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

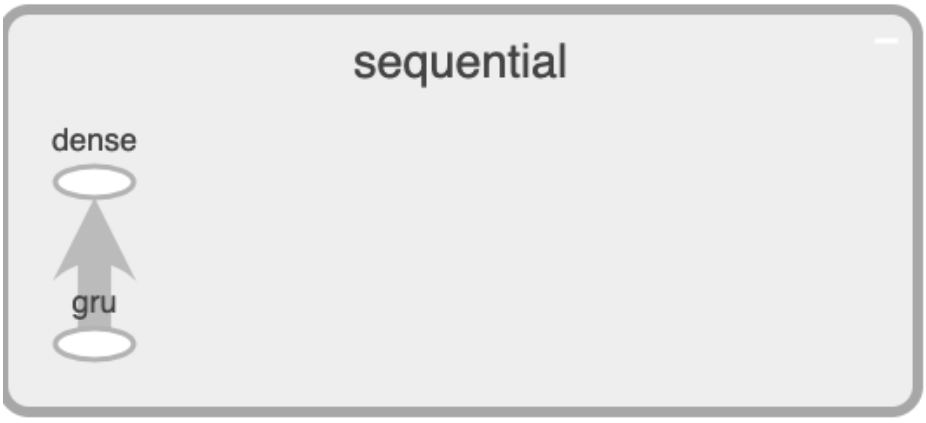
Neural network architecture: I formalized the task as a sequence classification task, where the choosen model must classify a current input audio data point as "speech" or "not speech" based on N preceding audio datapoints. Recurrent neural networks do exactly that. They use a recurrent architecture 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, you can launch a local server to visualize the graph in your web browser by running the command below in a linux (or MacOS unix) terminal:

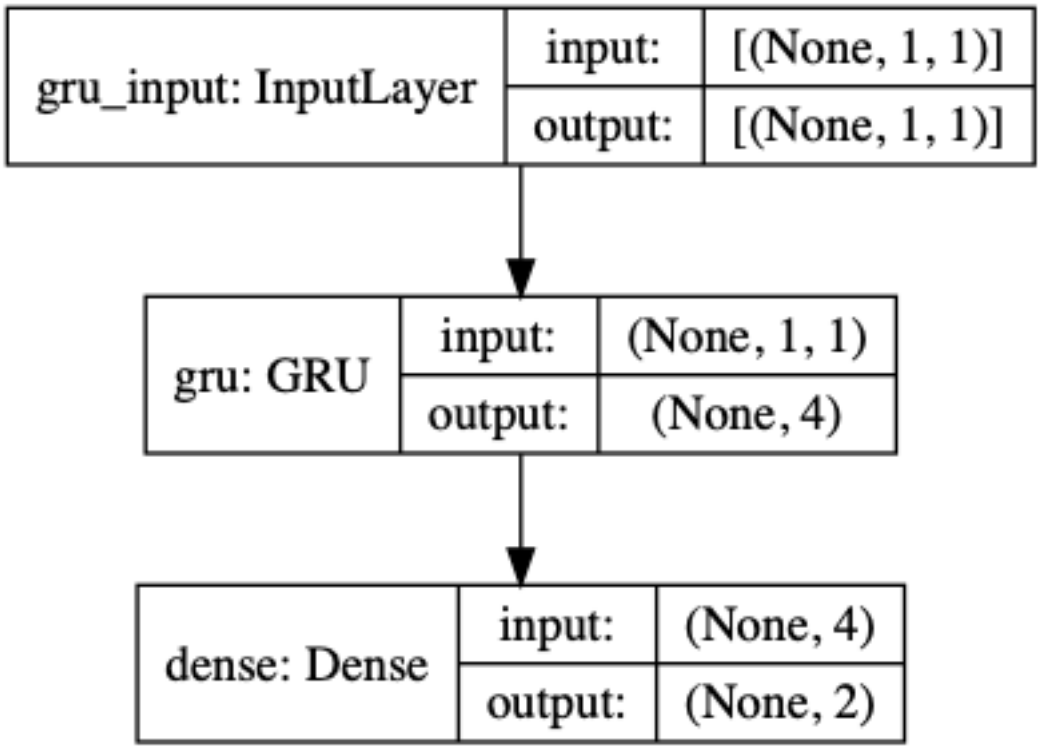
```
cd ../vad
tensorboard --logdir tbruns
```

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

I run several experiments changing the configurations of env/train_exps and env/predict_and_eval_exps pipelines.

```
kedro run --pipeline train --env train_exps
kedro run --pipeline predict_and_eval --env predict_and_eval_exps
```

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 assumed that a small 2.8 sec chunk of audio (20% of a 14 sec file) would be sufficiently representative of the entire dataset and extracted the first 2.8 sec of 19-198-0003.wav file. This assumption is useful as it allows for quick training (4 min instead of 1 hour for a full audio) although at the expense of generalization.

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 14 epochs. 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.

FRR on the full audio of another test subject (103-1240-0001) was 28% and precision 95% indicating that there is sufficient information in the data for a simple model to perform very well (**Experiment 1** in table). All FRRs reported during the experimental phase were done on the same test subject.

Lower resolution, better speed?:

I tested two bits resolutions: float16 and float32 on one subject audio file (103-1240-0001, **experiment 1 vs. 2**). Both trained as fast and FRR and precision on test did not differ. I did not test for other subjects due to time constraints.

Does learning from more look back timesteps improve model performance ?

I tested 1 and 10 timesteps (0.06 ms and 1.2 ms respectively with the actual 16 KHz sampling rate) in **experiment 1 vs 3**. - 10 timesteps took 5 min to train (compared to 50 sec for 1 timestep) but improved FRR 1.7 fold (from 28% to 16%), while keeping precision at 92% (from 95%). - I judged performance good enough to continue with quick iterations and did not train with larger timesteps.

Does a train/test split scheme in favor to the training set improve performance ?

I splitted the 2.8 sec audio into a 50/50 training/test set size and compared it with a 70/30 scheme. Increasing training set from 50% to 70% slightly but likely not significantly worsened FRR from 28% to 31% on another test subject (103-1240-0001, **experiment 1 vs 4**).

How long to convergence ?:

Training loss fell sharply and flattened at around 2 epochs which indicates that the model converges quickly. The validation loss indicates that loss tends to slowly increase over time, suggesting slight overfitting as training goes on.

The more data the better ?:

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

Training the model on the full 14 sec audio file improved FRRs by 18% (from 28% to 23%, tested

on speaker 103-1240, audio 103-1240-0001) but took 1 hour to train, while the 2.8 sec audio took 40 secs (**experiment 1 vs. 5**).

Precision slightly worsened from 95% to 93% indicating a slight increase in incorrect detections.

Which activation sigmoid (default) vs. biased Softmax ?

I used a biased softmax activation at the Dense classification layer to compensate class imbalance and compared the results with the default sigmoid activation function.

False rejection rate slightly increased with a biased softmax (from 28% to 31% on other subjects test audios, **experiment 1 vs. 5**) while precision increased from 95% to 96%.

I used mlflow to track training duration. Training was much faster and took 6.9 sec rather than 9.8 sec until stop while epoch to convergence did not differ (2 to 3 epochs).

Is Inference fast enough ?

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

The table below list the 6 experiments tested and the final setup retains for model training. The last 4 rows report the performance metrics (precision and FRR) and proxy ofr computational and optimization performances (training speed, training convergence, inference speed on a 16 sec test set).

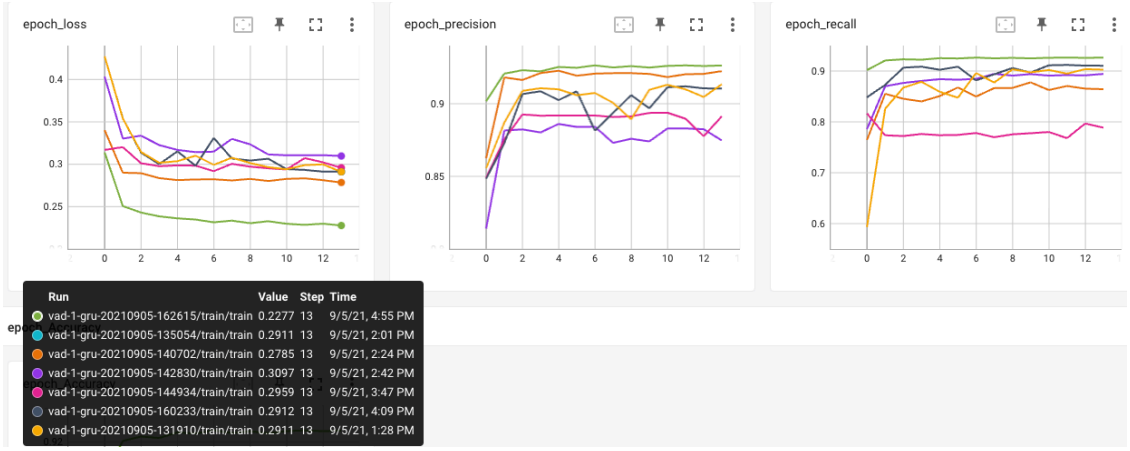
	Exp. 1	Exp. 2	Exp. 3	Exp. 4	Exp. 5	\
Activation	sigmoid	-	-	-	-	
epoch	14	-	-	-	-	
train size from split	0.5	-	-	0.7	-	
timesteps	1	-	10	-	-	
gru	1	-	-	-	-	
bias	FALSE	-	-	-	-	
resolution	32	16	-	-	-	
training chunking	0.2	-	-	-	1	
FRR on test	0.28	0.28	0.16	0.31	0.23	
precision on test	0.950	0.950	0.92	0.96	0.93	
training speed (min)	9.8	10.6	17	13.7	58.6	
training convergence speed (epochs)	2	2	1	3	3	
inference speed on 16 sec test (secs)	12.5	19	18.9	11.1	12	

	Exp. 6	Final setup
Activation	biased softmax	biased softmax
epoch	-	14
train size from split	-	0.7
timesteps	-	10
gru	-	1
bias	-	TRUE
resolution	-	16
training chunking	-	0.2
FRR on test	0.31	0.15
precision on test	0.96	0.91

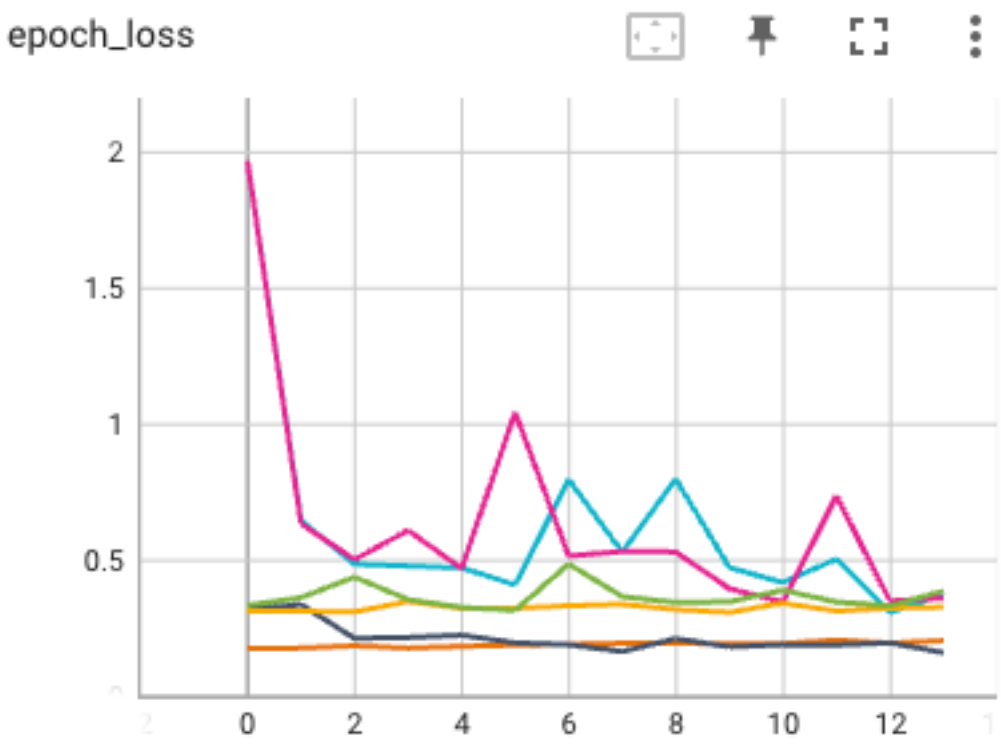
training speed (min)	6.9	29
training convergence speed (epochs)	3	2
inference speed on 16 sec test (secs)	12.4	24.3

I show below the evolution of training loss, validation precision and recall across the 14 epochs - light orange : experiment 1 - blue : experiment 2 - dark orange : experiment 3 - purple : experiment 4 - pink : experiment 5 - gray : experiment 6 - green : final setup

The final setup (green curves) was the best hyperparameter set, with the fastest drop in training loss, quick convergence with high recall and precisions. It was closely followed by experiment 3 (dark orange), indicating that timestep was the most important parameters to improve performance.



All validation losses quickly stabilized across training showing no sign of overfitting (increase in loss as training goes on),



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 14 secs audio file (i.e., 5 times more data) took 1 hour for just 18% improvement in FRR. All remaining training in the report rely on the 2.8 sec audio chunk from 19-198 speaker.

- 16 bits resolution

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:

- **batch size**: 1 batch
- **look back timesteps**: 10 timesteps

- **epochs**: 14 epochs
- **train/test split**: the 2.8 sec dataset was splitted on a 70% train and 30 % test dataset for a quick visual check of prediction performance generalisation on a small test set (see tensorboard test metrics. Here I rather report the model metrics calculated on the other subjects' entire dataset, which tells more on model generalisation performances.
- **Cross-validation**: the splitted training dataset was further split into 60% train and 40 % validation to display validation performance and loss curve in tensorboard

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: e.g., 118-47824-0000
  - women: e.g., 118-47824-0000
- Synthetic:
  - men: e.g., 1263-141777-0000

I also characterised speeches by their variety of amplitudes and pace:

- Normal vs fast pace: e.g., 1578-6379-0017

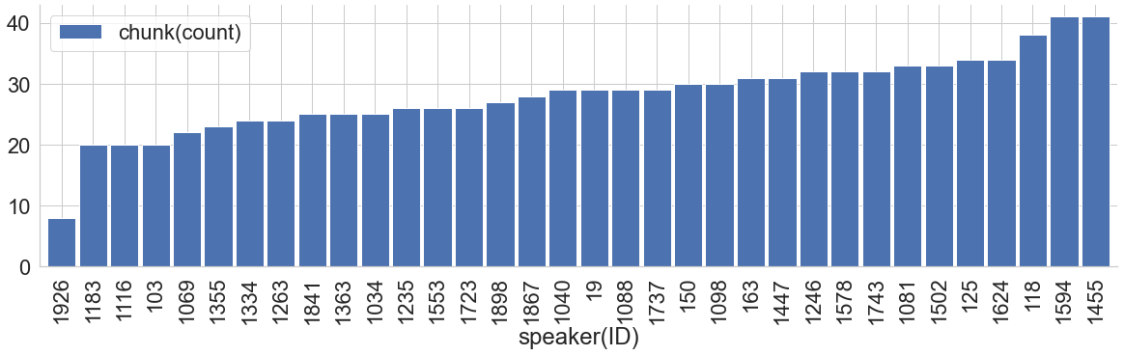
- Loud vs, low volume: e.g., 1447-130551-0019

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:

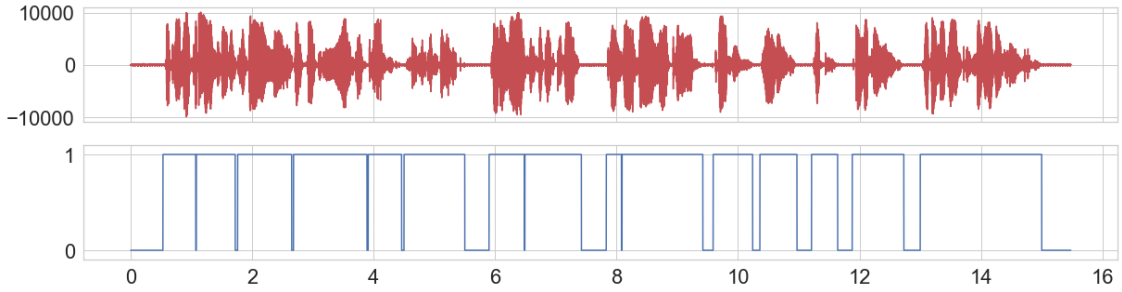
```
- the 957 label files were correctly mapped with the 957 audio files
```

```
- 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']
```

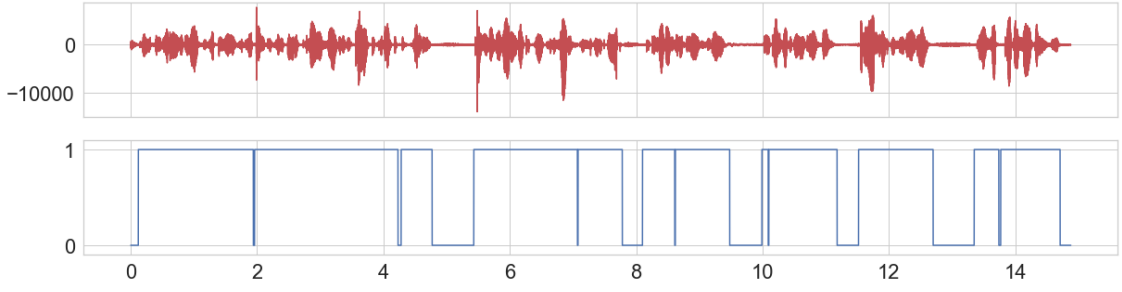


We show below a few interesting example chunks for two speakers. - Audio seem well labelled (see supplementary). - Background noise is low and stationary.

```
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:

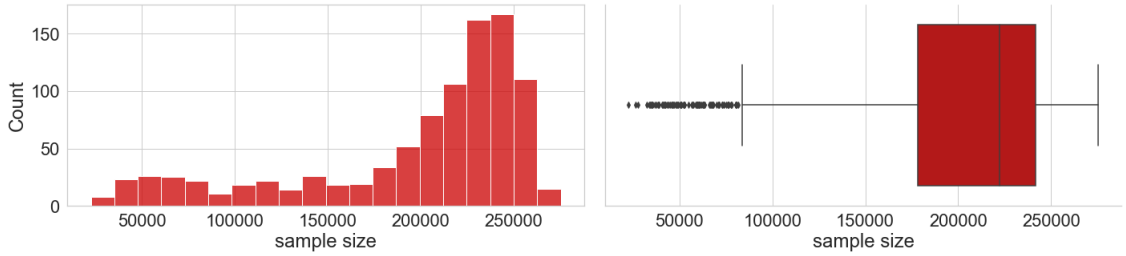
```
- loading duration: 3.24 sec
```

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:

$$speech = audio - noise$$

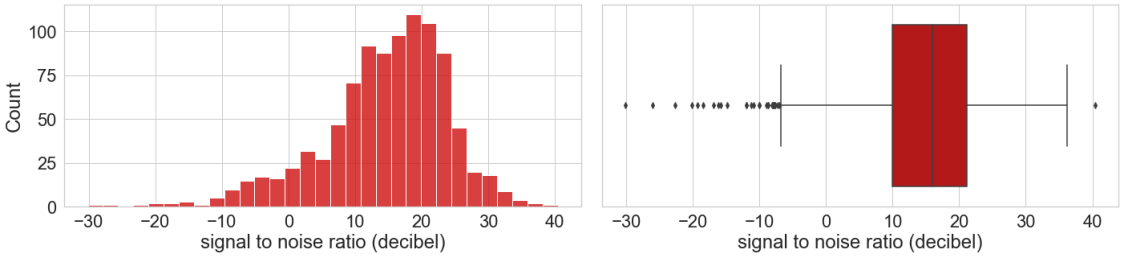
The signal to noise ratio in decibel is given by:

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

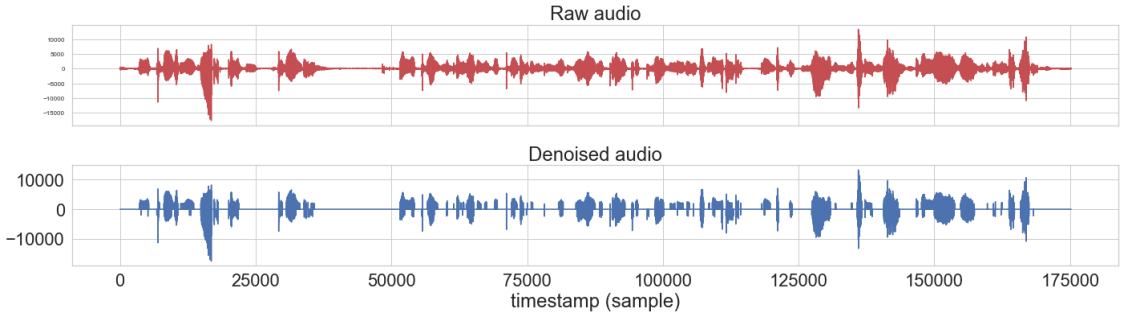
where speech_rms and noise_rms are the root mean square of the speech_signal and of the noise

SNR was spread over negative and positive values (-30 dB to +40 dB). The occurrence of negative SNR indicates that some recordings have noise with larger amplitudes the speech signal.

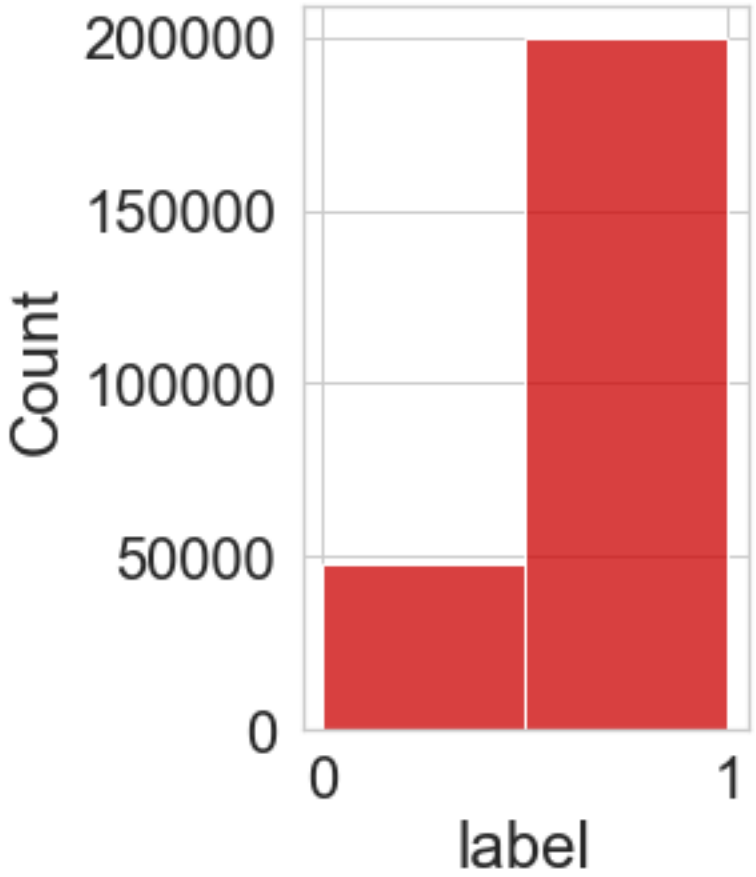
SNR was on average 15 dB with clear outliers towards the left tail (negative values, below the left interquartile range).



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.



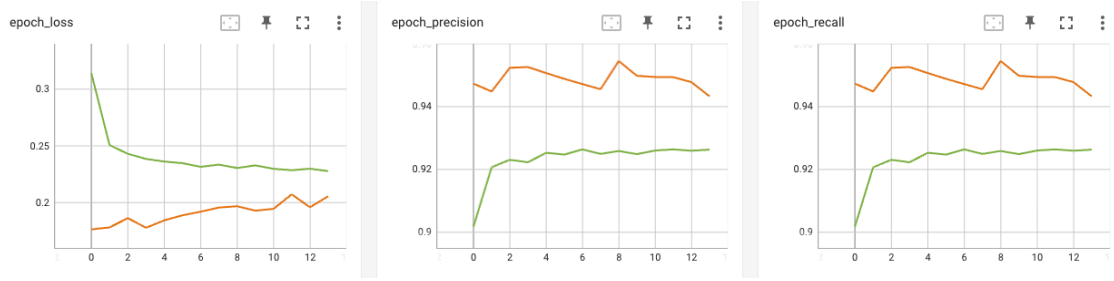
1.4.3 Speech and no-speech classes are imbalanced in the training dataset



1.4.4 The 1-layer GRU net converges within 7 epochs

I monitored training quality in Tensorboard api. I display below the training loss (green curve, left panel) calculated for 14 training epochs (x-axes ranging from 0 to 14) and validation loss (orange curve, left panel). I used the model referred to in the `final setup` section with 10 timesteps look-back (see logged runs in tensorboard for vad-1-gru-20210905-162615/train/train, vad-1-gru-20210905-162615/train/validation in tensorboard web api). The categorical cross-entropy training loss quickly fell sharply and flattened at about 2 epochs. A closer look at validation loss reveals a slight increase that indicate overfitting as training goes on after 3 epochs.

I also monitored recall to evaluate improvement in FRR. Recall quickly increased (thus FRR decreased) by 2% up to epoch 2 and took about 4 epochs to stabilise on the training set. Recall remained stable on the validation set.



1.4.5 Weights and biases follow well-behaved multimodal distributions

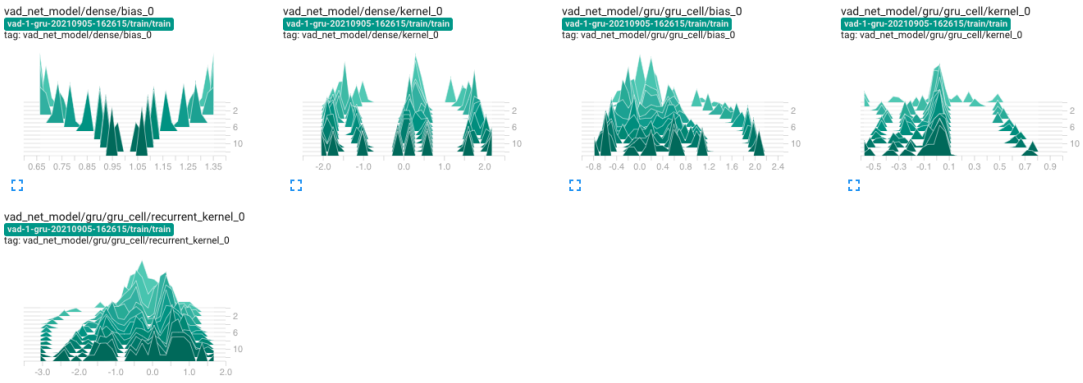
Both the Gru and dense layer weight distributions did not stay stuck at their initial values (no vanishing gradient problem), nor show extreme outliers. The weight and bias slowly but gradually converged toward new values.

Gru layer

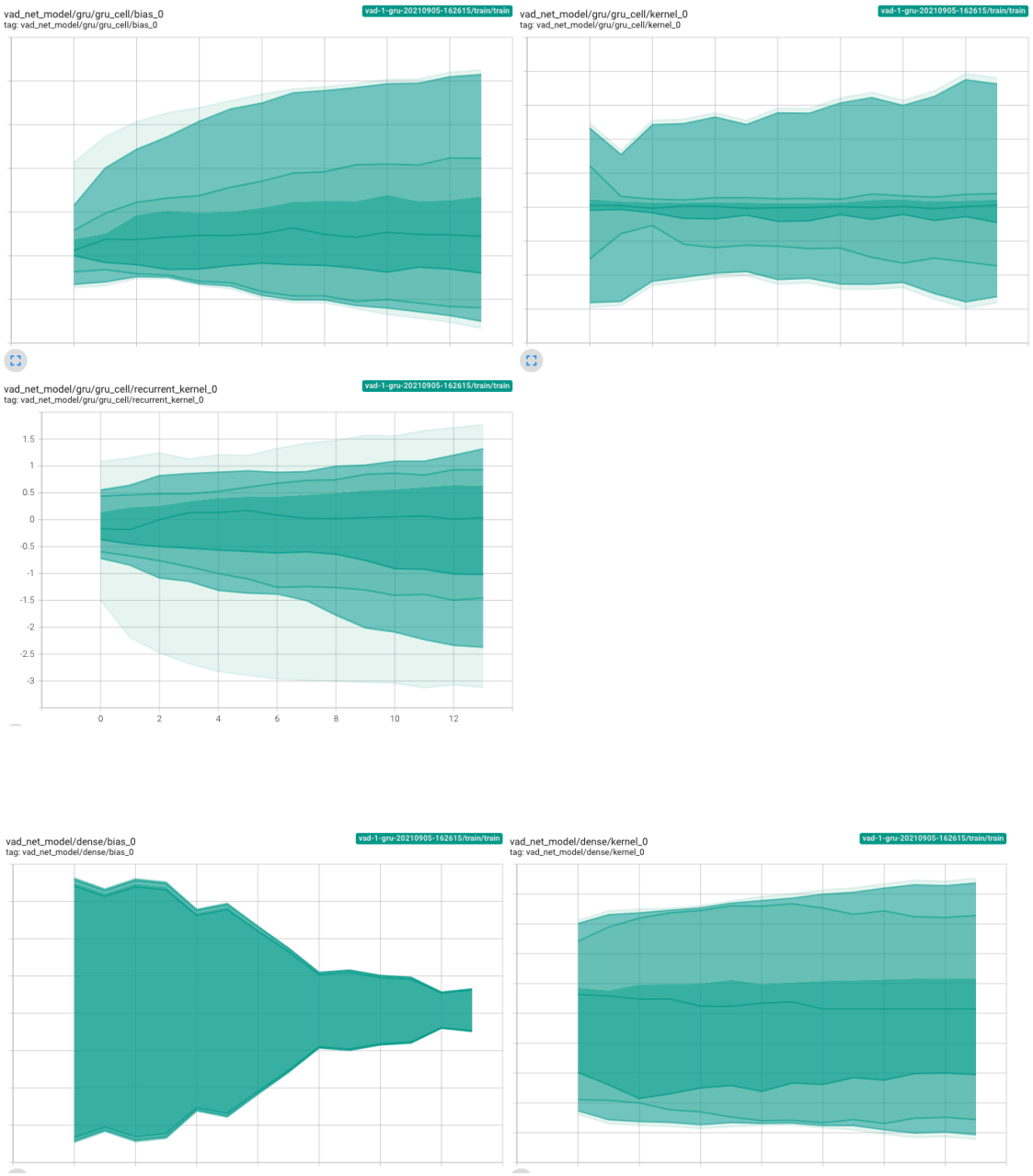
The GRU layer has 3 types of trainable parameters: - weights ("kernel_0") - recurrent weights ("kernel_0") - biases ("kernel_0")

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

- GRU'S weight and bias values:
  - The weight distribution was very sparse with central peak at 0 and a few weights spreading further from 0 toward negative and positive values as training progresses.
  - GRU biases and recurrent weights also spread out gradually toward larger range of negative and positive values as training progresses.
- Dense's weight and bias values:
  - Weights distrbution do not visibly change while bias concentrate toward a single value as learning progresses.



These changes in the bias and weights distributions are more apparent on the distribution plots of weight and bias vs. epochs (different shading columns represent the distributions' 90th, 60th percentiles …).



1.4.6 A 1-layer GRU model predicts well

The model generalized well on a single subject during the preliminary experiments described above. I tested whether the model conserved low FRRs and high precision on a larger test sample containing all 34 speakers' first audio file (which should took 14 min, i.e., 24 secs per audio file). Further analyses should assess whether the first audio is representative of speakers remaining recordings. I chose a sample and not the entire dataset because inference would take more than 3 hours on the 957 files.

The average FRR on test subjects' first audio files was as low as 15% and precision as high as 91%,

with a virtually null standard deviation indicating strongly reliable FRRs and generalizability to all speakers: human men, women and synthetic speakers.

	FRR	precision	f1	accuracy
count	34.00000	34.00000	34.00000	34.00000
mean	0.15084	0.91199	0.87946	0.79527
std	0.00000	0.00000	0.00000	0.00000
min	0.15084	0.91199	0.87946	0.79527
25%	0.15084	0.91199	0.87946	0.79527
50%	0.15084	0.91199	0.87946	0.79527
75%	0.15084	0.91199	0.87946	0.79527
max	0.15084	0.91199	0.87946	0.79527

1.4.7 Adding VAD modification

Enforcing a minimum speech time The minimum speech time observed in the dataset was surprisingly short (10 ms). I think a realistic minimum speech duration would be 1 secs, to say "hi" for example.

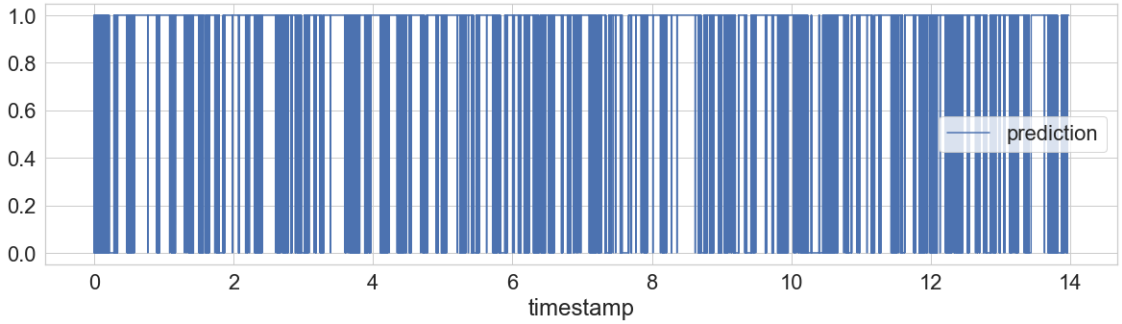
```
The minimum speech time derived from the data was: 10.0 ms
```

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	0.79527
precision	0.91199
recall	0.84916
f1	0.87946
false_rejection_rate	0.15084

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.

My custom loss function looked like this (see `src.vad.pipelines.train.nodes.MinSpeechLoss` for implementation details). The inconvenient is that penalty can only be calculated here by counting minimum speech violations in the last T timesteps.

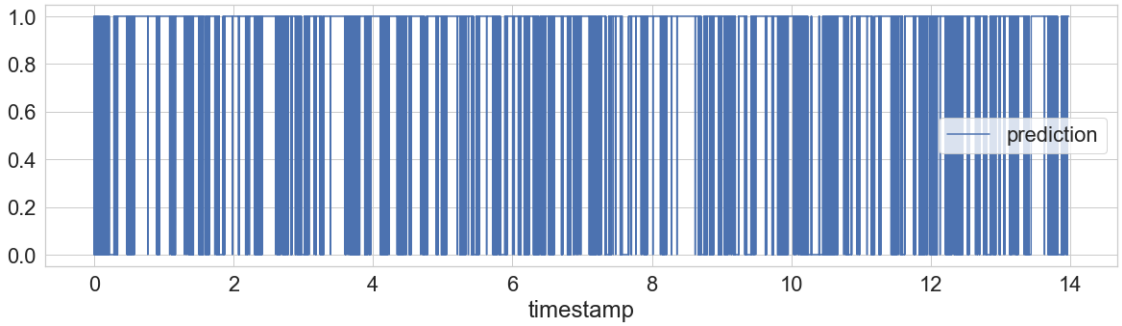
```
class MinSpeechLoss(tf.keras.losses.Loss):
    """Loss penalized to enforce a minimum speech time"""

    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 tensorflow's categorical cross-entropy loss and applied label smoothing. Label values are smoothed, relaxing the confidence on their binary values (0, 1). I used an intermediate value of `label_smoothing=0.5`.

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

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

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

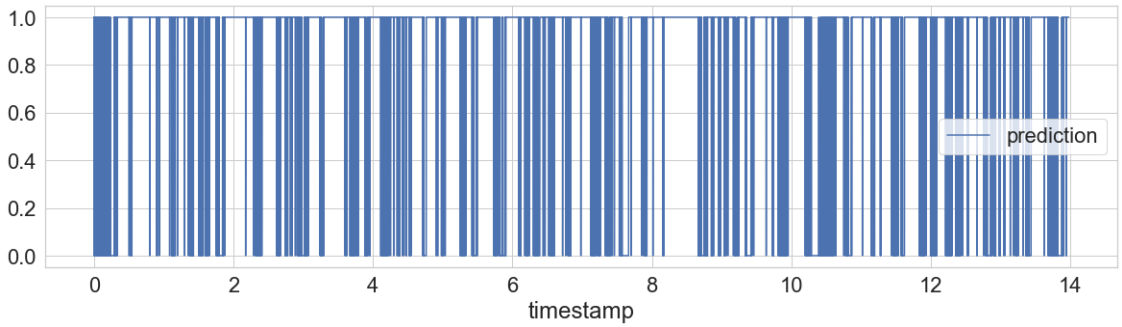


The performance metrics of the model on this example subject are:

	0
accuracy	0.79527
precision	0.91199
recall	0.84916
f1	0.87946
false_rejection_rate	0.15084

Predictions after 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 sligthly reduced FRR on test data for this subject at the expense of a little loss in precision.

	0
accuracy	0.81764
precision	0.90977
recall	0.87992
f1	0.89460
false_rejection_rate	0.12008

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 by 160 points in time and the end forward by 160 points.

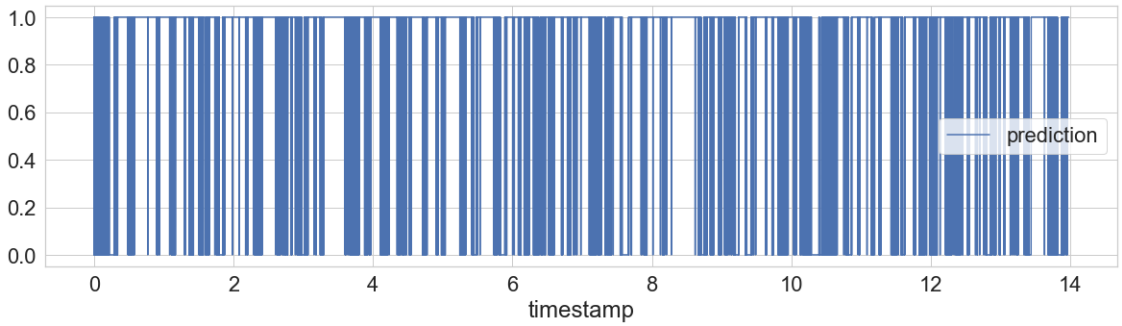
For example for speech interval 1:

- 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
```



Adding hangover slightly improved recall, thus FRR too compared to the basic model without any modification, but less than label smoothing (for this example 100 ms hangover and 0.5 label_smoothing).

	0
accuracy	0.80160
precision	0.91141
recall	0.85779
f1	0.88379
false_rejection_rate	0.14221

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.

The audio recording were very pure, recorded in no-background noise conditions which is not representative of more realistic noisy conditions. I could improve model robustness to realistic conditions by augmenting the dataset with noisier audio such as:

- `MUSAN`: music, speech, noise (11 GB)
- Ava-speech dataset (8,9)

To better disambiguate the effect of various types of noise on the performance, I could trained the model on speeches engineered with diverse background noises using:

- `QUT-Noise`: noise backgrounds (13, 14)
- `TIMIT`: clean speech

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

The codebased should be unit-tested. A src/tests/test_run.py has been initiated and can be run with:

```
pytest src/tests/test_run.py
```

I should work on matching several requirements :
- response latency: - improving sensitivity by further lowering FRR - reduce computational cost at training and inference - reduce model memory footprint to enable deployment a constrained resource device such as an iphone or BlackBerry Pi, for example: - model quantization - serializing the model to a lighter file format such as tensorflow lite

1.7 References

(13) https://research.qut.edu.au/saivt/databases/qut-noise-databases-and-protocols/

(14) Dean, D., Sridharan, S., Vogt, R., & Mason, M. (2010). The QUT-NOISE-TIMIT corpus for evaluation of voice activity detection algorithms. In Proceedings of the 11th Annual Conference of the International Speech Communication Association (pp. 3110-3113). International Speech Communication Association.

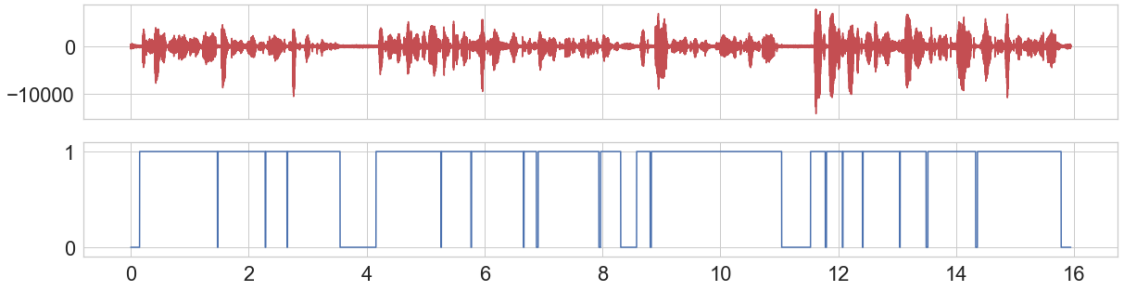
(15) https://research.google.com/ava/download.html

(16) Gu, C., Sun, C., Ross, D. A., Vondrick, C., Pantofaru, C., Li, Y., … & Malik, J. (2018). Ava: A video dataset of spatio-temporally localized atomic visual actions. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 6047-6056).

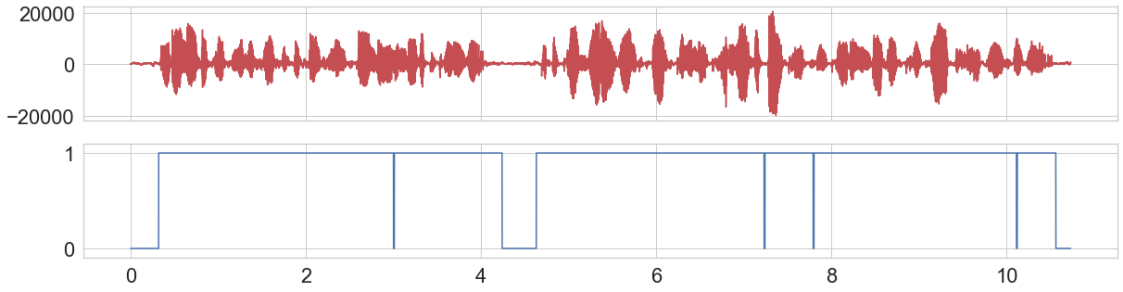
1.8 Supplementary results

1.9 Speakers' first audio

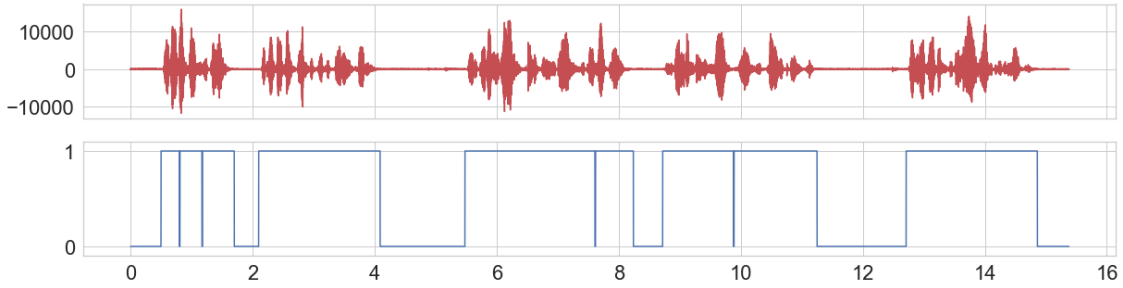
```
SPEAKER 103 - PANEL 0
  data/01_raw/vad_data/103-1240-0001.wav
  data/01_raw/vad_data/103-1240-0001.json
```



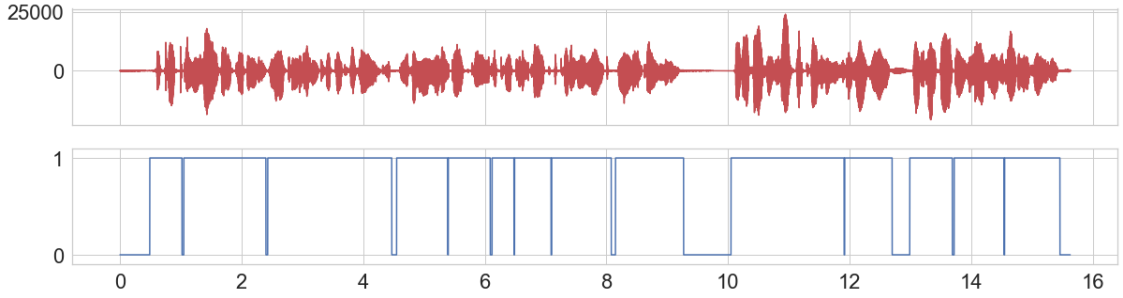

```
SPEAKER 1034 - PANEL 1
  data/01_raw/vad_data/1034-121119-0005.wav
  data/01_raw/vad_data/1034-121119-0005.json
```



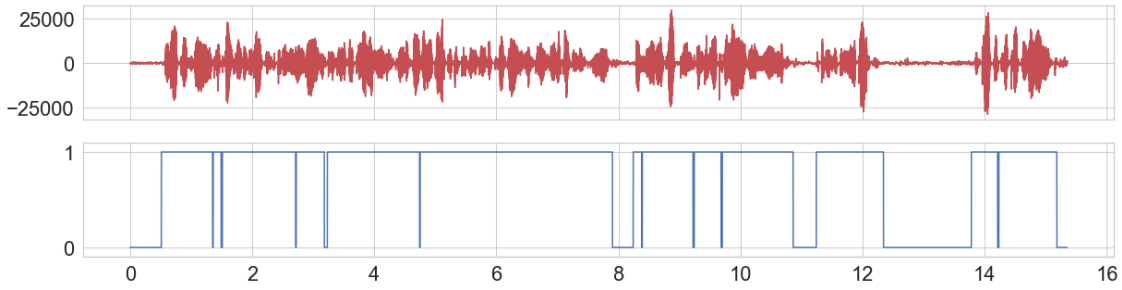
SPEAKER 1040 - PANEL 2
  data/01_raw/vad_data/1040-133433-0001.wav
  data/01_raw/vad_data/1040-133433-0001.json



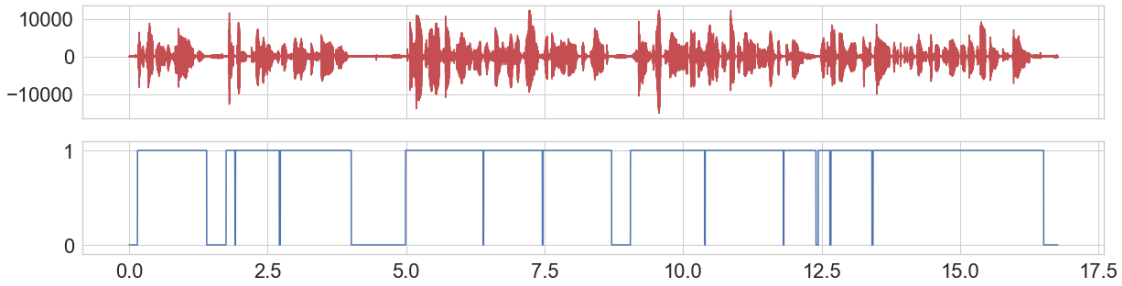
SPEAKER 1069 - PANEL 3
  data/01_raw/vad_data/1069-133699-0000.wav
  data/01_raw/vad_data/1069-133699-0000.json



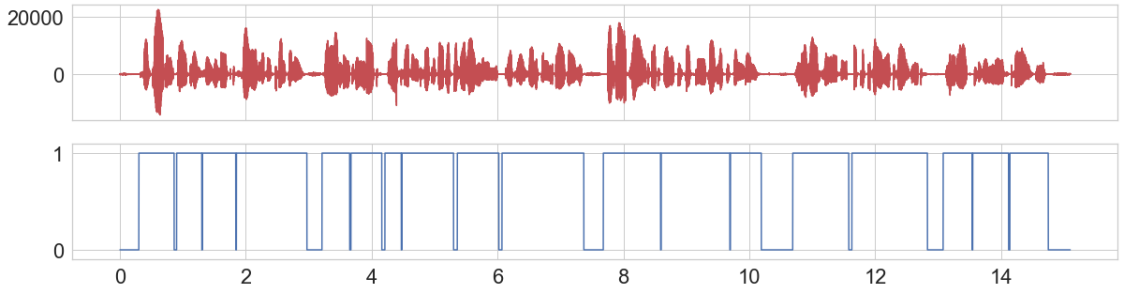
SPEAKER 1081 - PANEL 4
  data/01_raw/vad_data/1081-125237-0007.wav
  data/01_raw/vad_data/1081-125237-0007.json



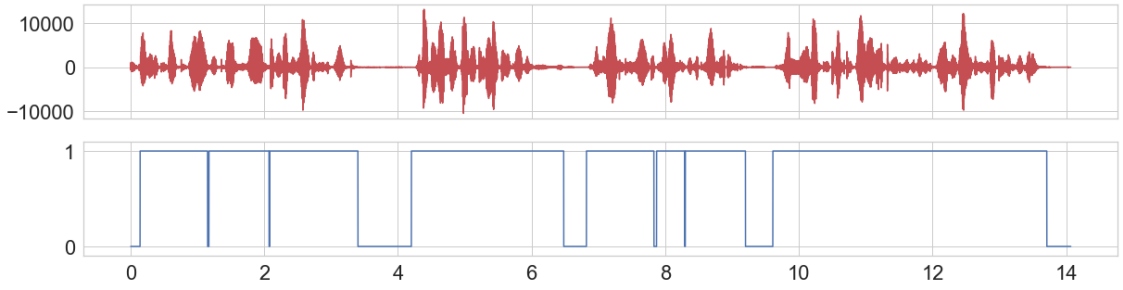
SPEAKER 1088 - PANEL 5
  data/01_raw/vad_data/1088-129236-0003.wav
  data/01_raw/vad_data/1088-129236-0003.json



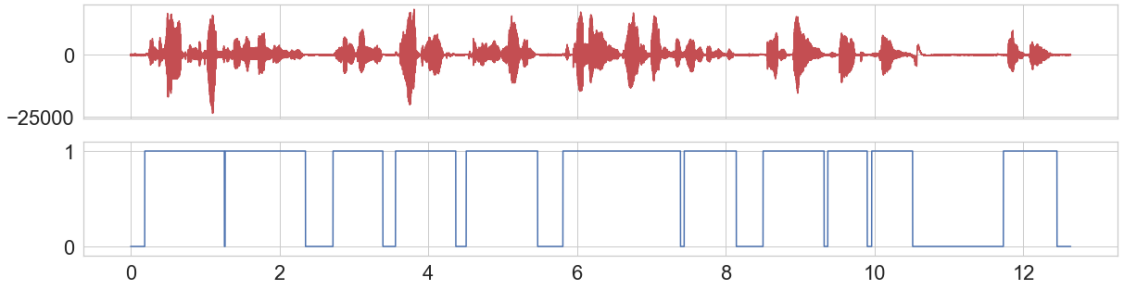
SPEAKER 1098 - PANEL 6
  data/01_raw/vad_data/1098-133695-0001.wav
  data/01_raw/vad_data/1098-133695-0001.json



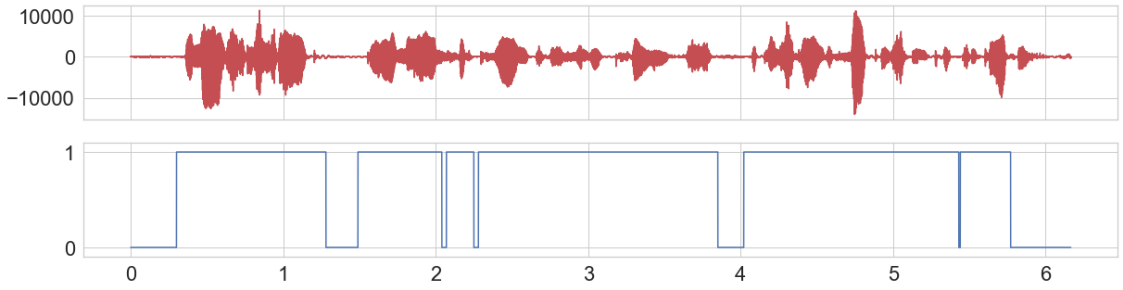
SPEAKER 1116 - PANEL 7
  data/01_raw/vad_data/1116-132847-0003.wav
  data/01_raw/vad_data/1116-132847-0003.json



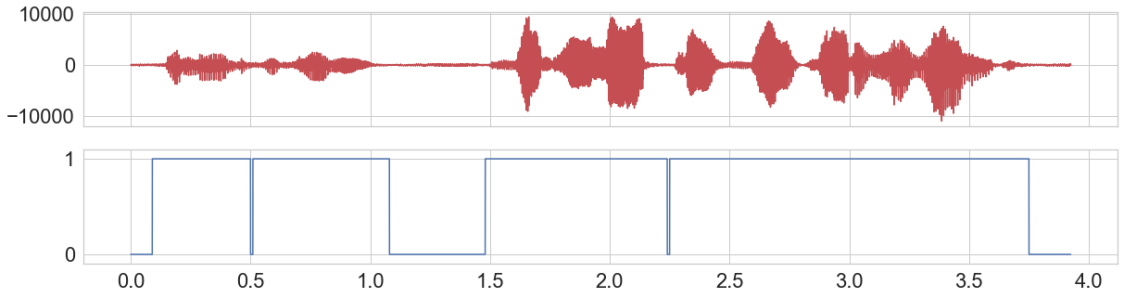
SPEAKER 118 - PANEL 8
  data/01_raw/vad_data/118-121721-0005.wav
  data/01_raw/vad_data/118-121721-0005.json



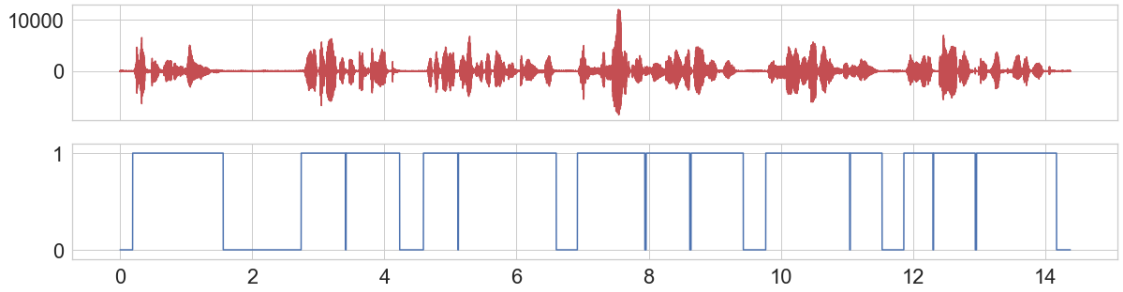
SPEAKER 1183 - PANEL 9
  data/01_raw/vad_data/1183-124566-0000.wav
  data/01_raw/vad_data/1183-124566-0000.json



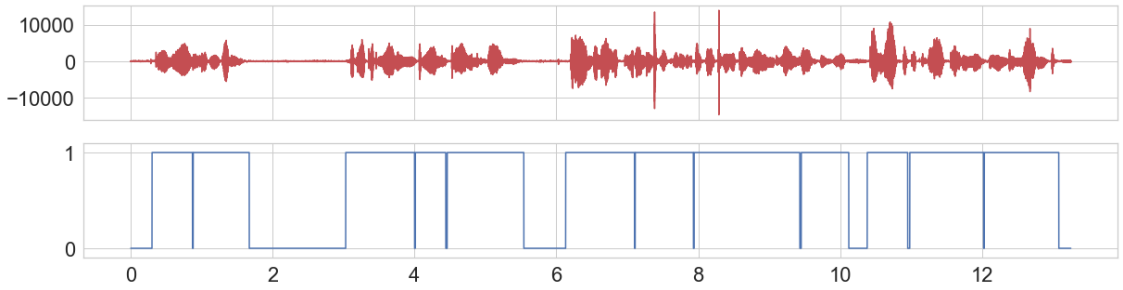
SPEAKER 1235 - PANEL 10
  data/01_raw/vad_data/1235-135883-0007.wav
  data/01_raw/vad_data/1235-135883-0007.json



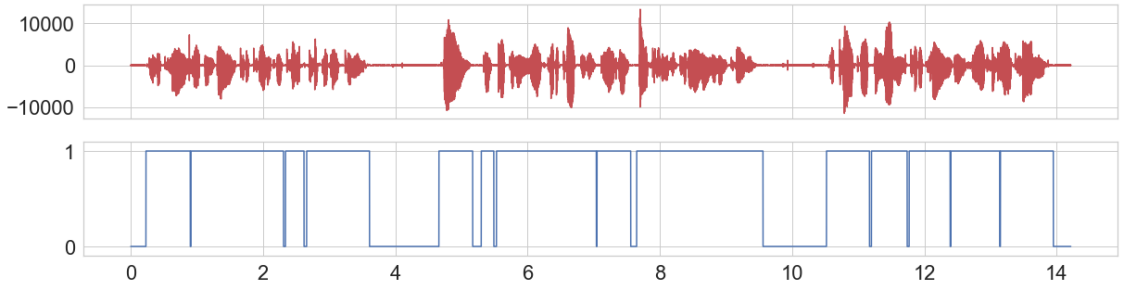
SPEAKER 1246 - PANEL 11
  data/01_raw/vad_data/1246-124548-0000.wav
  data/01_raw/vad_data/1246-124548-0000.json



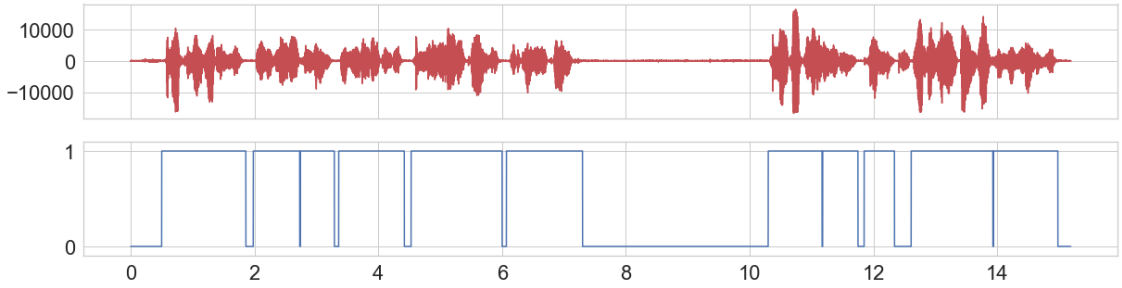
SPEAKER 125 - PANEL 12
  data/01_raw/vad_data/125-121124-0000.wav
  data/01_raw/vad_data/125-121124-0000.json



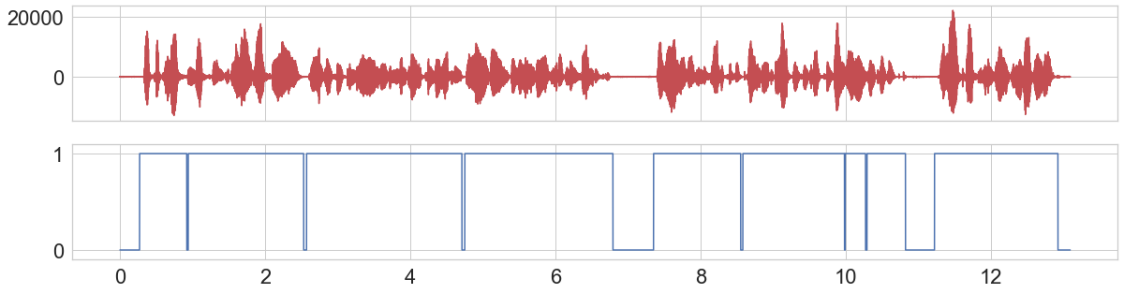
SPEAKER 1263 - PANEL 13
  data/01_raw/vad_data/1263-138246-0000.wav
  data/01_raw/vad_data/1263-138246-0000.json



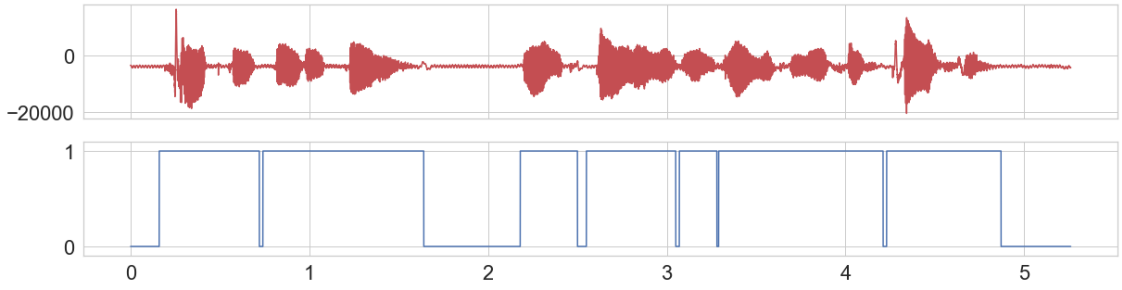
SPEAKER 1334 - PANEL 14
  data/01_raw/vad_data/1334-135589-0011.wav
  data/01_raw/vad_data/1334-135589-0011.json



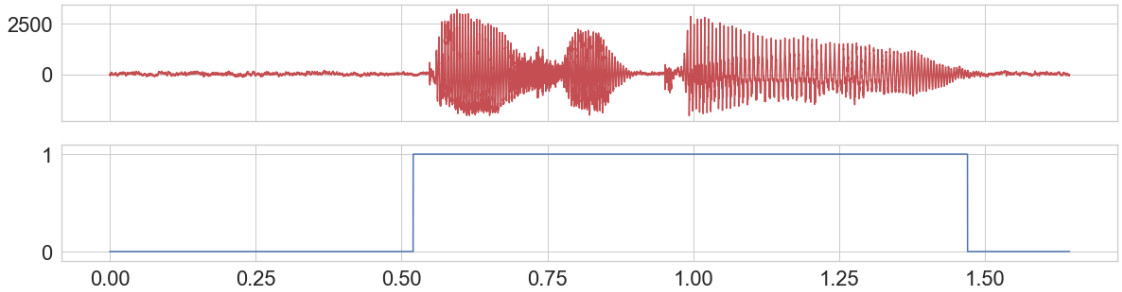
SPEAKER 1355 - PANEL 15
  data/01_raw/vad_data/1355-39947-0014.wav
  data/01_raw/vad_data/1355-39947-0014.json



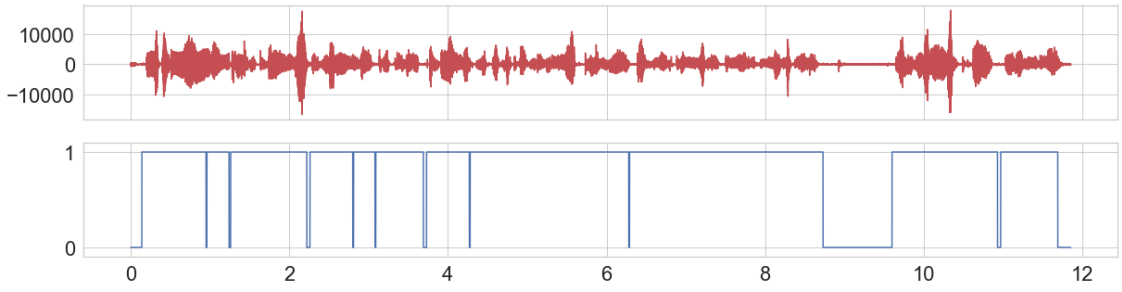
SPEAKER 1363 - PANEL 16
  data/01_raw/vad_data/1363-135842-0000.wav
  data/01_raw/vad_data/1363-135842-0000.json



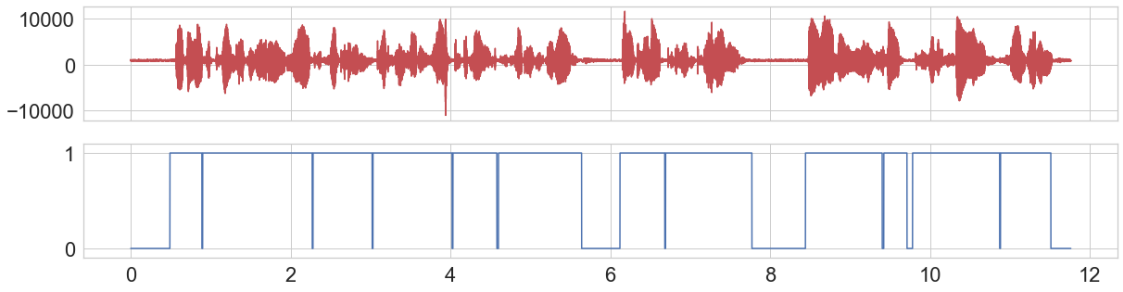
SPEAKER 1447 - PANEL 17
  data/01_raw/vad_data/1447-130550-0000.wav
  data/01_raw/vad_data/1447-130550-0000.json



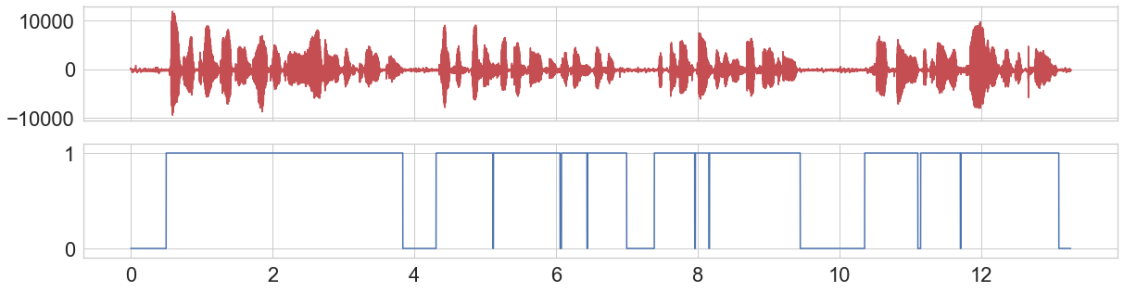
SPEAKER 1455 - PANEL 18
  data/01_raw/vad_data/1455-134435-0007.wav
  data/01_raw/vad_data/1455-134435-0007.json



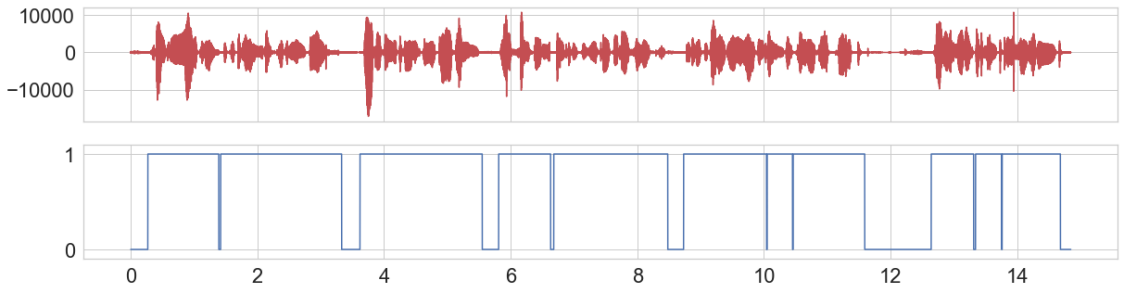
SPEAKER 150 - PANEL 19
  data/01_raw/vad_data/150-126107-0001.wav
  data/01_raw/vad_data/150-126107-0001.json



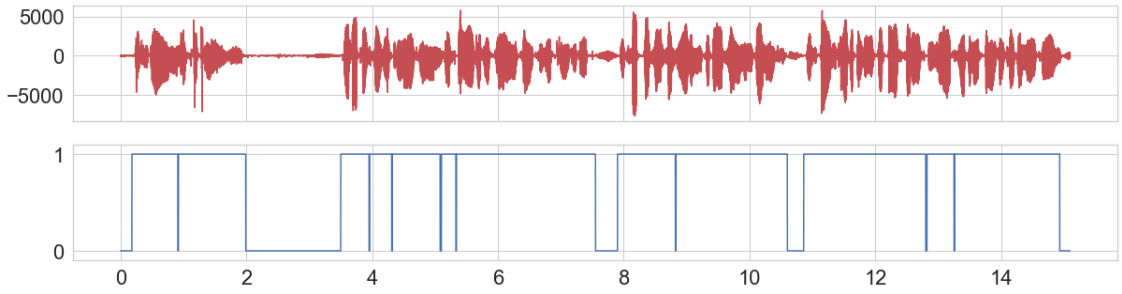
SPEAKER 1502 - PANEL 20
  data/01_raw/vad_data/1502-122615-0007.wav
  data/01_raw/vad_data/1502-122615-0007.json



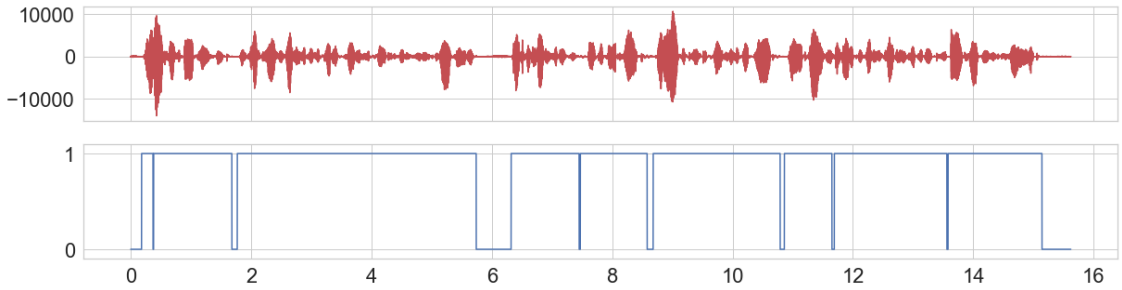
SPEAKER 1553 - PANEL 21
  data/01_raw/vad_data/1553-140047-0002.wav
  data/01_raw/vad_data/1553-140047-0002.json



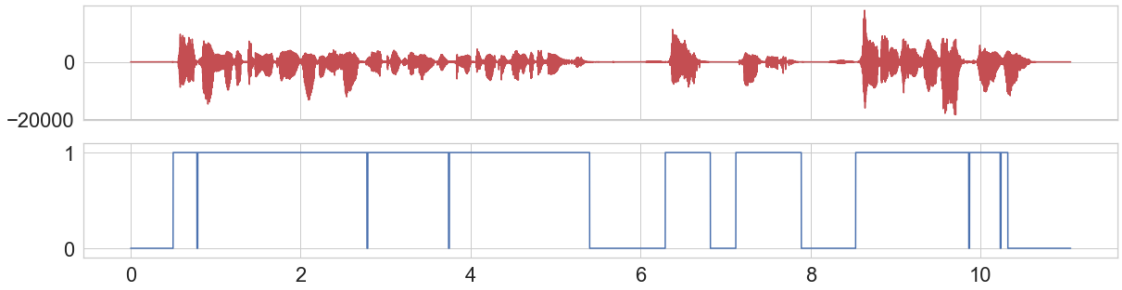
SPEAKER 1578 - PANEL 22
  data/01_raw/vad_data/1578-140045-0000.wav
  data/01_raw/vad_data/1578-140045-0000.json



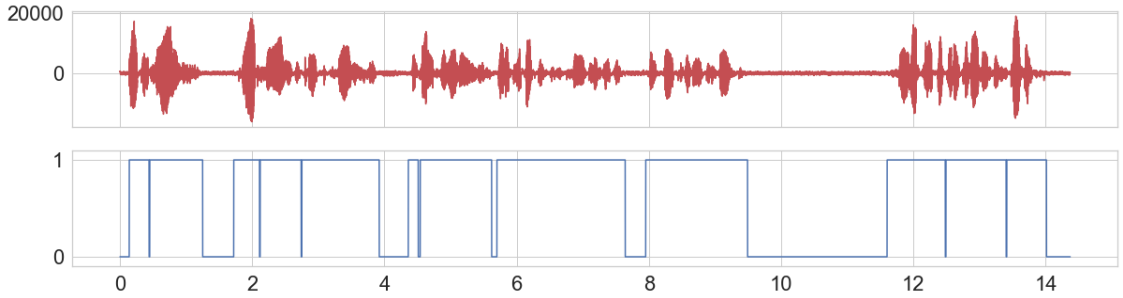

```
SPEAKER 1594 - PANEL 23
  data/01_raw/vad_data/1594-135914-0004.wav
  data/01_raw/vad_data/1594-135914-0004.json
```



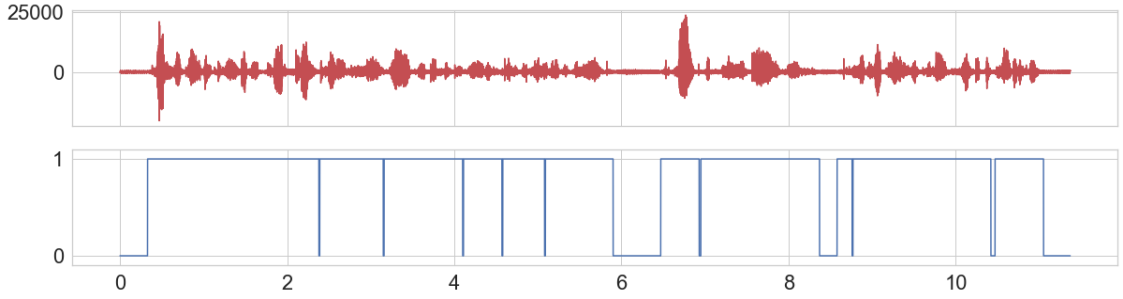

```
SPEAKER 1624 - PANEL 24
  data/01_raw/vad_data/1624-142933-0003.wav
  data/01_raw/vad_data/1624-142933-0003.json
```



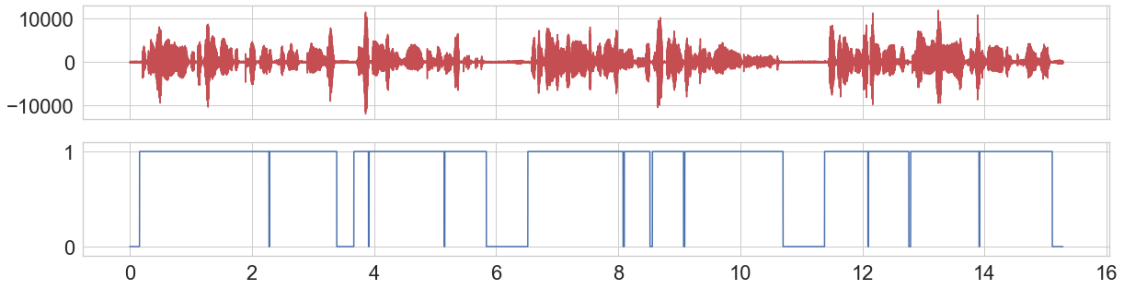

```
SPEAKER 163 - PANEL 25
  data/01_raw/vad_data/163-121908-0006.wav
  data/01_raw/vad_data/163-121908-0006.json
```



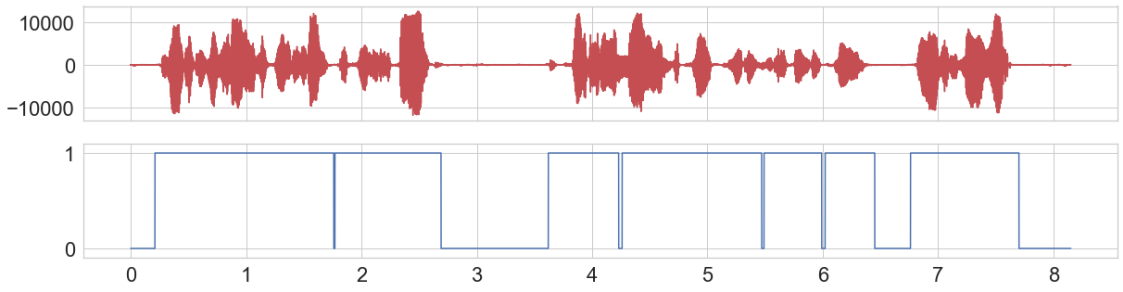
SPEAKER 1723 - PANEL 26
  data/01_raw/vad_data/1723-141149-0005.wav
  data/01_raw/vad_data/1723-141149-0005.json



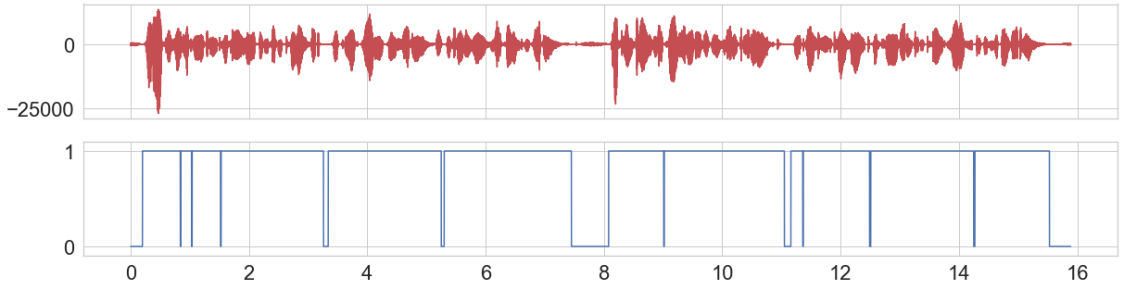
SPEAKER 1737 - PANEL 27
  data/01_raw/vad_data/1737-142396-0000.wav
  data/01_raw/vad_data/1737-142396-0000.json



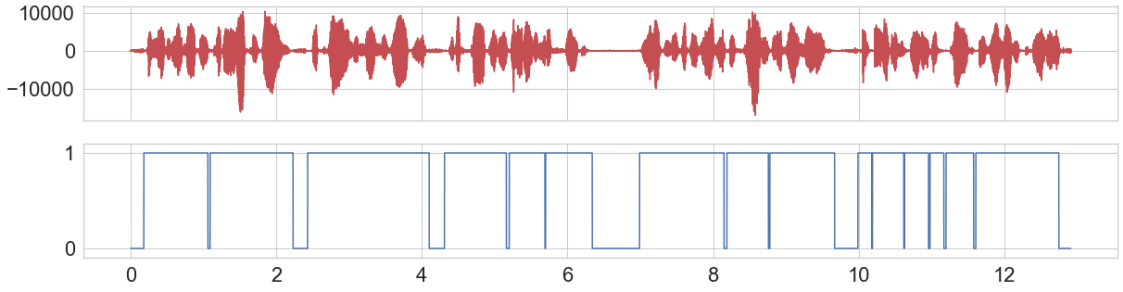
SPEAKER 1743 - PANEL 28
  data/01_raw/vad_data/1743-142912-0002.wav
  data/01_raw/vad_data/1743-142912-0002.json



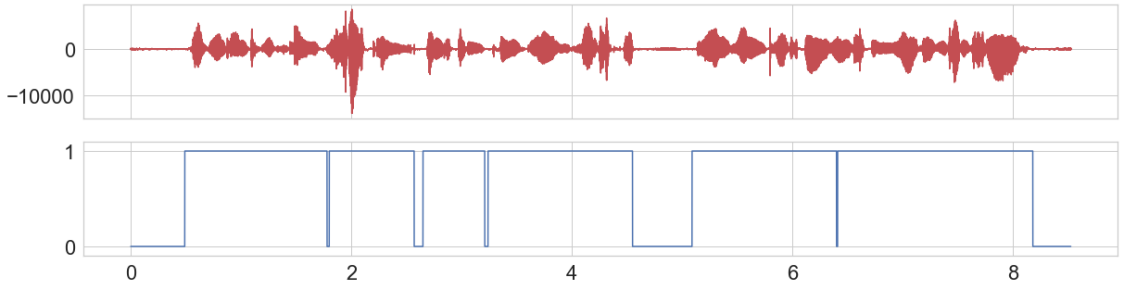
SPEAKER 1841 - PANEL 29
  data/01_raw/vad_data/1841-150351-0013.wav
  data/01_raw/vad_data/1841-150351-0013.json



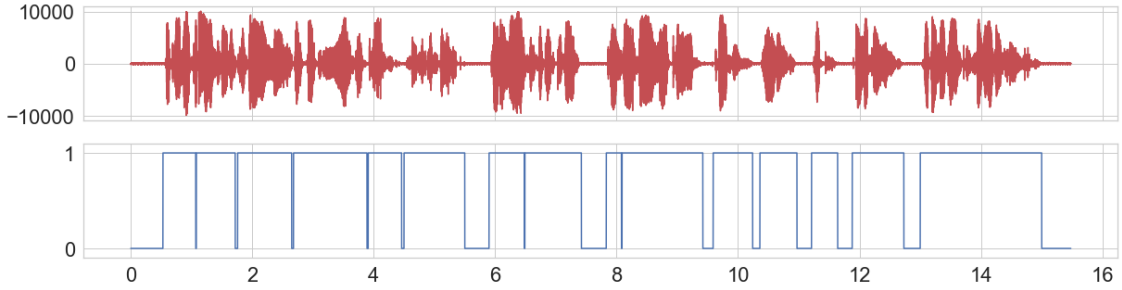
SPEAKER 1867 - PANEL 30
  data/01_raw/vad_data/1867-148436-0001.wav
  data/01_raw/vad_data/1867-148436-0001.json



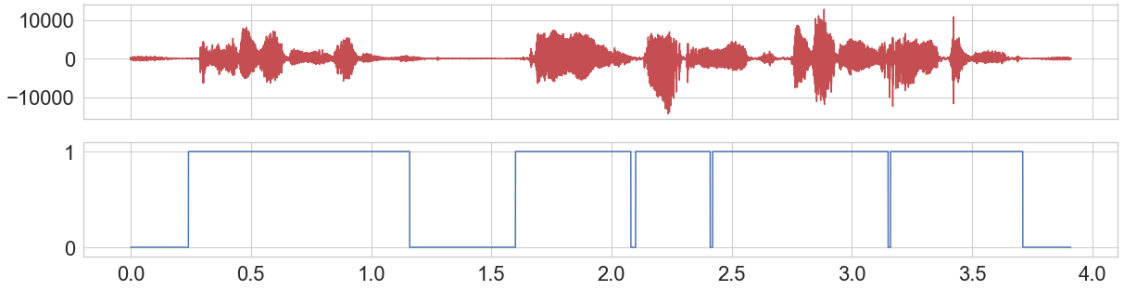
SPEAKER 1898 - PANEL 31
  data/01_raw/vad_data/1898-145702-0007.wav
  data/01_raw/vad_data/1898-145702-0007.json




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



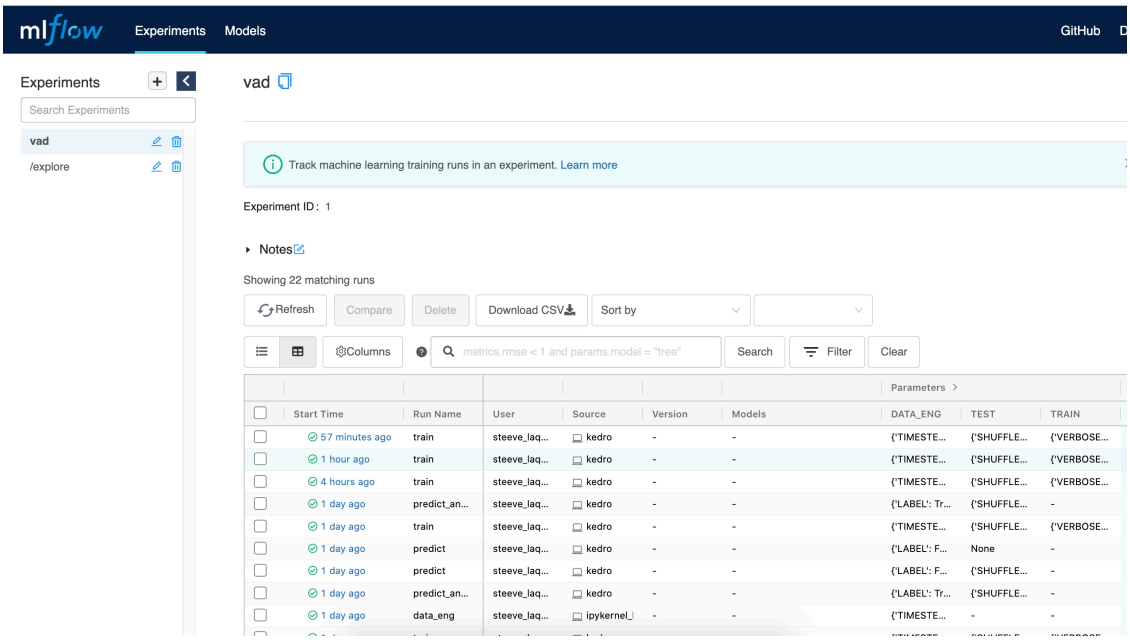

```
SPEAKER 1926 - PANEL 33
  data/01_raw/vad_data/1926-143879-0002.wav
  data/01_raw/vad_data/1926-143879-0002.json
```



1.10 Supplementary methods

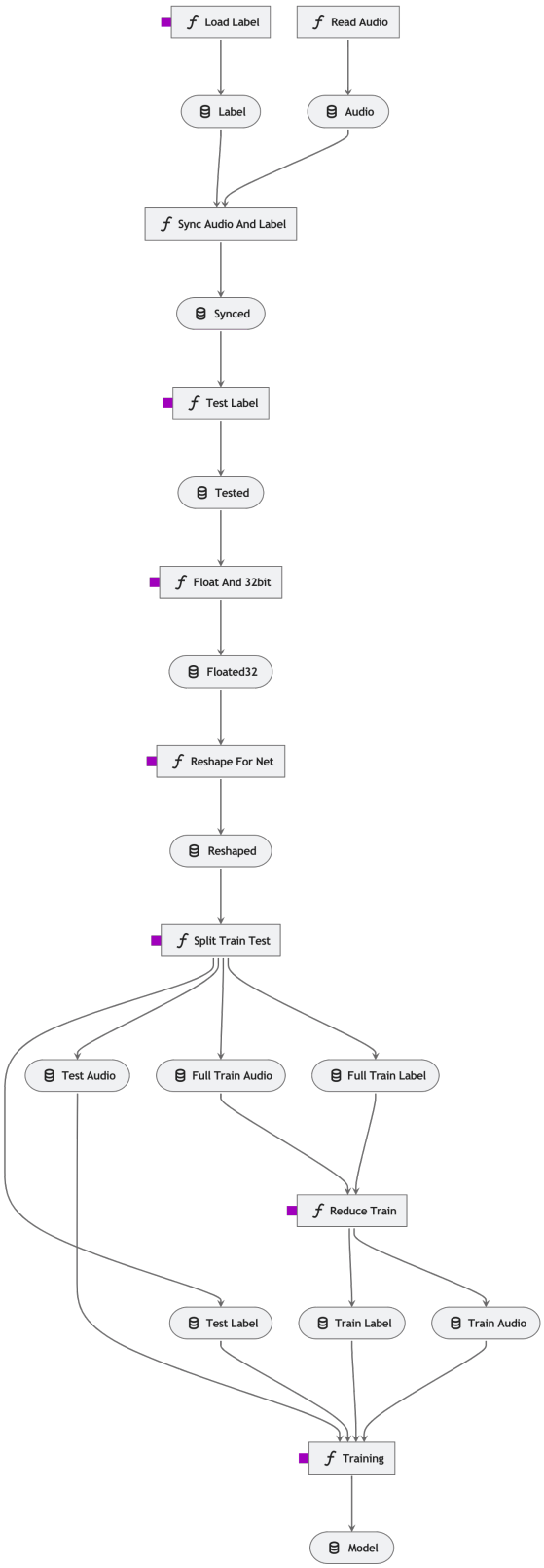
1.10.1 Experimental run tracking

Each run parameters was tracked with mlflow api.

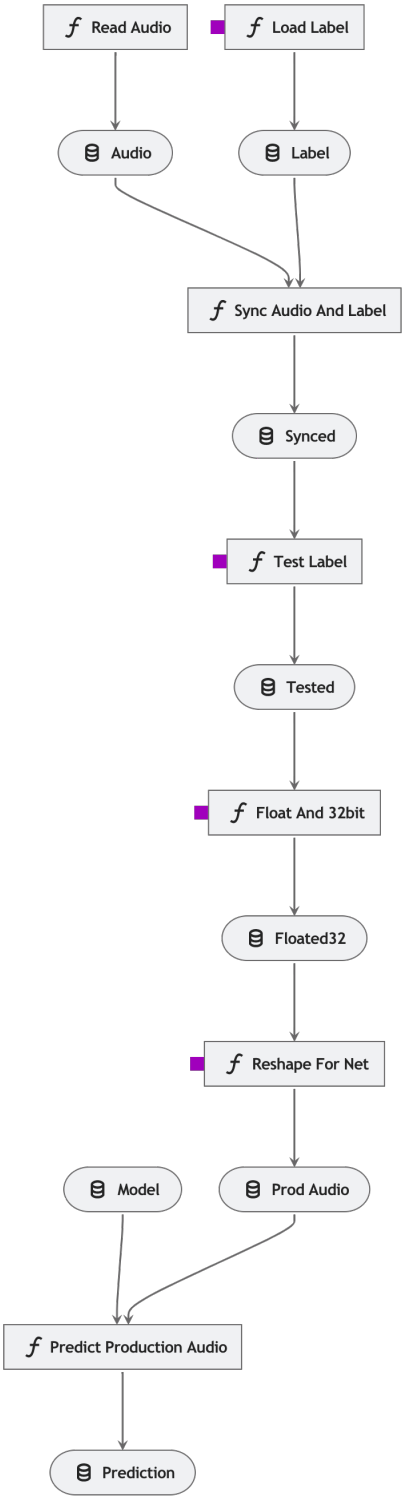


1.10.2 Training pipeline

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