auditok Documentation

Release v0.2.0

Amine Sehili

Mar 01, 2021

Getting started

`auditok` is an **Audio Activity Detection** tool that can process online data (read from an audio device or from standard input) as well as audio files. It can be used as a command line program or by calling its API.

CHAPTER 1

Installation

A basic version of `auditok` will run with standard Python (>=3.4). However, without installing additional dependencies, `auditok` can only deal with audio files in *wav* or *raw* formats. if you want more features, the following packages are needed:

- pydub : read audio files in popular audio formats (ogg, mp3, etc.) or extract audio from a video file.
- pyaudio : read audio data from the microphone and play audio back.
- tqdm : show progress bar while playing audio clips.
- matplotlib : plot audio signal and detections.
- numpy : required by matplotlib. Also used for some math operations instead of standard python if available.

Install the latest stable version with pip:

```
sudo pip install auditok
```

Install with the latest development version from github:

```
pip install git+https://github.com/amsehili/auditok
```

or

```
git clone https://github.com/amsehili/auditok.git
cd auditok
python setup.py install
```

CHAPTER 2

Loading audio data

Audio data is loaded with the `load()` function which can read from audio files, the microphone or use raw audio data.

2.1 From a file

If the first argument of `load()` is a string, it should be a path to an audio file.

```
import auditok
region = auditok.load("audio.ogg")
```

If input file contains a raw (headerless) audio data, passing *audio_format="raw"* and other audio parameters (*sampling_rate*, *sample_width* and *channels*) is mandatory. In the following example we pass audio parameters with their short names:

```
region = auditok.load("audio.dat",
                      audio_format="raw",
                      sr=44100, # alias for `sampling_rate`
                      sw=2      # alias for `sample_width`
                      ch=1      # alias for `channels`
                      )
```

2.2 From a *bytes* object

If the type of the first argument *bytes*, it's interpreted as raw audio data:

```
sr = 16000
sw = 2
ch = 1
data = b"\0" * sr * sw * ch
```

(continues on next page)

(continued from previous page)

```
region = auditok.load(data, sr=sr, sw=sw, ch=ch)
print(region)
```

output:

```
AudioRegion(duration=1.000, sampling_rate=16000, sample_width=2, channels=1)
```

2.3 From the microphone

If the first argument is *None*, `load()` will try to read data from the microphone. Audio parameters, as well as the *max_read* parameter are mandatory:

```
sr = 16000
sw = 2
ch = 1
five_sec_audio = load(None, sr=sr, sw=sw, ch=ch, max_read=5)
print(five_sec_audio)
```

output:

```
AudioRegion(duration=5.000, sampling_rate=16000, sample_width=2, channels=1)
```

2.4 Skip part of audio data

If the *skip* parameter is > 0, `load()` will skip that leading amount of audio data:

```
import auditok
region = auditok.load("audio.ogg", skip=2) # skip the first 2 seconds
```

This argument must be 0 when reading from the microphone.

CHAPTER 3

Basic split example

In the following we'll use the `split()` function to tokenize an audio file, requiring that valid audio events be at least 0.2 second long, at most 4 seconds long and contain a maximum of 0.3 second of continuous silence. Limiting the size of detected events to 4 seconds means that an event of, say, 9.5 seconds will be returned as two 4-second events plus a third 1.5-second event. Moreover, a valid event might contain many *silences* as far as none of them exceeds 0.3 second.

`split()` returns a generator of `AudioRegion`. An `AudioRegion` can be played, saved, repeated (i.e., multiplied by an integer) and concatenated with another region (see examples below). Notice that `AudioRegion` objects returned by `split()` have a `start` a `stop` information stored in their meta data that can be accessed like *object.meta.start*.

```
import auditok

# split returns a generator of AudioRegion objects
audio_regions = auditok.split(
    "audio.wav",
    min_dur=0.2,     # minimum duration of a valid audio event in seconds
    max_dur=4,       # maximum duration of an event
    max_silence=0.3, # maximum duration of tolerated continuous silence within an␣
↪event
    energy_threshold=55 # threshold of detection
)

for i, r in enumerate(audio_regions):

    # Regions returned by `split` have 'start' and 'end' metadata fields
    print("Region {i}: {r.meta.start:.3f}s -- {r.meta.end:.3f}s".format(i=i, r=r))

    # play detection
    # r.play(progress_bar=True)

    # region's metadata can also be used with the `save` method
    # (no need to explicitly specify region's object and `format` arguments)
    filename = r.save("region_{meta.start:.3f}-{meta.end:.3f}.wav")
```

(continues on next page)

(continued from previous page)

```
    print("region saved as: {}".format(filename))
```

output example:

```
Region 0: 0.700s -- 1.400s
region saved as: region_0.700-1.400.wav
Region 1: 3.800s -- 4.500s
region saved as: region_3.800-4.500.wav
Region 2: 8.750s -- 9.950s
region saved as: region_8.750-9.950.wav
Region 3: 11.700s -- 12.400s
region saved as: region_11.700-12.400.wav
Region 4: 15.050s -- 15.850s
region saved as: region_15.050-15.850.wav
```

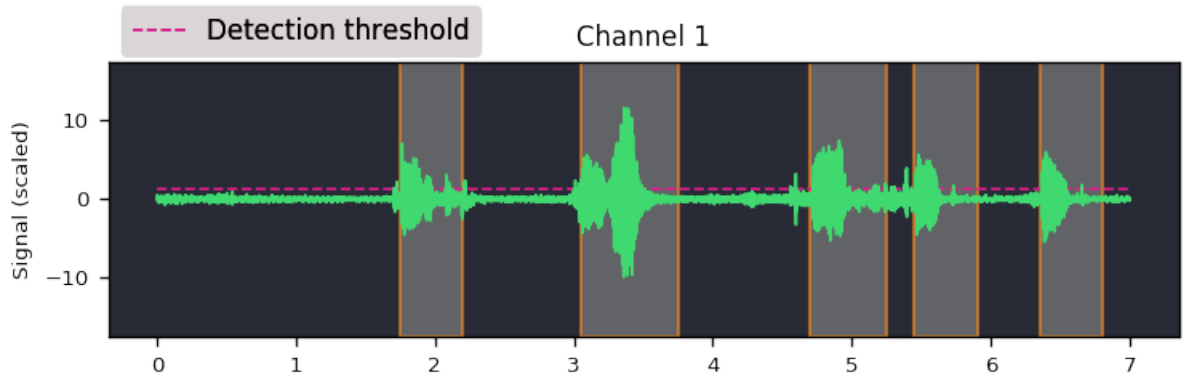
CHAPTER 4

Split and plot

Visualize audio signal and detections:

```
import auditok
region = auditok.load("audio.wav") # returns an AudioRegion object
regions = region.split_and_plot(...) # or just region.splitp()
```

output figure:



CHAPTER 5

Read and split data from the microphone

If the first argument of `split()` is None, audio data is read from the microphone (requires pyaudio):

```
import auditok

sr = 16000
sw = 2
ch = 1
eth = 55 # alias for energy_threshold, default value is 50

try:
    for region in auditok.split(input=None, sr=sr, sw=sw, ch=ch, eth=eth):
        print(region)
        region.play(progress_bar=True) # progress bar requires `tqdm`
except KeyboardInterrupt:
     pass
```

`split()` will continue reading audio data until you press `Ctrl-C`. If you want to read a specific amount of audio data, pass the desired number of seconds with the *max_read* argument.

CHAPTER 6

Accessing recorded data after split

Using a `Recorder` object you can get hold of acquired audio data:

```
import auditok

sr = 16000
sw = 2
ch = 1
eth = 55 # alias for energy_threshold, default value is 50

rec = auditok.Recorder(input=None, sr=sr, sw=sw, ch=ch)

try:
    for region in auditok.split(rec, sr=sr, sw=sw, ch=ch, eth=eth):
        print(region)
        region.play(progress_bar=True) # progress bar requires `tqdm`
except KeyboardInterrupt:
     pass

rec.rewind()
full_audio = load(rec.data, sr=sr, sw=sw, ch=ch)
# alternatively you can use
full_audio = auditok.AudioRegion(rec.data, sr, sw, ch)
```

`Recorder` also accepts a *max_read* argument.

CHAPTER 7

Working with AudioRegions

The following are a couple of interesting operations you can do with `AudioRegion` objects.

7.1 Basic region information

```
import auditok
region = auditok.load("audio.wav")
len(region) # number of audio samples int the regions, one channel considered
region.duration # duration in seconds
region.sampling_rate # alias `sr`
region.sample_width # alias `sw`
region.channels # alias `ch`
```

7.2 Concatenate regions

```
import auditok
region_1 = auditok.load("audio_1.wav")
region_2 = auditok.load("audio_2.wav")
region_3 = region_1 + region_2
```

Particularly useful if you want to join regions returned by `split()`:

```
import auditok
regions = auditok.load("audio.wav").split()
gapless_region = sum(regions)
```

7.3 Repeat a region

Multiply by a positive integer:

```
import auditok
region = auditok.load("audio.wav")
region_x3 = region * 3
```

7.4 Split one region into N regions of equal size

Divide by a positive integer (this has nothing to do with silence-based tokenization):

```
import auditok
region = auditok.load("audio.wav")
regions = regions / 5
assert sum(regions) == region
```

Note that if no perfect division is possible, the last region might be a bit shorter than the previous N-1 regions.

7.5 Slice a region by samples, seconds or milliseconds

Slicing an `AudioRegion` can be interesting in many situations. You can for example remove a fixed-size portion of audio data from the beginning or from the end of a region or crop a region by an arbitrary amount as a data augmentation strategy.

The most accurate way to slice an *AudioRegion* is to use indices that directly refer to raw audio samples. In the following example, assuming that the sampling rate of audio data is 16000, you can extract a 5-second region from main region, starting from the 20th second as follows:

```
import auditok
region = auditok.load("audio.wav")
start = 20 * 16000
stop = 25 * 16000
five_second_region = region[start:stop]
```

This allows you to practically start and stop at any audio sample within the region. Just as with a *list* you can omit one of *start* and *stop*, or both. You can also use negative indices:

```
import auditok
region = auditok.load("audio.wav")
start = -3 * region.sr # `sr` is an alias of `sampling_rate`
three_last_seconds = region[start:]
```

While slicing by raw samples is flexible, slicing with temporal indices is more intuitive. You can do so by accessing the `millis` or `seconds` views of an *AudioRegion* (or their shortcut alias *ms* and *sec* or *s*).

With the `millis` view:

```
import auditok
region = auditok.load("audio.wav")
five_second_region = region.millis[5000:10000]
```

or with the `seconds` view:

```
import auditok
region = auditok.load("audio.wav")
five_second_region = region.seconds[5:10]
```

`seconds` indices can also be floats:

```
import auditok
region = auditok.load("audio.wav")
five_second_region = region.seconds[2.5:7.5]
```

7.6 Get arrays of audio samples

If *numpy* is not installed, the *samples* attributes is a list of audio samples arrays (standard *array.array* objects), one per channels. If numpy is installed, *samples* is a 2-D *numpy.ndarray* where the fist dimension is the channel and the second is the the sample.

```
import auditok
region = auditok.load("audio.wav")
samples = region.samples
assert len(samples) == region.channels
```

If *numpy* is not installed you can use:

```
import numpy as np
region = auditok.load("audio.wav")
samples = np.asarray(region)
assert len(samples.shape) == 2
```

`auditok` can also be used from the command-line. For more information about parameters and their description type:

```
auditok -h
```

In the following we'll a few examples that covers most use-cases.

CHAPTER 8

Read and split audio data online

To try `auditok` from the command line with you voice, you should either install pyaudio so that `auditok` can directly read data from the microphone, or record data with an external program (e.g., *sox*) and redirect its output to `auditok`.

Read data from the microphone (*pyaudio* installed):

```
auditok
```

This will print the *id*, *start time* and *end time* of each detected audio event. Note that we didn't pass any additional arguments to the previous command, so `auditok` will use default values. The most important arguments are:

- `-n`, `--min-duration` : minimum duration of a valid audio event in seconds, default: 0.2
- `-m`, `--max-duration` : maximum duration of a valid audio event in seconds, default: 5
- `-s`, `--max-silence` : maximum duration of a consecutive silence within a valid audio event in seconds, default: 0.3
- `-e`, `--energy-threshold` : energy threshold for detection, default: 50

CHAPTER 9

Read audio data with an external program

If you don't have *pyaudio*, you can use *sox* for data acquisition (*sudo apt-get install sox*) and make `auditok` read data from standard input:

```
rec -q -t raw -r 16000 -c 1 -b 16 -e signed - | auditok - -r 16000 -w 2 -c 1
```

Note that when data is read from standard input, the same audio parameters must be used for both *sox* (or any other data generation/acquisition tool) and `auditok`. The following table summarizes audio parameters.

Audio parameter	sox option	*auditok* option	*auditok* default
Sampling rate	-r	-r	16000
Sample width	-b (bits)	-w (bytes)	2
Channels	-c	-c	1
Encoding	-e	NA	always a signed int

According to this table, the previous command can be run with the default parameters as:

```
rec -q -t raw -r 16000 -c 1 -b 16 -e signed - | auditok -i -
```

CHAPTER 10

Play back audio detections

Use the `-E` option (for echo):

```
auditok -E
# or
rec -q -t raw -r 16000 -c 1 -b 16 -e signed - | auditok - -E
```

The second command works without further argument because data is recorded with `auditok`'s default audio parameters . If one of the parameters is not at the default value you should specify it alongside `-E`.

Using `-E` requires *pyaudio*, if it's not installed you can use the `-C` (used to run an external command with detected audio event as argument):

```
rec -q -t raw -r 16000 -c 1 -b 16 -e signed - | auditok - -C "play -q {file}"
```

Using the `-C` option, `auditok` will save a detected event to a temporary wav file, fill the `{file}` placeholder with the temporary name and run the command. In the above example we used `-C` to play audio data with an external program but you can use it to run any other command.

CHAPTER 11

Print out detection information

By default `auditok` prints out the **id**, the **start** and the **end** of each detected audio event. The latter two values represent the absolute position of the event within input stream (file or microphone) in seconds. The following listing is an example output with the default format:

```
1 1.160 2.390
2 3.420 4.330
3 5.010 5.720
4 7.230 7.800
```

The format of the output is controlled by the `--printf` option. Alongside `{id}`, `{start}` and `{end}` placeholders, you can use `{duration}` and `{timestamp}` (system timestamp of detected event) placeholders.

Using the following format for example:

```
auditok audio.wav  --printf "{id}: [{timestamp}] start:{start}, end:{end}, dur:
↪{duration}"
```

the output would be something like:

```
1: [2021/02/17 20:16:02] start:1.160, end:2.390, dur: 1.230
2: [2021/02/17 20:16:04] start:3.420, end:4.330, dur: 0.910
3: [2021/02/17 20:16:06] start:5.010, end:5.720, dur: 0.710
4: [2021/02/17 20:16:08] start:7.230, end:7.800, dur: 0.570
```

The format of `{timestamp}` is controlled by `--timestamp-format` (default: *"%Y/%m/%d %H:%M:%S"*) whereas that of `{start}`, `{end}` and `{duration}` by `--time-format` (default: *%S*, absolute number of seconds). A more detailed format with `--time-format` using *%h* (hours), *%m* (minutes), *%s* (seconds) and *%i* (milliseconds) directives is possible (e.g., "%h:%m:%s.%i).

To completely disable printing detection information use `-q`.

CHAPTER 12

Save detections

You can save audio events to disk as they're detected using `-o` or `--save-detections-as`. To get a uniq file name for each event, you can use `{id}`, `{start}`, `{end}` and `{duration}` placeholders. Example:

```
auditok --save-detections-as "{id}_{start}_{end}.wav"
```

When using `{start}`, `{end}` and `{duration}` placeholders, it's recommended that the number of decimals of the corresponding values be limited to 3. You can use something like:

```
auditok -o "{id}_{start:.3f}_{end:.3f}.wav"
```

CHAPTER 13

Save whole audio stream

When reading audio data from the microphone, you most certainly want to save it to disk. For this you can use the `-O` or `--save-stream` option.

```
auditok --save-stream "stream.wav"
```

Note this will work even if you read data from another file on disk.

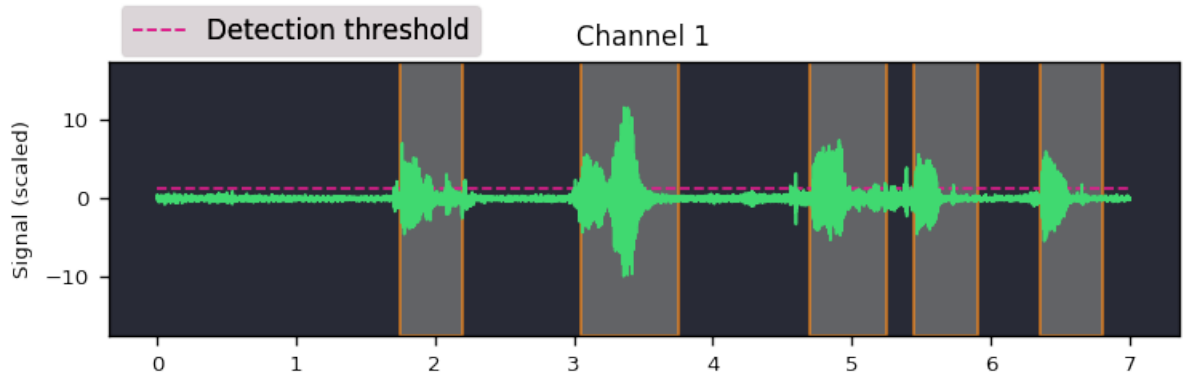
CHAPTER 14

Plot detections

Audio signal and detections can be plotted using the `-p` or `--plot` option. You can also save plot to disk using `--save-image`. The following example does both:

```
auditok -p --save-image "plot.png" # can also be 'pdf' or another image format
```

output example:



Plotting requires matplotlib.

CHAPTER 15

Core

CHAPTER 16

Util

CHAPTER 17

Low-level IO

CHAPTER 18

Signal processing

CHAPTER 19

Dataset

CHAPTER 20

License

MIT.