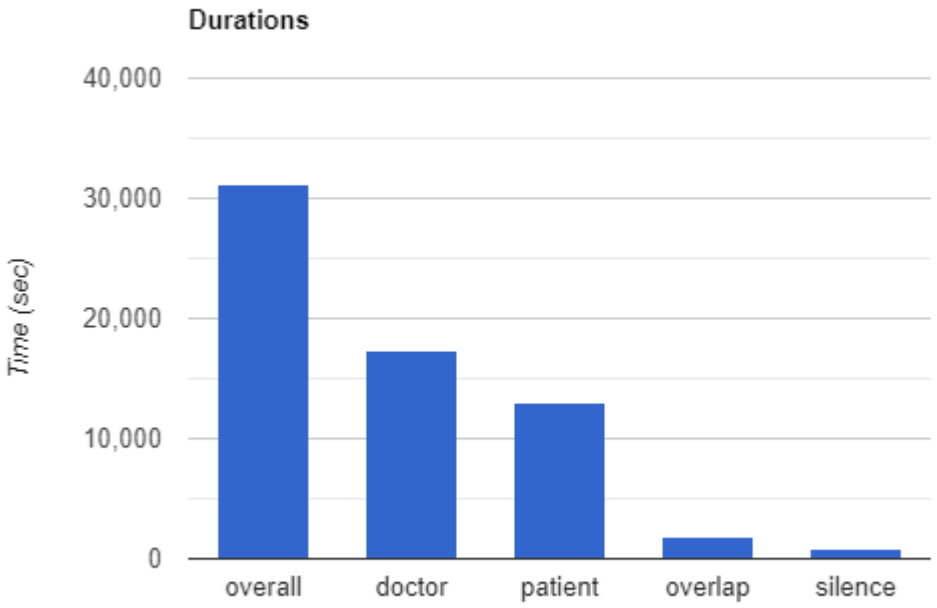
In a given audio, the **Doctor** and **Patient** are talking to each other as can be seen above.



There is a significant overlap as well, which can be seen above.

There are a total of 57 files, comprising 31086 seconds where the doctor is speaking for 17346 seconds, and the patient is speaking for 12935 seconds. There is an

overlap of 1786 seconds and a silence of 803 seconds.



The dataset is split into train set (80%), dev set (10%) and test set (10%) for fine-tuning and evaluating the models.

Speaker Diarization

Speaker diarization automatically partitions multi-speaker audio recordings into segments associated with individual speakers. It tackles the "who spoke when?" question, offering valuable insights in various fields.

Diarization algorithms rely on feature extraction, speaker clustering, segmentation, and increasingly, machine learning models. Challenges include overlapping speech, noise, and speaker similarity. Research efforts focus on deep learning integration, unsupervised learning, and speaker identification.

Applications range from meeting summarization and media monitoring to forensic investigations and accessibility enhancements. As the field evolves, expect a deeper understanding of conversations, efficient information access, and exciting new possibilities across diverse domains.

Approaches

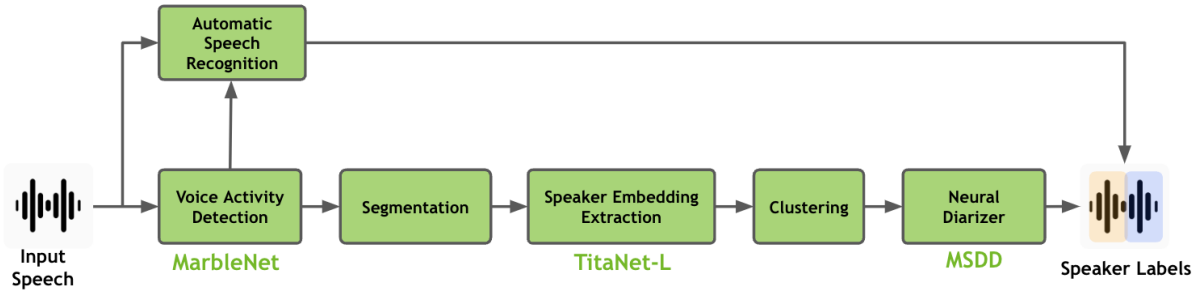
Pyannote-audio

pyannote.audio is an open-source toolkit written in Python for speaker diarization. Based on PyTorch machine learning framework, it comes with state-of-the-art pretrained models and pipelines, that can be further finetuned to custom data for even better performance. It is a three-stage pipeline comprising speaker segmentation, neural embedding, and agglomerative clustering.

- **Strengths:**
  - Lightweight and easy to use.
  - Actively developed with a strong research community.
  - Performs well on low-quality audio.
  - Offers diverse diarization methods and customization options.
- **Weaknesses:**
  - May not achieve the same accuracy as NeMo on high-quality recordings.
  - Primarily focused on speaker diarization, lacks broader NLP capabilities.
  - Might require more effort for fine-tuning and optimization.

NeMO

In NeMO, the speaker diarization system consists of Voice Activity Detection (VAD) model to get the timestamps of audio where speech is being spoken ignoring the background and speaker embeddings model to get speaker embeddings on segments that were previously time stamped. These speaker embedding vectors are then grouped into clusters and the number of speakers is estimated by clustering algorithm. Finally, based on the speaker profiles created from clustering results, neural diarizer generates speaker labels including overlap speech.



- **Strengths:**

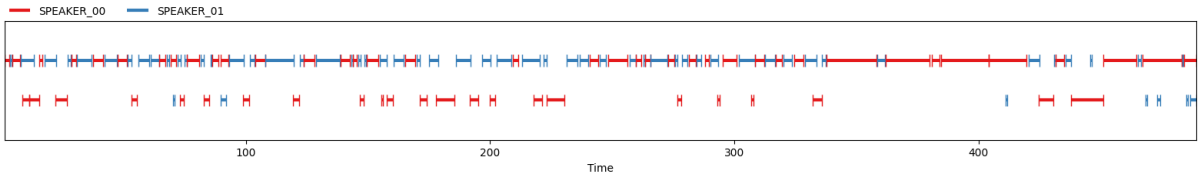
- Powerful deep learning framework with state-of-the-art performance.
- Handles overlapping speech well.
- Offers multiscale embedding for improved accuracy.
- Part of a larger NLP toolkit with additional processing capabilities.

- **Weaknesses:**
  - Requires more setup and computational resources.
  - Less research-oriented compared to Pyannote.audio.
  - May not be ideal for low-quality audio.

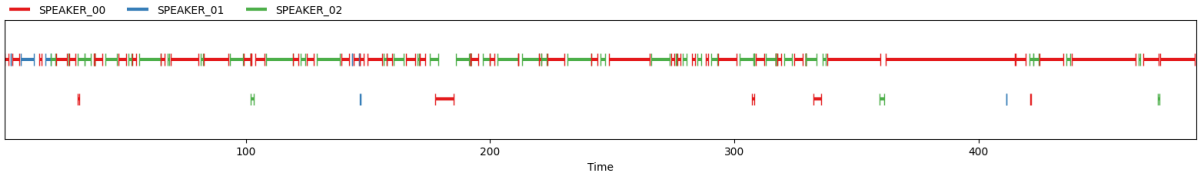
Results (DER)

We use the Diarization Error Rate (DER) as the metric to evaluate the models. See `der.ipynb` for more details.

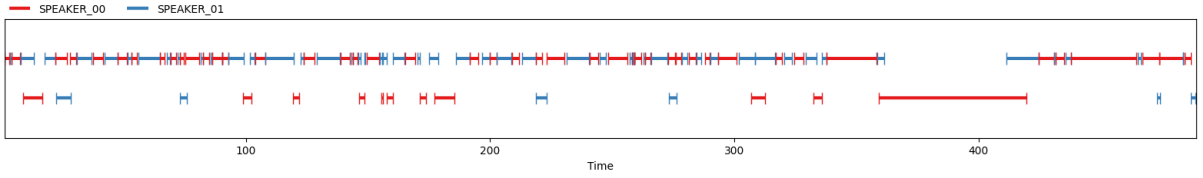
We can take sample `day5_consultation07` for comparing the annotations.



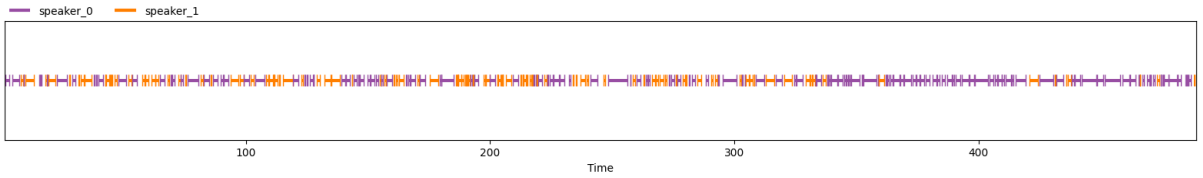
- Pretrained `pyannote` `speaker-diarization-3.1` and `segmentation-3.0` are used to get the annotations. The pre-trained pyannote pipeline reaches a Diarization Error Rate (DER) of 23.7% on the test set.



- The above is fine-tuned on the given dataset. The estimated `best_segmentation_threshold` was 0.595, and `best_clustering_threshold` was 0.799. The model checkpoint can be found at `lightning_logs/version_0/checkpoints/epoch=19.ckpt`. The fine-tuned pyannote pipeline reaches a Diarization Error Rate (DER) of 16.6% on the test set.



- Pre-trained NeMo Speaker Diarization model is used. It uses `marblenet` for VAD, `titanet_large` for speaker embeddings and `Multi-scale Diarization Decoder (MSDD)` as the neural diarizer. The pre-trained NeMO pipeline reaches a Diarization Error Rate (DER) of 31.9% on the test set.



We find that in the scenario of pre-trained zero shot models, pyannote works much better than NeMo. This could be because pyannote is known for performing well on noisy or degraded audio, while NeMo might require higher quality input for optimal accuracy. There could be a lot of other reasons such as model complexity, or better clustering techniques of pyannote.

We can also see that after adapting the pre-trained pyannote model on the custom domain-specific dataset, the DER goes down substantially. This shows us that by fine-tuning the model, we can get much better results for a given dataset.