report

September 5, 2021

1 VAD

author: steeve LAQUITAINE

date: 04/08/2021

1.1 Abstract

Voice activity detection is critical to reduce the computational cost of continuously monitoring large volume of speech data necessary to swiftly detect command utterances such as wakewords. My objective was to code a Voice Activity Detector (VAD) with reasonable performances (Low false rejection rate) based on a neural network within a week and with low computing resources. I trained and tested the model on labelled audio data containing speech from diverse speakers including male, female, synthetic, low and low volume, slow and fast space speech properties. The dataset came from LibriSpeech and was prepared and provided by SONOS. I used a variety of tools to extract, preprocess and develop and test the model but I mostly relied on Tensorflow advanced Subclassing api, tensorflow ops and Keras. I also relied on Tensorboard, Seaborn and the more classical matplotlib visualization tools to make sense of the data, clean the data and inspect the inner workings of the model. To track my training, inference and evaluation pipelines I draw them as directed acyclic graphs using Kedro-Viz api. To track code versioning and feature addition worklow I used git-graph. I have also implemented several common VAD modifications such as label smoothing, minimum speech time and hangover scheme constrains. I used software engineering practices, as much as possible within this constrained deadline to ensure a reproducible, readable, portable and quickly deployable codebase. Future work could augment the dataset with a variety of speech and noise datasets (e.g., from http://openslr.org) and use serialisation compression technique that reduce model memory size and the computational cost of inference such as model quantization, and/or conversion to tensorflow lite format for deployment on mobile devices.

1.2 Method

1.2.1 Performance metrics

I selected `False rejection rate` (FRR) as the main measure of model performance but also report other traditional metrics such as accuracy, precision and recall. `FRR` also called False negative rate or miss rate equates to 1 - Recall and is given by:

$$FRR = \frac{FN}{FN + TP}$$

where "FN" is the count of false negatives, i.e., the incorrect rejections of speech and `TP` stands for `True positives` and is the count of correct speech detections.

I reasoned that:

- the most important objective is not to miss speech events in order to maximize the experience of a speaker user:
- reporting FRR will enable comparison with performance reported in other studies. FRR is widely reported in the literature to assess voice activity detection model performances.

The disadvantage of greedily maximizing that metrics is that it could train a model that detects everything as speech, generating lots of false positive, a poor user experience. It would also be computationally expensive constantly activating downstream tasks such as speech recognition.

1.2.2 Dataset

The dataset contains 1914 files separated in 957 .wav audio files associated with a label file (n=957) with the same name. Label files contained key-value pairs delimiting each speech interval formatted as start_time: XX secs and end_time: YY secs in secs.

1.2.3 Model implementation

```
/Users/steeve_laquitaine/Desktop/vad/src/vad/pipelines/data_eng/nodes.py:20:
YAMLLoadWarning: calling yaml.load() without Loader=… is deprecated, as the
default Loader is unsafe. Please read https://msg.pyyaml.org/load for full
details.
  config = yaml.load(conf)
```

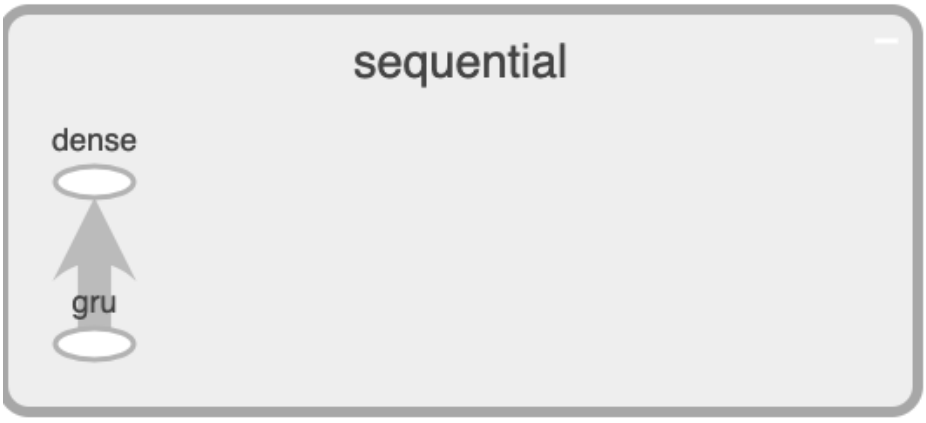
Neural network architecture: I formalized the task as a sequence classification task, where the choosen model must classify a current input audio data point as "speech" or "not speech" based on N preceding audio datapoints. Recurrent neural networks do exactly that. They use a recurrent architecture that retain past data in "memory" long enough to learn the weights that integrate them with current data in order to best predict current data.

Because i'm developping on a laptop, I chose to test a minimal network architecture of one GRU layer and a dense binary classification layer. I generated the neural network's conceptual graph shown below with tensorboard graph api.

After cloning the project's repository, the graph can be visualized in your web browser by running the command below in a linux (or MacOS unix) terminal:

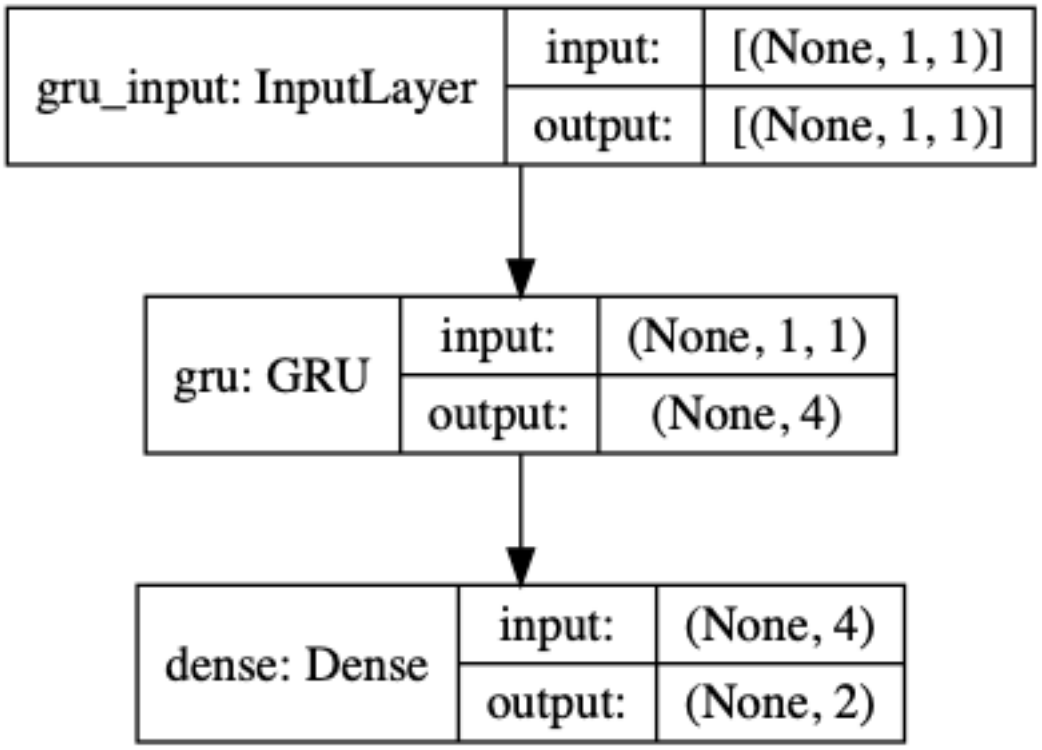
```
cd ../vad
tensorboard --logdir tbruns
```

where ../vad is the clone repository directory.



I used `tf.keras.utils.plot_model()` api to obtain a more detailed description of each layer of the network and its number of units (e.g., I used 4 units for the GRU layer and 2 units for the classification layer.

Audio time series were reshaped to fit with the input format required by the GRU layer (number of samples in 1 batch, number of look back timesteps, 1 feature).



The model had 94 parameters which were all trainable (see below plotted with `model.summary()`).

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
gru (GRU)                    (1, 4)                    84
_________________________________________________________________
dense (Dense)                (1, 2)                    10
=================================================================
Total params: 94
Trainable params: 94
Non-trainable params: 0
_________________________________________________________________
```

1.3 Experiments

1.3.1 Approach

I chose a simple model as its simplicity permits to quickly iterate during development add new features and to debug easily. I tested several hypothesis to converge to a pipeline:

Does a minimal model perform better than chance ?

I first extracted a small sample of an audio data file (2.8 secs of 19-198-0003.wav) for quick training at the possible expense of performance. I trained a simple 1-layer GRU model which takes as input a single batch with one look-back timestep and projects to a fully connected binary classification layer with a softmax activation function. I first trained the model with one epoch. I used `Adam`, an adaptive optimizer that is well known to handle the complex training dynamics of recurrent networks better than simple gradient descent optimization. Training the model on a 2.8 secs audio from one speaker (20% of the full audio chunk) took 50 secs.

Lower resolution, better speed?:

I tested two data encoding resolutions: float16 and float32 bits. Both resolution took the same time to train.

Does learning from more timesteps reduce performance ?:

I tested 1, 2, 10 and 320 timesteps which correspond to 0.06 ms, 0.12 ms, 1.2 ms and 20 ms with the current sampling rate of 16 Khz. - Moving from 1 (50 sec training) to 2 timesteps (2 min training) produced a 2.5 fold improvement in FRR from 1 timestep (0.34 on average to 0.14) over subjects - FRR improved by 20% from 2 to 10 timesteps which took 5 min to train. - Training on 1 timestep took about 50 secs, and 10 timesteps took 90 secs. - 320 timesteps took was too long to train to enable quick iteration within the time constraints and was stopped.

Performance vs. train-test split ?:

I splitted a 2.8 sec audio (20% of a 14 sec audio file) in a 50% training set / 50% test set or 70% / 30% scheme. Increasing the size of the training set from 50% to 70% drastically improved performance on the test set from 100% FRR to about 30%.

How long to convergence ?:

The evolution of loss on training nearly converged, flattened, at about 7 epochs which indicates that the model converges quickly and tend to slowly increase over time the validation set, suggesting slight overfitting.

The more data the better ?:

I trained the model on 2.8 secs and 14 secs of the same training audio file, for 7 epochs:

The 14 secs case had: * 15% better FRR recall and recall * 10% lower precision, indicating an increase in incorrect speech detections

Does training on 2.8 secs audio perform well enough ?:

I trained the model on the 2.8 secs chunk and tested whether it generalizes its performance to the same audio file's remaining audio chunk and to other speakers' audios:

- I trained on speaker (19-198) and predicted on a sample of other test audio chunks from the same speaker. The test performance were about 0.85 f1-score.

I tested how well the model performance generalizes to other speakers audio:

- I trained on speaker (19-198) and predicted on two other persons' audio chunks (103-1240-0001, 118-47824-0000). As a result:
  - 103-1240-0001:
    - I observed an increase in false rejection rate (50% only compared to ~30%)

        * candidate explanation: other speakers could speak with lower average amplitude, or shorter speeches which could produce detection misses.
    - 118-47824-0000:
        * produced the same False rejection rate as for the training speaker

- Synthetic vs. person
    - train on person 1 - predict on synthetic (1263-141777-0000)
        * result: FRR was good

- Loud vs. low voice
    - train on person 1 - predict on human (1447-130551-0019)
        * kept good f1 score 0.7 with good recall and reasonable precision (0.6)

- normal vs. high pace
    - I train on speaker 1 - predict on human (1578-6379-0017)
    - kept good f1 score with good recall and reasonable precision

Which activation sigmoid (default) vs. biased Softmax ?

False rejection rate was slightly lower (from 13% to 9% on other subjects test audios) for a a softmax biased at the Dense classification layer to compensate class imbalance than for the default sigmoid activation function. So I used biased softmax activation. The speed of convergence remained unchanged.

Is Inference fast enough ?

Inference took 12.6 sec on a 15 sec audio that is 840 ms / sec.

1.3.2 A few sanity checks

I tested the inference pipeline:

1. Audio label mapping was shuffled
    - result: FRR worsened to 40% (possibly not significantly different from chance).
2. I tested a dummy model that always predicts "speech":
    - result: FRR worsened to 40% (precision was high which is expected if labels are imbalanced toward speech)
3. I tested a dummy model that always predicts "no speech":
    - result: FRR worsened to 100%, together with precision.

I tested the train pipeline by shuffling the audio label-mapping:

- As expected:
    - FRR was 100% and precision 0%
    - there was virtually no reduction in loss over epochs indicating no learning

1.3.3 Final setup

Training dataset:

- 2.8 secs audio from one speaker: this small chunk took 4 min to train while using a full 15 secs audio file (i.e., 5 times more data) took 20 min for a 10% performance improvement.

All remaining training in the report are done with this first 2.8 sec audio chunk from 19-198 speaker.

Basic model's architecture:

- 1-layer GRU with 4 units
- Dense classification layer with 2 units, a Softmax activation with an initial bias that counterbalances the label imbalance

Training:

- 1 batch
- 10 timesteps

- at least 7 epochs
- split_frac: 0.7 : had the biggest effect on performance (0 at 0.5 to 0.7 at 0.7)

1.4 Results

1.4.1 Description of the audio data

There are several speakers Listening to a sample of the audio files revealed that a variety of speakers

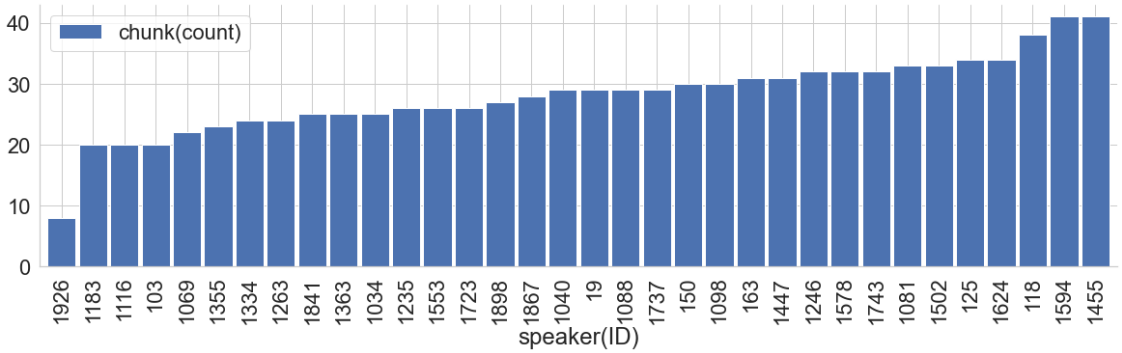
- Humans
  - men
  - women
- Synthetic
  - men
  - women

I also characterised speeches by their variety of amplitudes and pace - Normal vs fast pace
- Loud vs, low volume

We show below the best typical example of an audio signal (top panel). and its associated speech labels "1" for speech and "0" for no speech (bottom panel).

All audio signals were 32 bits float single channel time series. We run a few sanity checks:

```
- the 957 label files were correctly mapped with the 957 audio files

- Number: 34 speakers
- Speakers'ID: ['103' '1034' '1040' '1069' '1081' '1088' '1098' '1116' '118'
'1183'
 '1235' '1246' '125' '1263' '1334' '1355' '1363' '1447' '1455' '150'
 '1502' '1553' '1578' '1594' '1624' '163' '1723' '1737' '1743' '1841'
 '1867' '1898' '19' '1926']
```

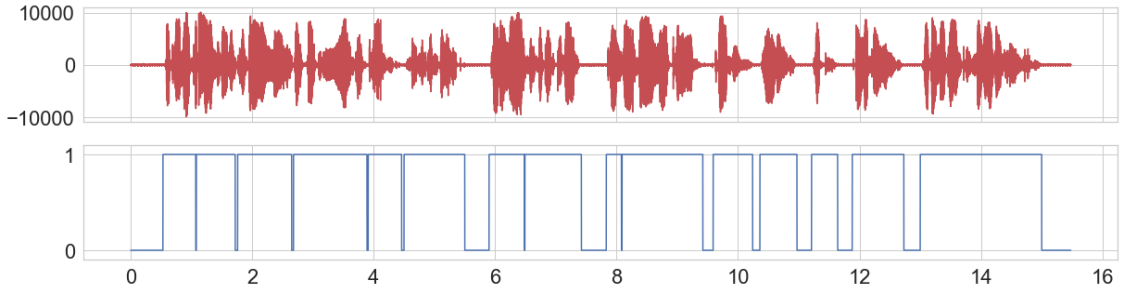


We show below a few interesting example chunks for two speakers. - Audio seem well labelled (see supplementary). - Background noise is low and stationary.

SPEAKER 19

data/01_raw/vad_data/19-198-0003.wav

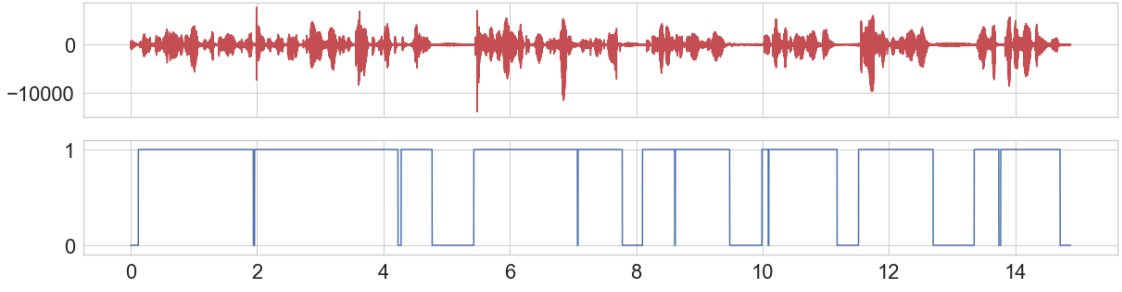
data/01_raw/vad_data/19-198-0003.json



SPEAKER 103

data/01_raw/vad_data/103-1241-0027.wav

data/01_raw/vad_data/103-1241-0027.json



We validated that all audio files were associated with a .json label file.

```
- audio file sample size: 957
- label file sample size: 957
```

The entire sample could be loaded quickly:

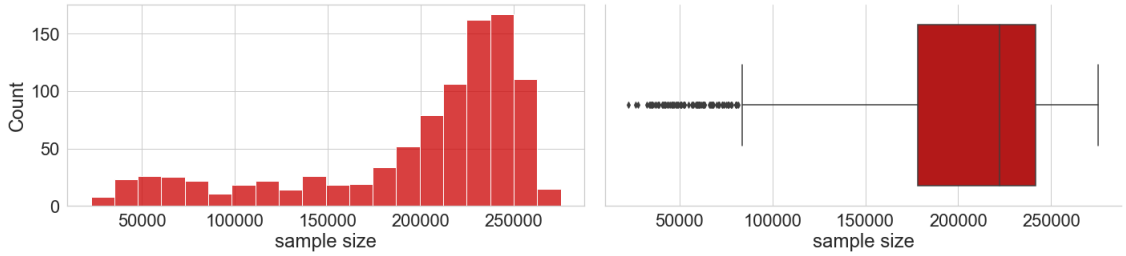
```
- loading duration: 2.75 sec
```

Sample size and sampling rate:

```
Sample rate information:
- 1 sample rate(s)
- rate: 16000 Hz
```

We kept the signal at 16Khz which is enough to cover the frequency range of human speech according to the literature (Human voice b/w `85hz to 8khz` [REF], hearing b/w `20 hz to 20kh`[REF]).

```
Sample size information:
- 711 sample size(s)
- max: 275280 samples ( [17.205] secs)
- min: 22560 samples ( [1.41] secs)
- median: 222080.0 samples ( [13.88] secs)
```



Signal amplitudes: the true decibel amplitude of the audio will depend on each speaker's microphone characteristics, the speaker's distance to its microphone, the speaker's volume configuration. Having no acces to these information we did not derive the true decibel amplitude (dB) from the raw audio signal amplitude or compared absolute amplitudes between speakers. Rather we compared the signals' signal-to-noise ratio (SNR).

1.4.2 Speech signals are nearly pure

I made the naive assumption that audio can be linearly decomposed as the sum of independent speech and noise amplitude components. This assumption would not hold in case of reverberation.

$$audio = speech + noise$$

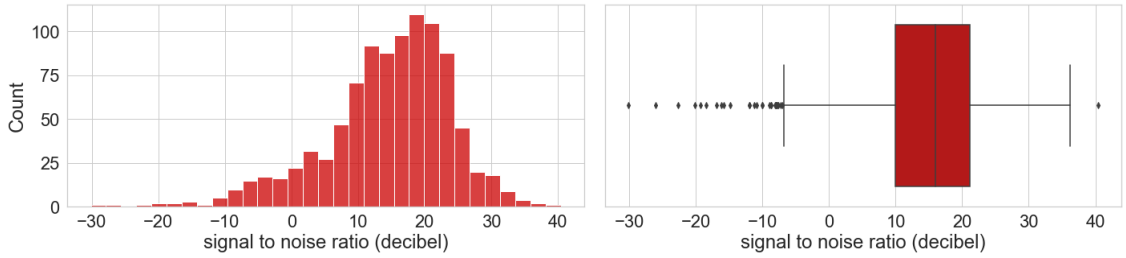
Where i respectively categorize noise as the portion of the audio that is not labelled as speech. Thus:

$$speech = audio - noise$$

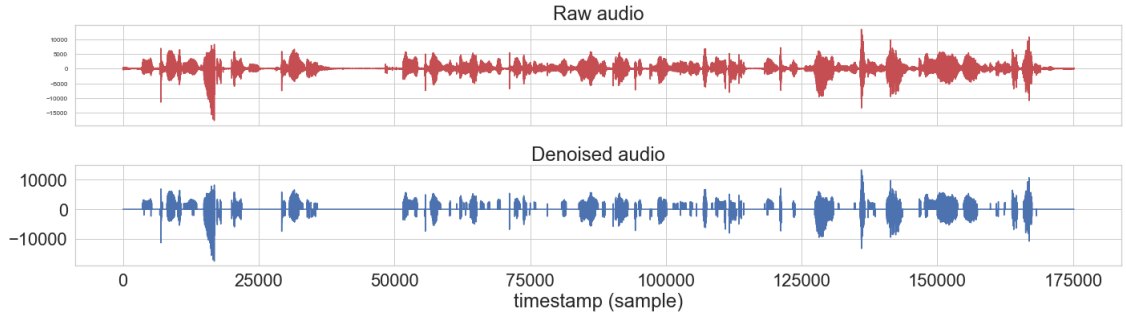
The signal to noise ratio in decibel is given by:

$$SNR = 20.log_{10}(\frac{speech\_rms}{noise\_rms})$$

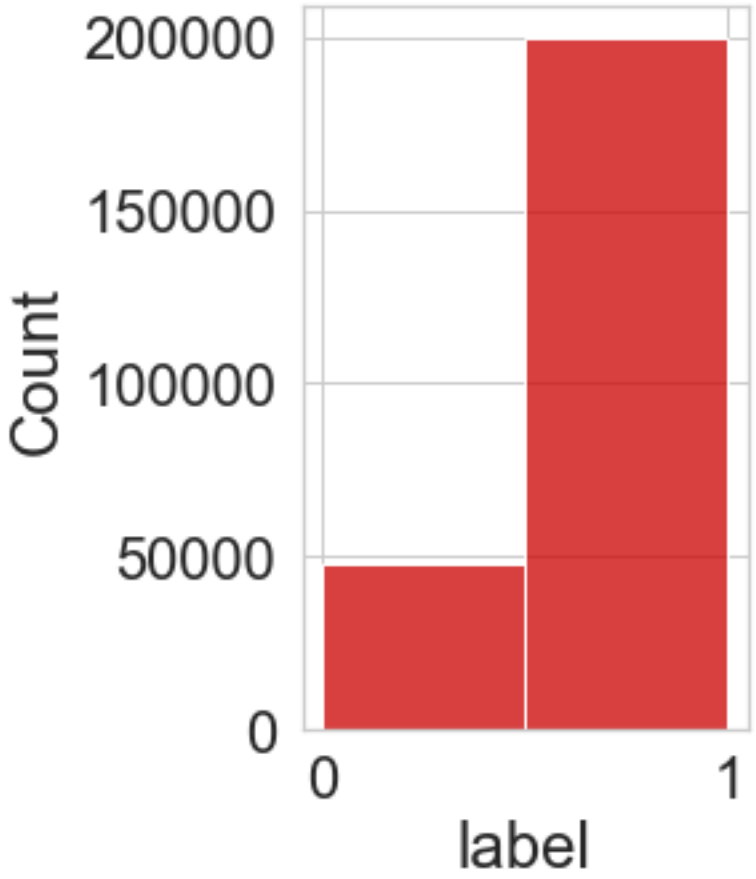
where speech_rms and noise_rms are the root mean square of the speech_signal and of the noise



I show below what noise dampening (blue in bottom panel) does to the raw signal below (red, in top panel) for an example audio file.



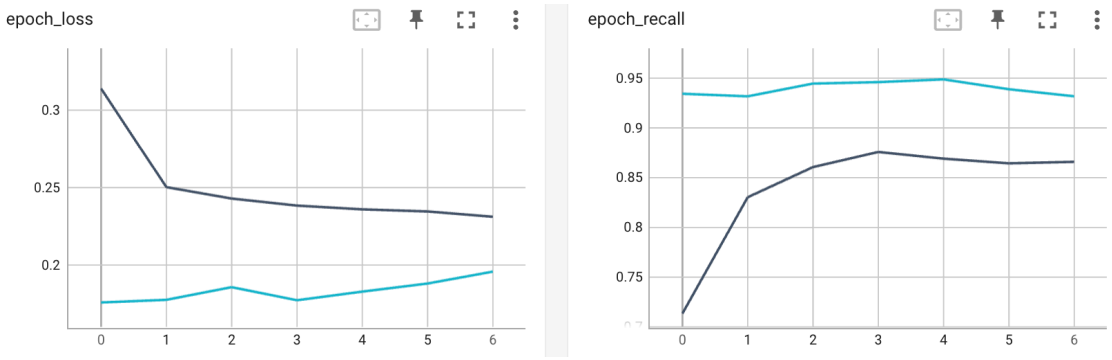
1.4.3 Speech and no-speech classes are imbalanced in the training dataset



1.4.4 The 1-layer GRU net converges within 7 epochs

I monitored training quality in Tensorboard api. I display below the loss curve calculated for 7 training epochs (x-axes ranging from 0 to 6) on training dataset (dark curve) and validation dataset (blue curve). The model was trained with 10 timesteps look-back (see logged runs in tensorboard for vad-1-gru-20210901-131436/train/train, vad-1-gru-20210901-131436/train/validation in tensorboard web api). Categorical cross-entropy on training data was not flat as expected from a model that doesn't learn but goes down until convergence near the 7th epoch.

I also monitored recall to evaluate improvement in recall (decrease in FRR) over training. Recall increased (thus FRR decreased) by 10% within 7 epochs and took about 4 epochs to converge on the training dataset. Recall on validation was already high since the first epoch and remained stable over training (see logged runs in tensorboard for vad-1-gru-20210901-131436/train/train, vad-1-gru-20210901-131436/train/validation).



1.4.5 Weights and biases follow well-behaved multimodal distributions

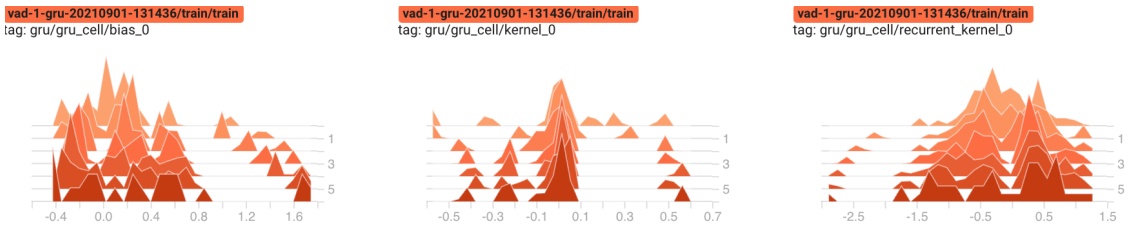
Both the Gru and dense layer show distribution that did not get stuck at 0 values and did not display large outlier values. This suggests that our GRU units successfully circumvented the vanishing gradient problem that is typical of more traditional RNN, thanks to its "Gates" which avoid the long-term information from "vanishing" away.

Gru layer

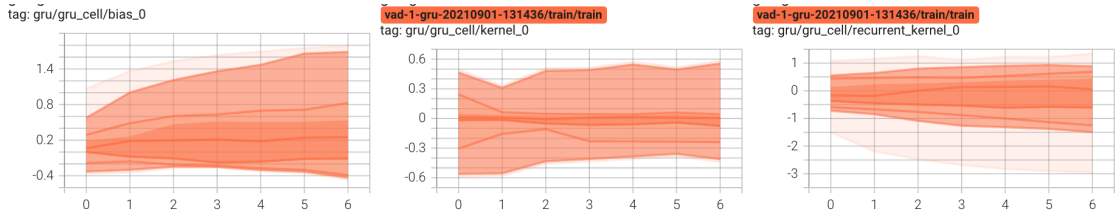
The GRU l layer has 3 types of trainable parameters: - weights ("kernel_0") - recurrent weights ("kernel_0") - biases ("kernel_0")

I will refer to iterations as epochs as for a single batch, iterations equal the number of epochs.

- Weight and bias values
  - The weight distribution was very sparse with the majority of weights being null and some values being spread at negative and positive values.
  - GRU layer's biases and recurrent weights slightly changed over epochs (y-axis), spreading across a larger range of values.

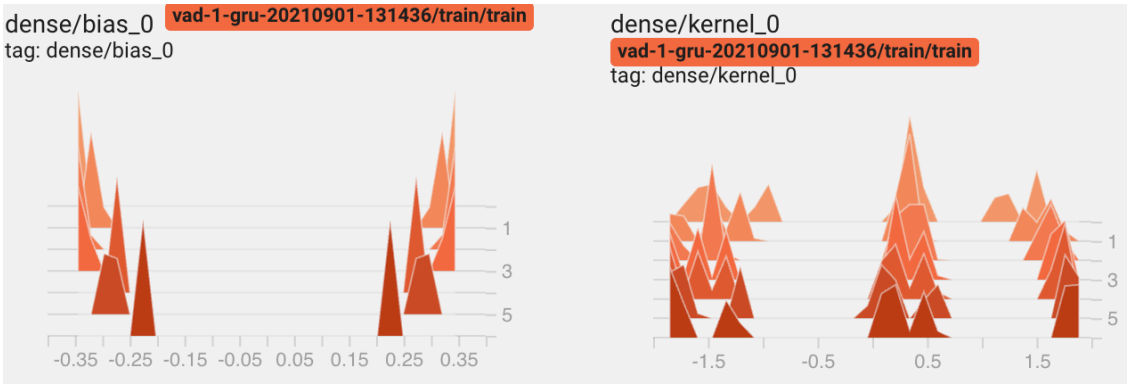


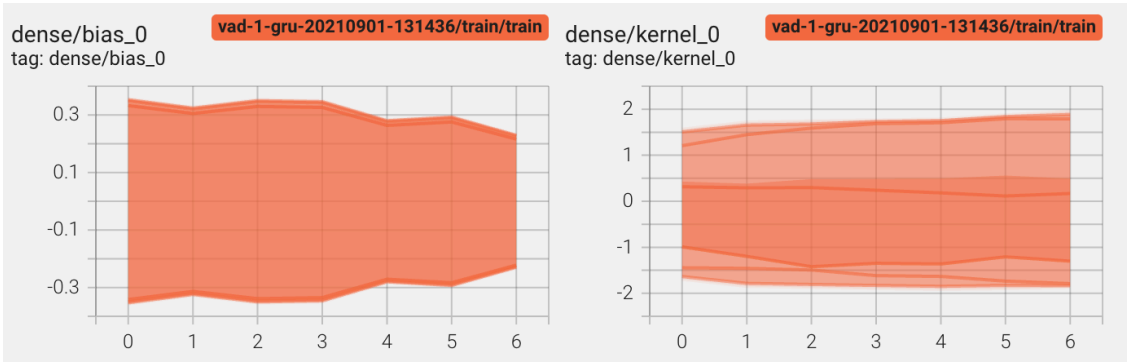
These changes in the bias and weights distributions are more apparent on the plot of the weight values against epochs (different shading columns represent the distributions' 90th, 60th percentiles ...) (x-axis).



Dense layer

- Dense layer's weights and biases did not get not stuck at 0 and did not display large outlier values.
  - Biases followed a bi-modal distributions with some weights at -1, others peaking at -0.3 and other at +0.3. There are no outlier biases.
  - Weights followed a tri-modal distributions with some weights at -1, a small majority peaking at 0 and some peaking at +1.5. There were no outlier bias values.





1.4.6 A 1-layer GRU model predicts well

I first trained the model on a small chunk (2.8 secs) of one speaker data and I tested it on one full audio for each of the 34 speakers' (about 3.5 min, for 15 sec inference per audio file). I chose the

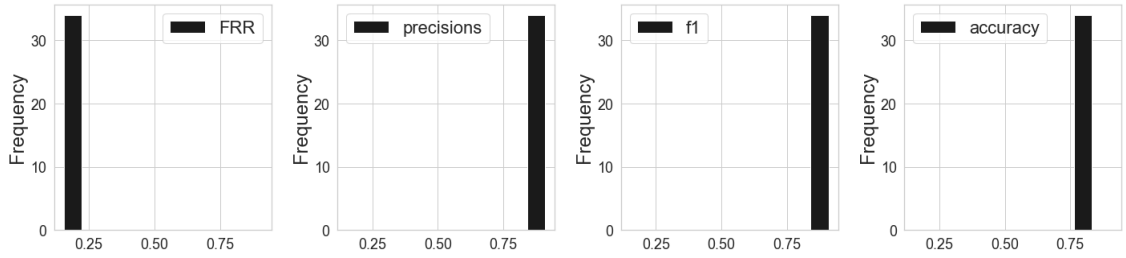
first audio for simplicity. Further analyses must assess whether the first audio is representative of the other audios for each speaker. I chose a sample and not the entire dataset because inference would take more than 3 hours on the 957 files. I show below the distribution of the best model's False rejection rates over the sample test dataset.

I display below the sequential steps performed by my inference pipeline and the False rejection rates for all speaker's first audio recording.

I show below the FRR statistics calculated over the FRR calculated from the first audio of the test subjects with various other usual performance metrics:

	FRR	precision	f1	accuracy
count	3.400000e+01	3.400000e+01	3.400000e+01	3.400000e+01
mean	1.508353e-01	9.119925e-01	8.794579e-01	7.952717e-01
std	2.817298e-17	1.126919e-16	1.126919e-16	1.126919e-16
min	1.508353e-01	9.119925e-01	8.794579e-01	7.952717e-01
25%	1.508353e-01	9.119925e-01	8.794579e-01	7.952717e-01
50%	1.508353e-01	9.119925e-01	8.794579e-01	7.952717e-01
75%	1.508353e-01	9.119925e-01	8.794579e-01	7.952717e-01
max	1.508353e-01	9.119925e-01	8.794579e-01	7.952717e-01

The FRR histogram (shown below) concentrates near 9% with no apparent outlier. This indicates that the model generalize well to all speakers: human men, women and synthetic speakers.



1.4.7 Adding VAD modification

Enforcing a minimum speech time The minimum speech time observed in the dataset was very short. A realistic minimum speech duration would be 1 secs, to say "hi" for example.

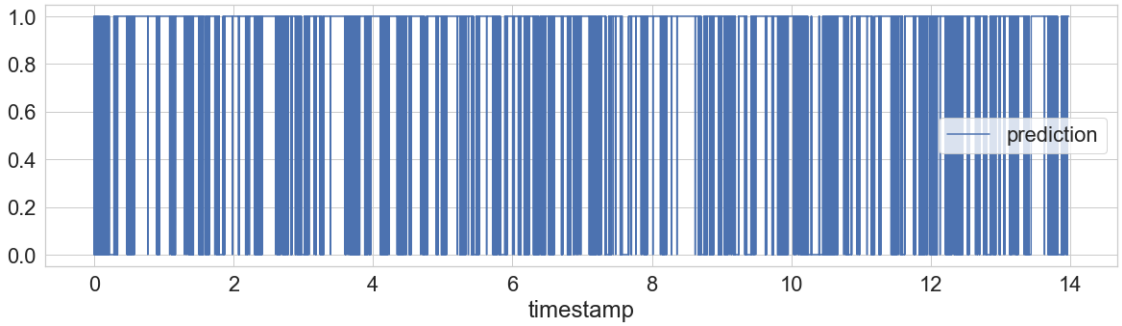
```
The minimum speech time derived from the data was: 10.0 ms
```

Method 1: Filtering out speech events with a duration below minimum speech time To enforce a minimum speech time I modified the training labels before fitting the model. I chose a mimum speech time of 1,000 ms (16,000 points at 16Khz). I detected speech event intervals start and end timestamps and re-labelled as not speech ("0") all speech intervals with durations below 1000 ms (length below 16,000 points).

Filtering out short speech events with duration below 1 sec should favor model weights that produce speech intervals longer than 1 sec. This is a soft constraining method as it does not ensure that 100% of speech events will be above 1 sec but increase their probability of occurrence.

To run the train and predict pipelines run the commands (in a linux/unix terminal):

```
kedro run --pipeline train --env train_min_speech
kedro run --pipeline predict_and_eval --env predict_and_eval_min_speech
```



	0
accuracy	0.795272
precision	0.911992
recall	0.849165
f1	0.879458
false_rejection_rate	0.150835

Method 2: constraining the loss function with my custom loss I modified the model to predict a sequence of labels. I implemented a Gru layer that outputs a (.., T timesteps, ..) fed to replaced a classification layer. I replaced the Dense classification layer used for the former model by a TimeDistributed Dense layer with a softmax activation function to predict a sequence of (T timesteps, 2 classes) label probabilities. I added a penalty to the loss function such that each occurrence of minimum speech time violation with the choosen a "timesteps" period increases the loss by +1.

I display my custom loss function below (see `src.vad.pipelines.train.nodes.MinSpeechLoss`).

```python
class MinSpeechLoss(tf.keras.losses.Loss):
    """Loss penalized to enforce a minimum speech time"""

    def __init__(self):
        """Instantiate loss constrained to enforce a minimum speech time"""
        super().__init__()

    def call(self, y_true, y_pred):
        # replicate y_pred over timestep axis and calculate loss
        cce = CategoricalCrossentropy()
        cce_loss = cce(y_true, y_pred)
        cce_loss = tf.convert_to_tensor(cce_loss)

        # penalize minimum speech violations
        minspeech_loss = get_penalty(y_pred)
```

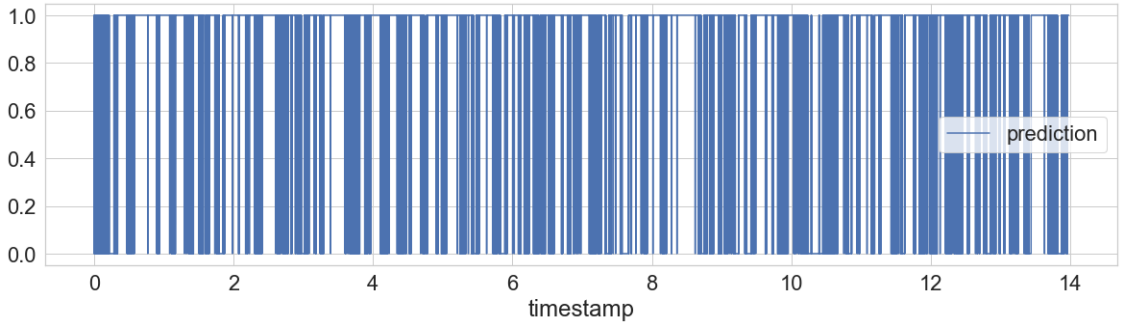
```
        return cce_loss + minspeech_loss

    @classmethod
    def from_config(cls, config):
        return cls(**config)
```

Applying label smoothing I used categorical cross entropy loss and applied label smoothing (0.5). Label values are smoothed, relaxing the confidence on their values. For example, i applied a `label_smoothing=0.5` which corresponds to 0.5 for the non-speech label and 1 for the speech label.

```
model.compile(
    loss=CategoricalCrossentropy(label_smoothing=0.5), ...
)
```

Before smoothing Train the model and predict in the terminal (linux):

```
kedro run --pipeline train --env train
kedro run --pipeline predict --env predict
```

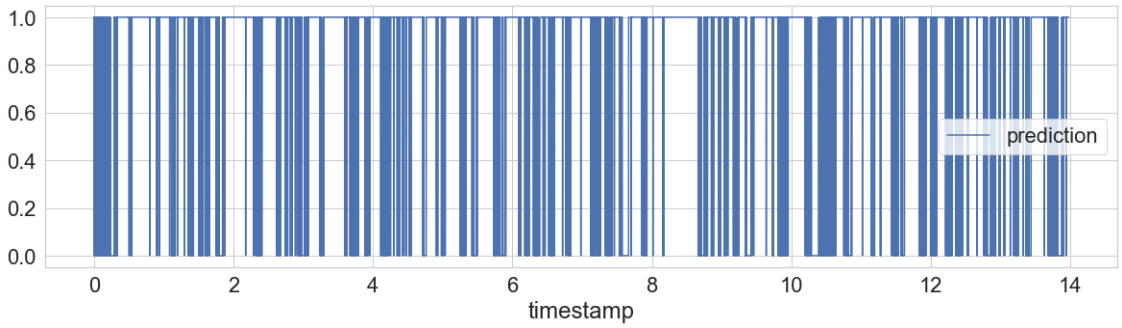


The performance metrics of the model on this example subject are:

	0
accuracy	0.795272
precision	0.911992
recall	0.849165
f1	0.879458
false_rejection_rate	0.150835

After smoothing ("label smoothing") I train the model with label smoothing on 2.8 secs of my training subject's audio and calculated model predictions on test data from the example subject (another subject). To run inference run the commands below in the terminal:

```
kedro run --pipeline train --env train_smooth
kedro run --pipeline predict --env predict_smooth
```



Smoothing the labels increased the performance on test data, producing lower FRR for this example.

	0
accuracy	0.817639
precision	0.909771
recall	0.879921
f1	0.894597
false_rejection_rate	0.120079

Enforcing a hangover of 100 ms To add a hangover I modified the training labels before fitting the model. I chose a hangover of 100 ms (160 points at 16Khz) which was added before and after each speech event. I detected speech event intervals start and end timestamps and moved the start back in time by 160 points and the end forward by 160 points.

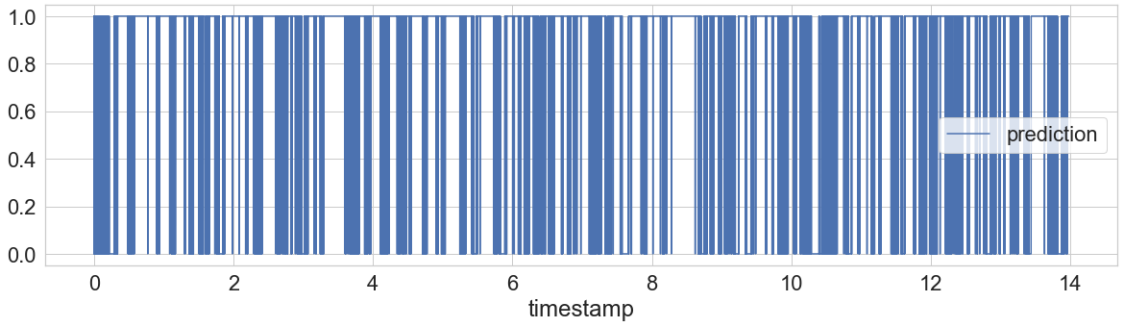
For example for speech interval 1:

- before hangover (raw data): start_time: 160th point - end_time: 200th point
- after hangover: start_time: 0th point - end_time: 360th point

Increasing speech intervals with force model optimization to settle on weights that produce longer speech intervals.

To run the train and predict pipelines run the commands (in a linux/unix terminal):

```
kedro run --pipeline train --env train_hangover
kedro run --pipeline predict_and_eval --env predict_and_eval_hangover
```



	0
accuracy	0.801600
precision	0.911414
recall	0.857789
f1	0.883789
false_rejection_rate	0.142211

1.5 Conclusion & Discussions

I have developped a 1-layer GRU neural network that detects speech activity with an average of 9% of false rejection rat. It was trained on 2.8 secs audio from one speaker and tested on datasets from other speakers. The dataset contained audio with diverse types of voices which included male voices, female voices, synthetic voices, loud and low-volume voices, normal and faster paces. All development was realised on a resource-constrained device (a 2.3Gz RAM 4 cores MacOS laptop).

I have implemented three voice activity detection modifications: - **mimimum speech time** - by filtering out all speech events below a minimum speech time. I label them as not speech. - by creating a custom loss functions including a +1 penalty to cross categorical loss for each mimium speech time violation within windows of 10 ms (working, but not tuned and not optimal). - **label smoothing** to reduce the too frequent and unrealistic alternance betwen speech events and no-speech intervals in predictions. - **hangover** to enforce speech predictions with longer durations

I have used several tools and techniques: - Tensorflow - tensorflow ops including control flow - tensorflow keras - subclassing api for advanced model tuning - Tensorflow board - mlflow - kedro - see the list of dependencies in requirements.txt

1.6 Perspectives

To improve the model, i could apply a few preprocessing steps:

- data minmax rescaling between 0 and 1
- z-scoring

to speed up learning.

To improve the model the next steps would be to try a more thorough hyperparameter search:

- different learning rates
- more model architectures

The model should also be tested for online speech detection with streaming audio data from a microphone: - Add a streaming inference pipeline that makes online predictions

Deployment a constrained resource device such as an iphone or BlackBerry Pi could be tested and would require, for example:

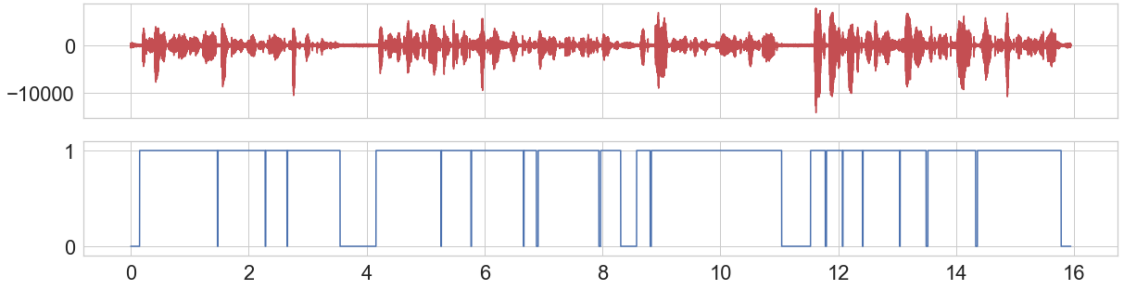
- model quantization
- serializing the model to a lighter file format such as tensorflow lite

1.7 References

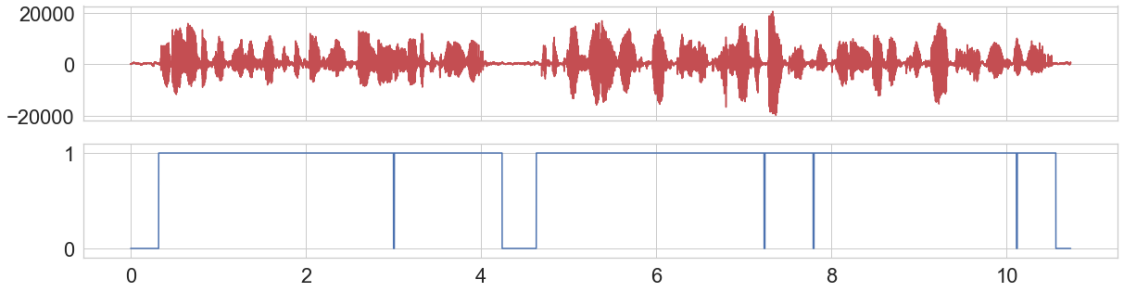
1.8 Supplementary results

1.9 Speakers' first audio

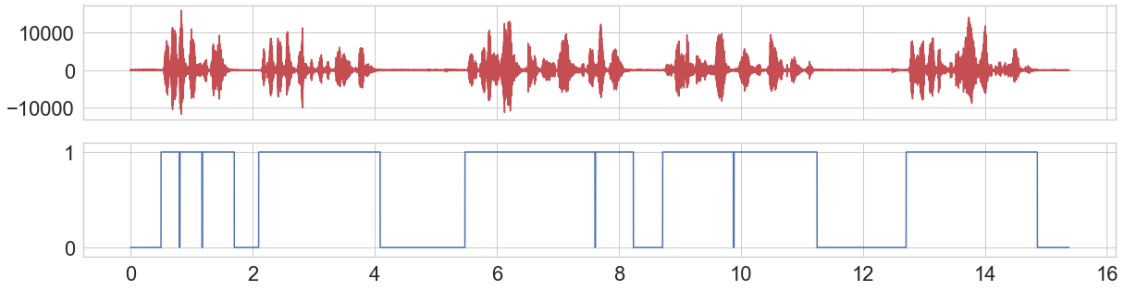
```
SPEAKER 103 - PANEL 0
  data/01_raw/vad_data/103-1240-0001.wav
  data/01_raw/vad_data/103-1240-0001.json
```



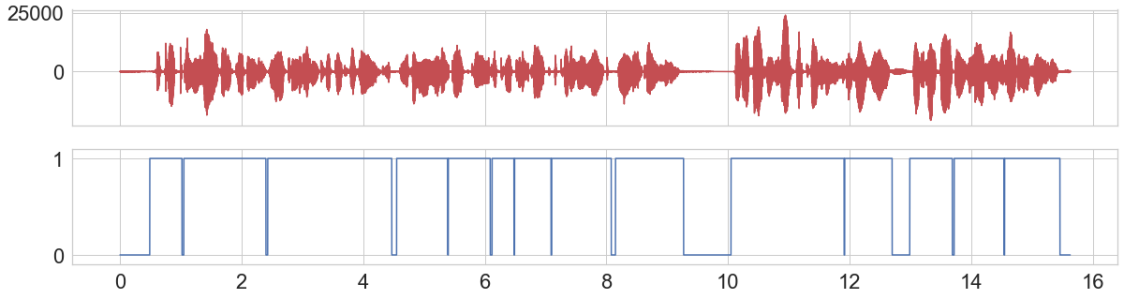

```
SPEAKER 1034 - PANEL 1
  data/01_raw/vad_data/1034-121119-0005.wav
  data/01_raw/vad_data/1034-121119-0005.json
```



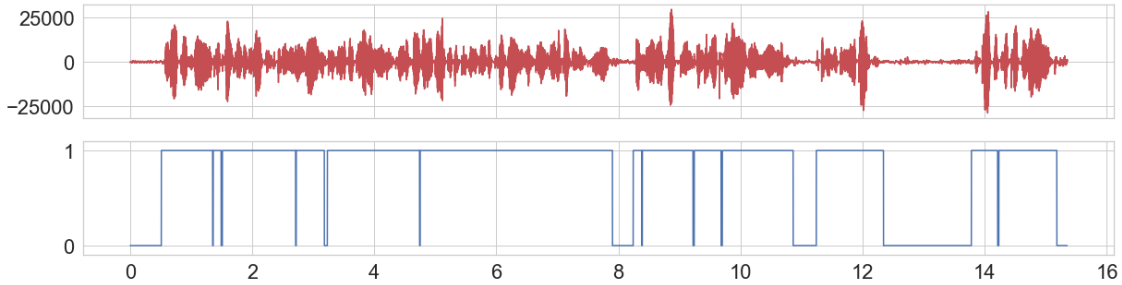

```
SPEAKER 1040 - PANEL 2
  data/01_raw/vad_data/1040-133433-0001.wav
  data/01_raw/vad_data/1040-133433-0001.json
```



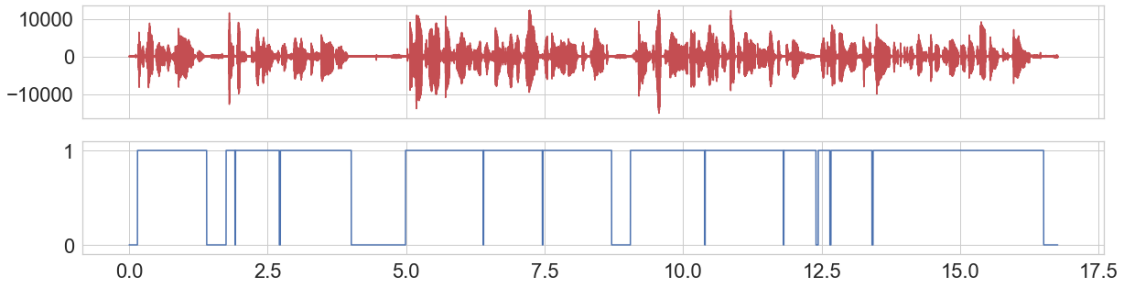
SPEAKER 1069 - PANEL 3
  data/01_raw/vad_data/1069-133699-0000.wav
  data/01_raw/vad_data/1069-133699-0000.json



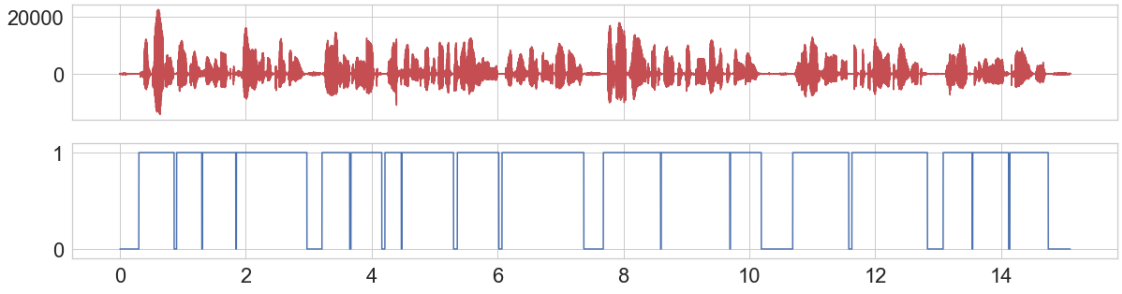
SPEAKER 1081 - PANEL 4
  data/01_raw/vad_data/1081-125237-0007.wav
  data/01_raw/vad_data/1081-125237-0007.json



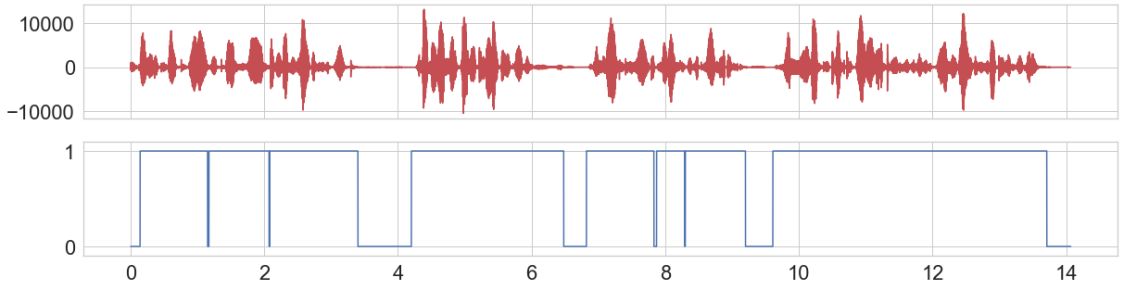
SPEAKER 1088 - PANEL 5
  data/01_raw/vad_data/1088-129236-0003.wav
  data/01_raw/vad_data/1088-129236-0003.json



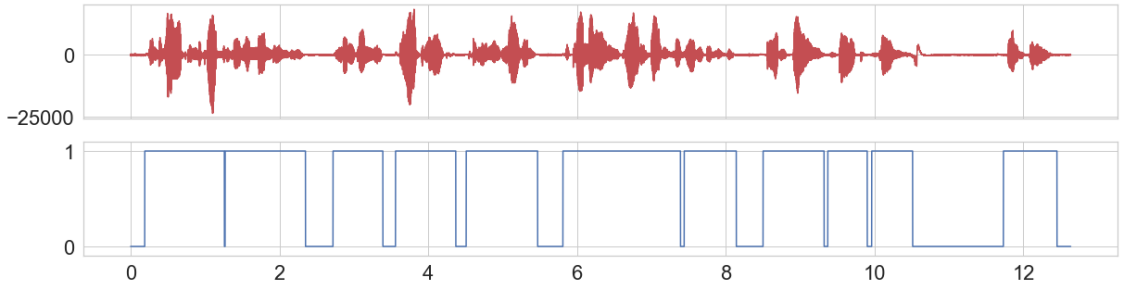
SPEAKER 1098 - PANEL 6
  data/01_raw/vad_data/1098-133695-0001.wav
  data/01_raw/vad_data/1098-133695-0001.json



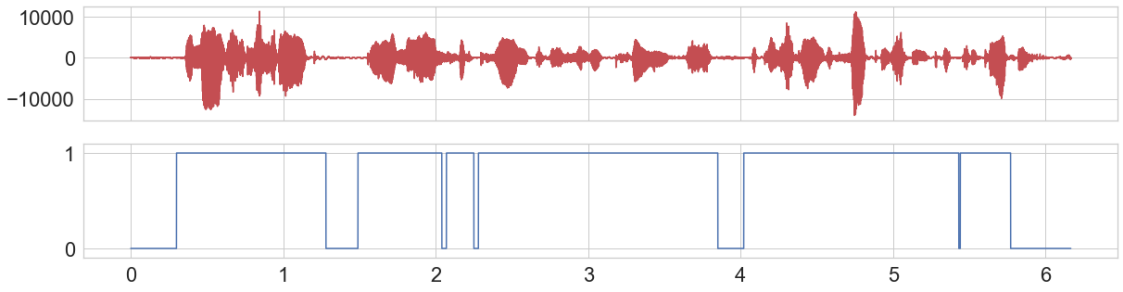
SPEAKER 1116 - PANEL 7
  data/01_raw/vad_data/1116-132847-0003.wav
  data/01_raw/vad_data/1116-132847-0003.json



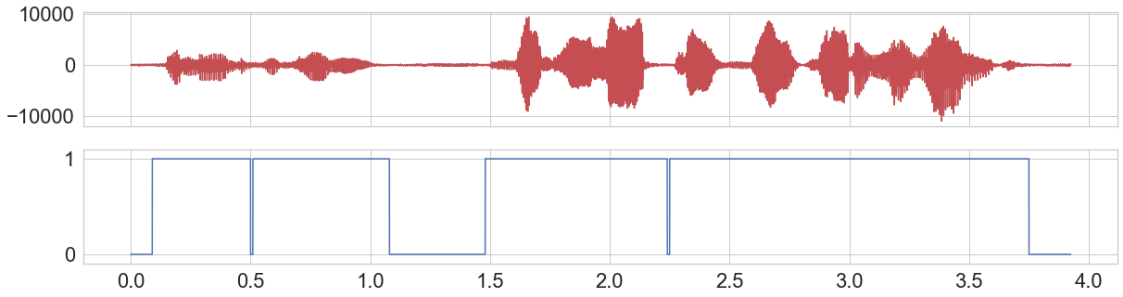
SPEAKER 118 - PANEL 8
  data/01_raw/vad_data/118-121721-0005.wav
  data/01_raw/vad_data/118-121721-0005.json



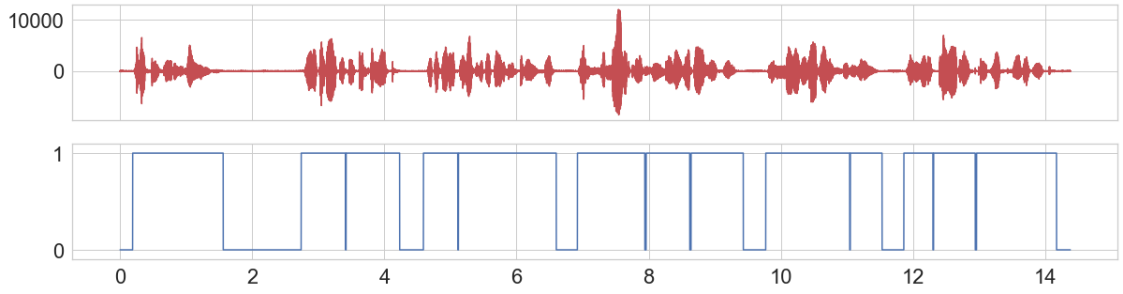
SPEAKER 1183 - PANEL 9
  data/01_raw/vad_data/1183-124566-0000.wav
  data/01_raw/vad_data/1183-124566-0000.json



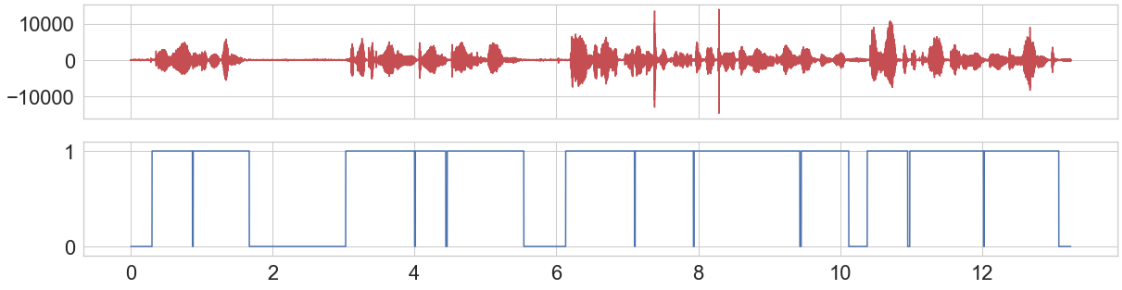
SPEAKER 1235 - PANEL 10
  data/01_raw/vad_data/1235-135883-0007.wav
  data/01_raw/vad_data/1235-135883-0007.json



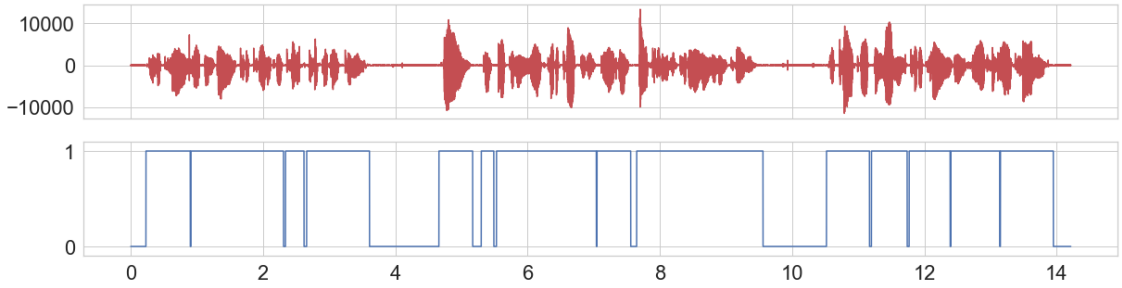
SPEAKER 1246 - PANEL 11
  data/01_raw/vad_data/1246-124548-0000.wav
  data/01_raw/vad_data/1246-124548-0000.json



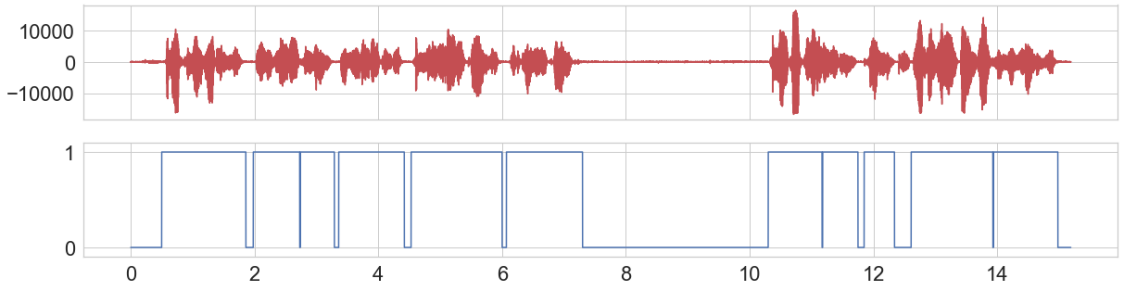
SPEAKER 125 - PANEL 12
  data/01_raw/vad_data/125-121124-0000.wav
  data/01_raw/vad_data/125-121124-0000.json



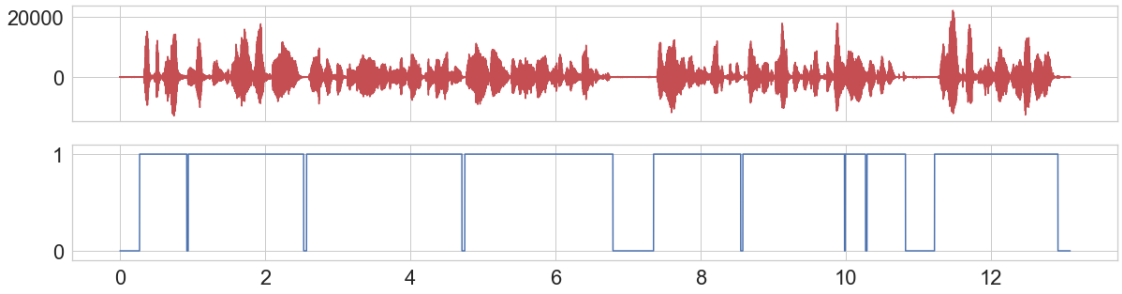
SPEAKER 1263 - PANEL 13
  data/01_raw/vad_data/1263-138246-0000.wav
  data/01_raw/vad_data/1263-138246-0000.json



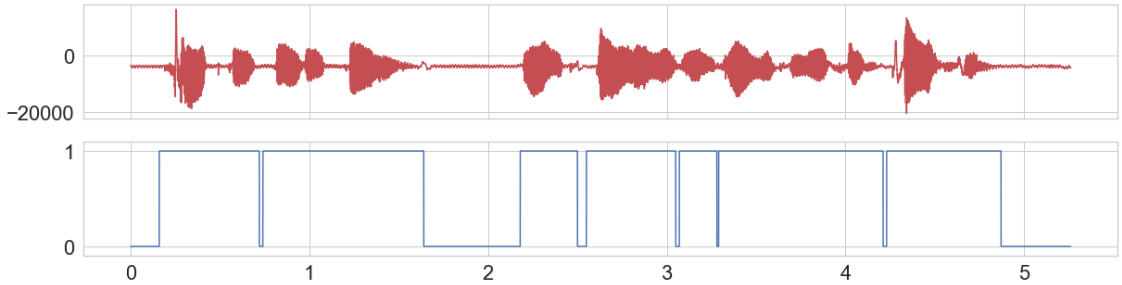
SPEAKER 1334 - PANEL 14
  data/01_raw/vad_data/1334-135589-0011.wav
  data/01_raw/vad_data/1334-135589-0011.json



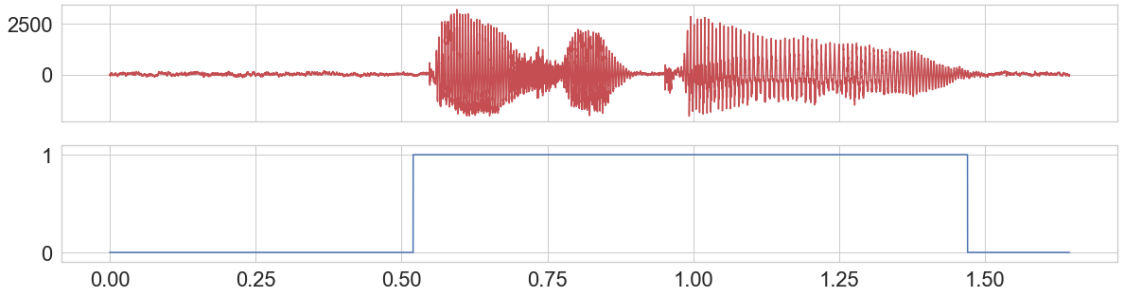
SPEAKER 1355 - PANEL 15
  data/01_raw/vad_data/1355-39947-0014.wav
  data/01_raw/vad_data/1355-39947-0014.json



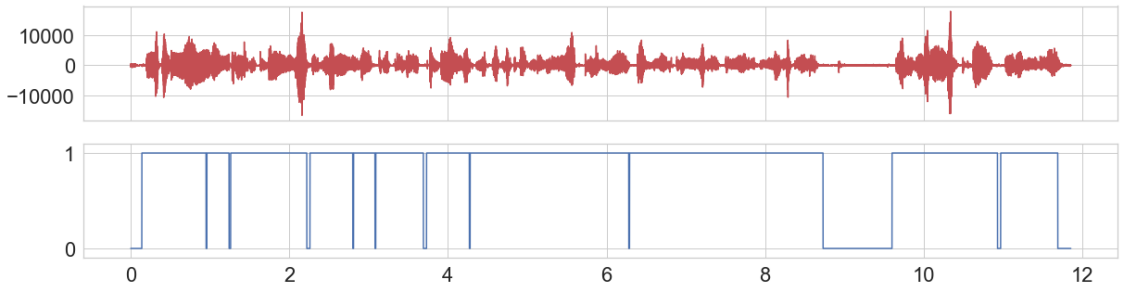
SPEAKER 1363 - PANEL 16
  data/01_raw/vad_data/1363-135842-0000.wav
  data/01_raw/vad_data/1363-135842-0000.json



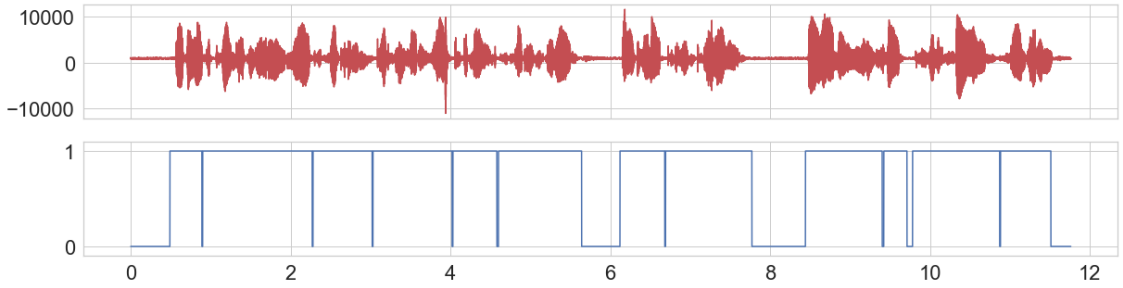
SPEAKER 1447 - PANEL 17
  data/01_raw/vad_data/1447-130550-0000.wav
  data/01_raw/vad_data/1447-130550-0000.json



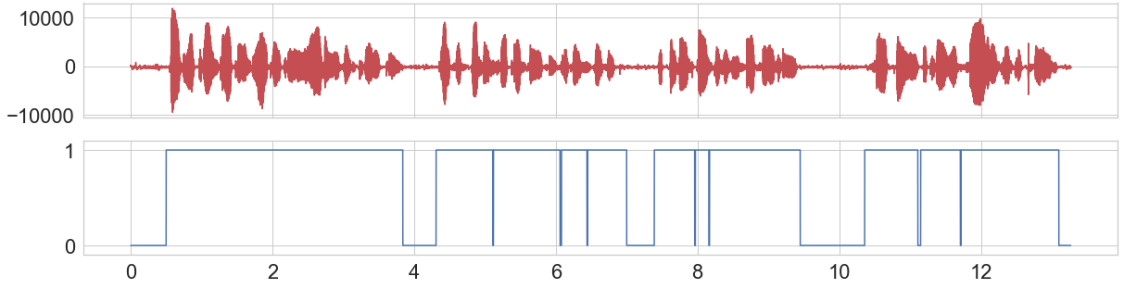
SPEAKER 1455 - PANEL 18
  data/01_raw/vad_data/1455-134435-0007.wav
  data/01_raw/vad_data/1455-134435-0007.json



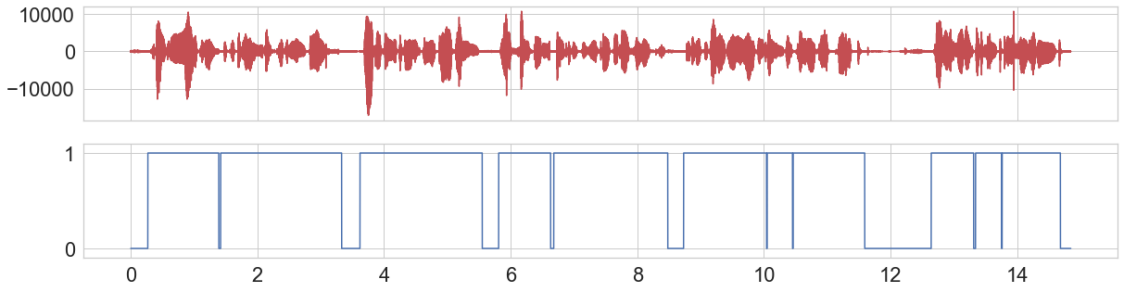
SPEAKER 150 - PANEL 19
  data/01_raw/vad_data/150-126107-0001.wav
  data/01_raw/vad_data/150-126107-0001.json



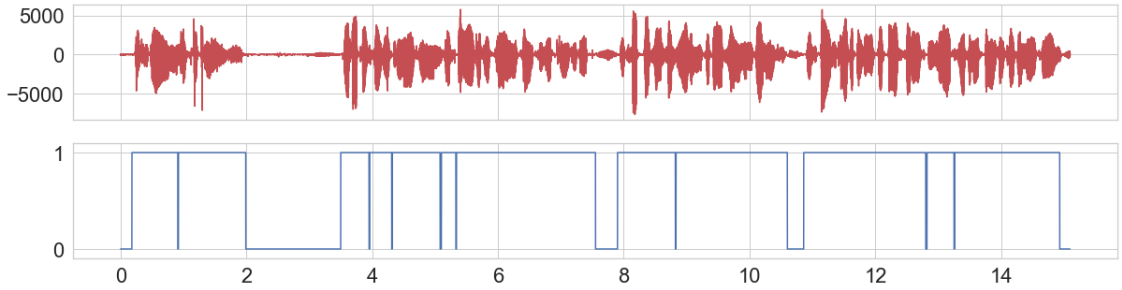
SPEAKER 1502 - PANEL 20
  data/01_raw/vad_data/1502-122615-0007.wav
  data/01_raw/vad_data/1502-122615-0007.json



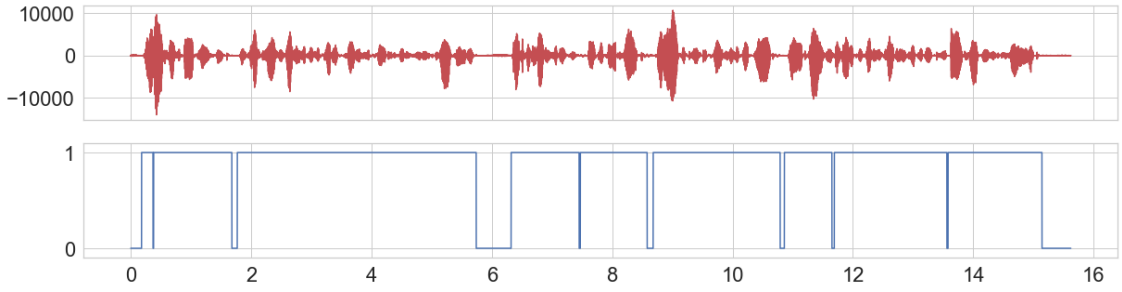
SPEAKER 1553 - PANEL 21
  data/01_raw/vad_data/1553-140047-0002.wav
  data/01_raw/vad_data/1553-140047-0002.json



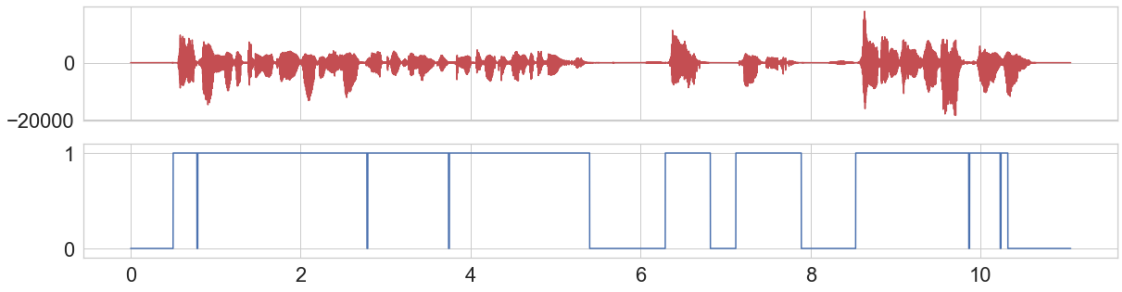
SPEAKER 1578 - PANEL 22
  data/01_raw/vad_data/1578-140045-0000.wav
  data/01_raw/vad_data/1578-140045-0000.json



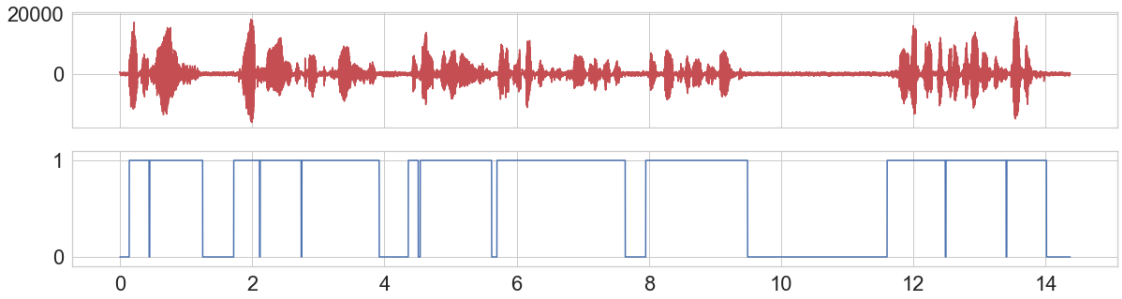
SPEAKER 1594 - PANEL 23
  data/01_raw/vad_data/1594-135914-0004.wav
  data/01_raw/vad_data/1594-135914-0004.json



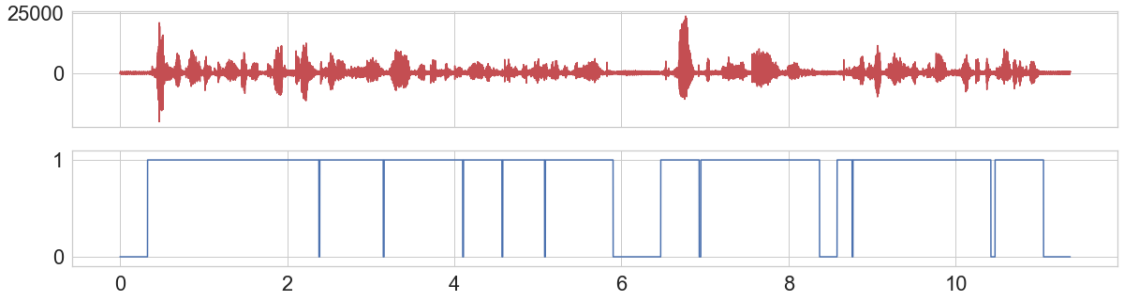
SPEAKER 1624 - PANEL 24
  data/01_raw/vad_data/1624-142933-0003.wav
  data/01_raw/vad_data/1624-142933-0003.json



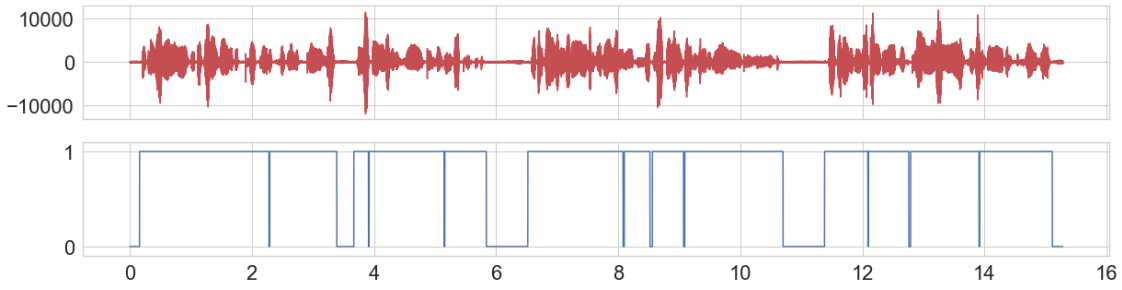
SPEAKER 163 - PANEL 25
  data/01_raw/vad_data/163-121908-0006.wav
  data/01_raw/vad_data/163-121908-0006.json



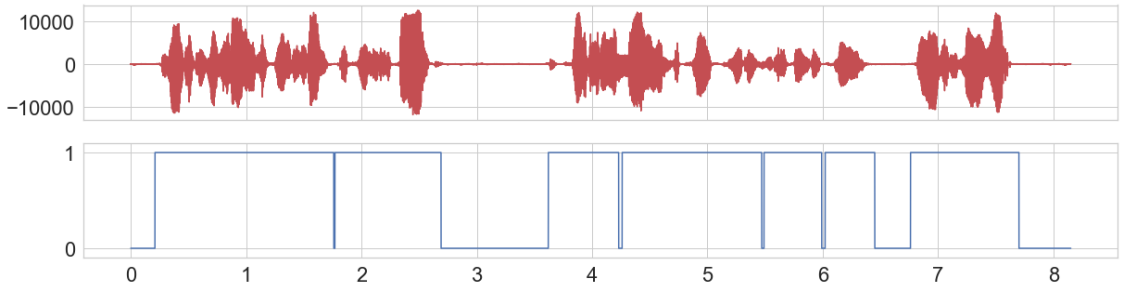
SPEAKER 1723 - PANEL 26
  data/01_raw/vad_data/1723-141149-0005.wav
  data/01_raw/vad_data/1723-141149-0005.json



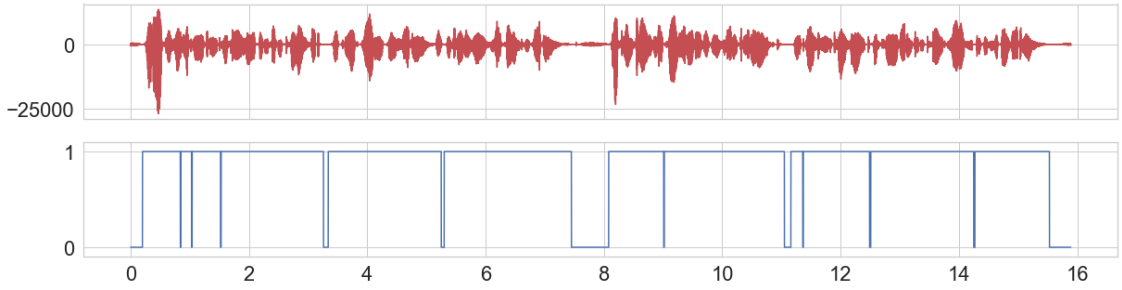
SPEAKER 1737 - PANEL 27
  data/01_raw/vad_data/1737-142396-0000.wav
  data/01_raw/vad_data/1737-142396-0000.json



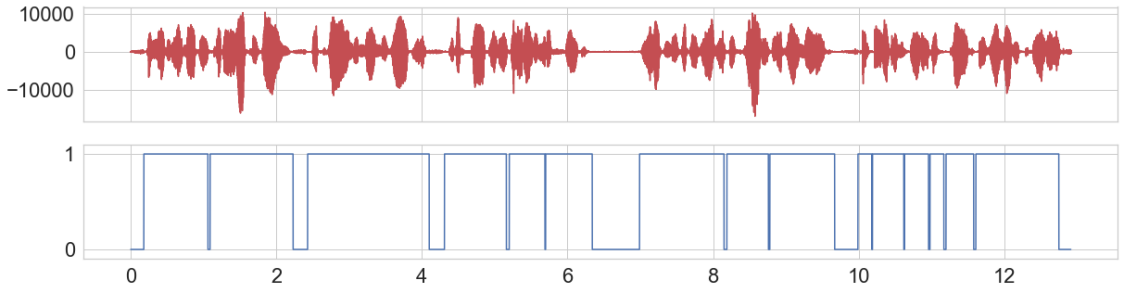
SPEAKER 1743 - PANEL 28
  data/01_raw/vad_data/1743-142912-0002.wav
  data/01_raw/vad_data/1743-142912-0002.json



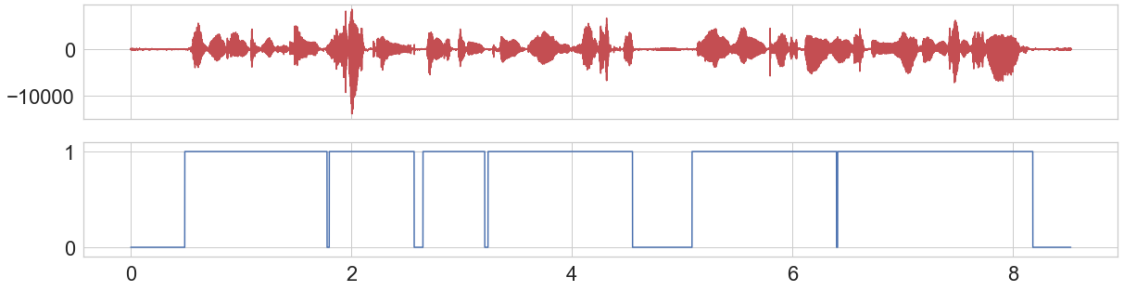
SPEAKER 1841 - PANEL 29
  data/01_raw/vad_data/1841-150351-0013.wav
  data/01_raw/vad_data/1841-150351-0013.json



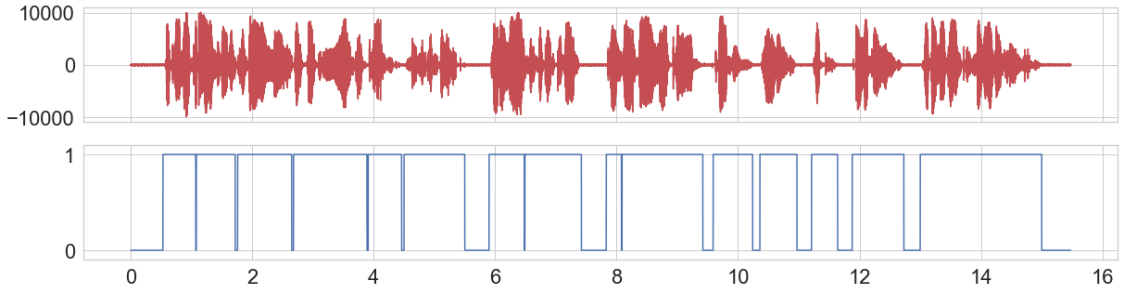
SPEAKER 1867 - PANEL 30
  data/01_raw/vad_data/1867-148436-0001.wav
  data/01_raw/vad_data/1867-148436-0001.json



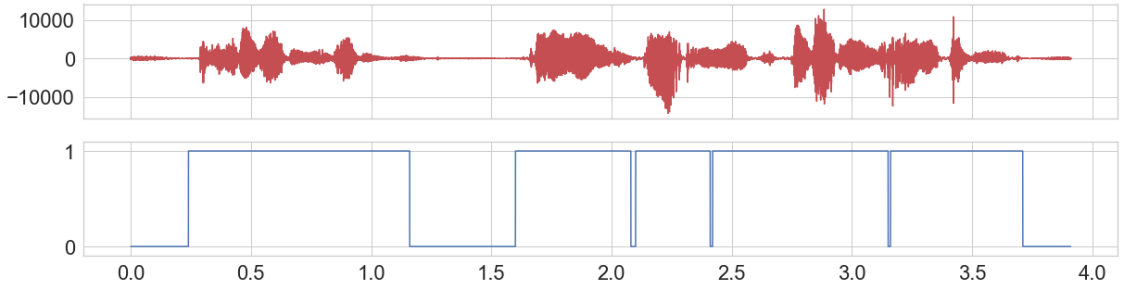
SPEAKER 1898 - PANEL 31
  data/01_raw/vad_data/1898-145702-0007.wav
  data/01_raw/vad_data/1898-145702-0007.json



SPEAKER 19 - PANEL 32
  data/01_raw/vad_data/19-198-0003.wav
  data/01_raw/vad_data/19-198-0003.json



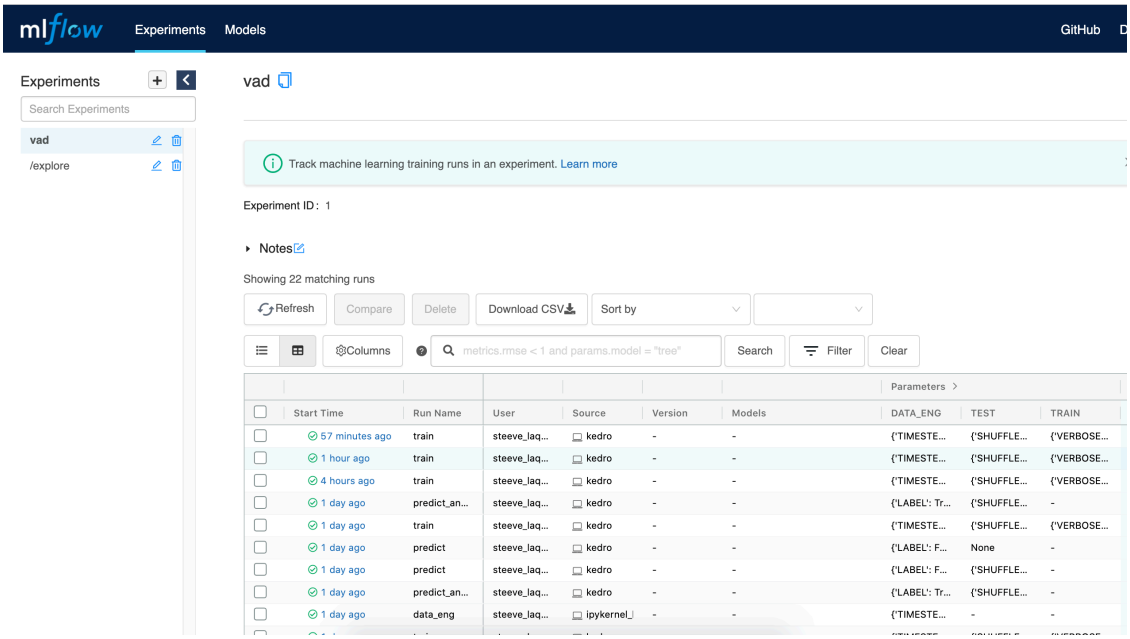
SPEAKER 1926 - PANEL 33
  data/01_raw/vad_data/1926-143879-0002.wav
  data/01_raw/vad_data/1926-143879-0002.json



1.10 Supplementary methods

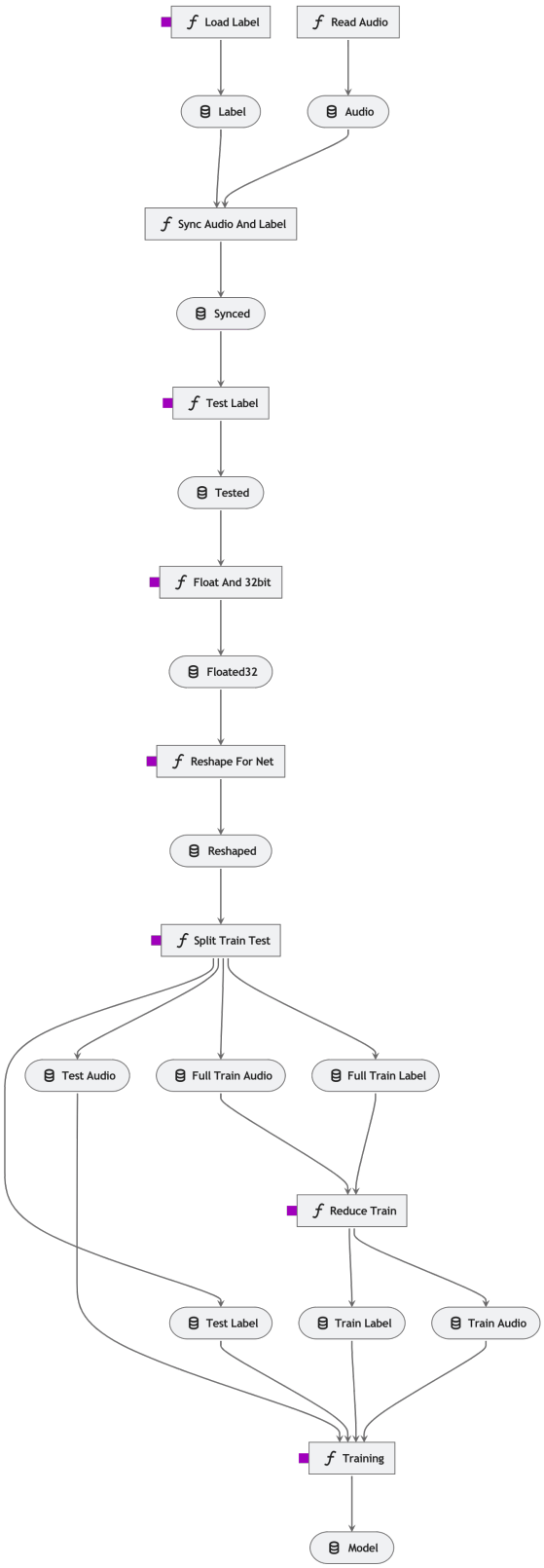
1.10.1 Experimental run tracking

Each run parameters was tracked with mlflow api.

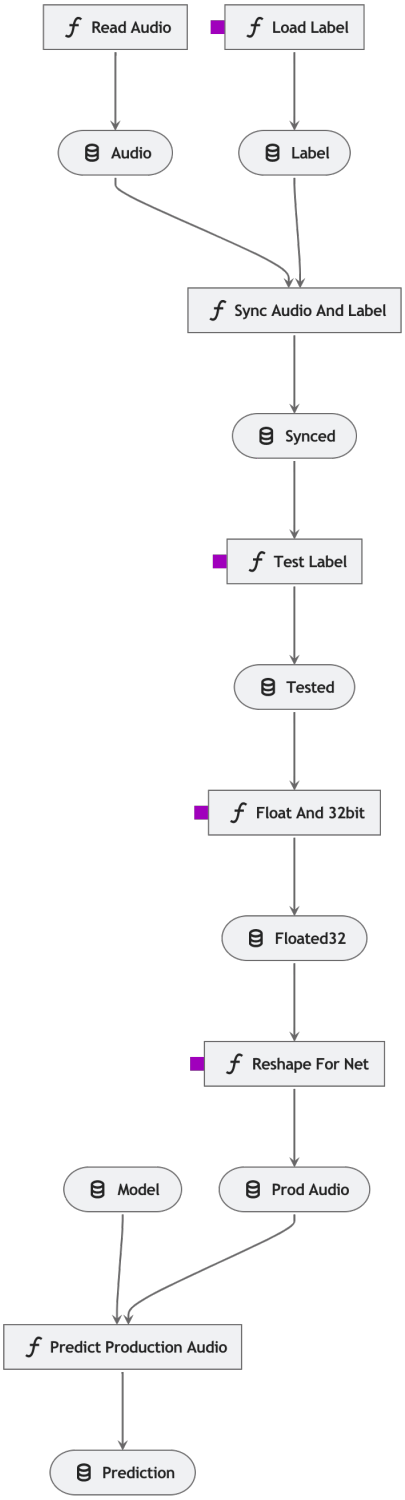


1.10.2 Training pipeline

we display below the directed acyclic graph of our training pipeline, plotted with the `kedro-viz` api.


Load Label
Read Audio
Label
Audio
Sync Audio And Label
Synced
Test Label
Tested
Float And 32bit
Floated32
Reshape For Net
Reshaped
Split Train Test
Test Audio
Full Train Audio
Full Train Label
Reduce Train
Test Label
Train Label
Train Audio
Training
Model

1.10.3 Inference pipeline



1.10.4 Evaluation pipeline