Sounddevice

Soundfile: <https://pysoundfile.readthedocs.io/en/0.9.0/#soundfile.read>

Soundfile.read(file): Provide audio data from a sound file as NumPy array.

Returns:

**audiodata** (numpy.ndarray or type(out)) – A two-dimensional NumPy array is returned, where the channels are stored along the first dimension, i.e. as columns. If the sound file has only one channel, a one-dimensional array is returned.

**samplerate** (int) – The sample rate of the audio file.

Sounddevice

Play: <https://python-sounddevice.readthedocs.io/en/0.3.13/examples.html#plot-microphone-signal-s-in-real-time>

Simultaneous Playback and Recording

<https://python-sounddevice.readthedocs.io/en/0.3.13/usage.html#simultaneous-playback-and-recording>

myrecording = sd.playrec(myarray, fs, channels=2)

Realtime Audio Visualization In Python

<https://www.swharden.com/wp/2016-07-19-realtime-audio-visualization-in-python/>

<http://zwmiller.com/projects/streamAudio.html>

Continuesly streaming audio signal real time infinitely, Python

<https://stackoverflow.com/questions/48653745/continuesly-streaming-audio-signal-real-time-infinitely-python>

Pyaudio

Record sound

<https://dsp.stackexchange.com/questions/13728/what-are-chunks-when-recording-a-voice-signal>

chunks:

chunk = 1024

The chunk is like a buffer, so therefore each buffer will contain 1024 samples, which you can then either keep or throw away. We use CHUNKS of data, instead of a continuous amount of audio because of processing power. (Let's assume this was a continuous flow of data being received from a Microphone and is being recorded and saved) then it would just eat up the processor, thus causing potential crashes. In terms of Raspberry Pi / Arduino development (Where the RAM is very small) CHUNKING the data like this makes the stream flow easier and thus prevents memory leaks.

Another reason:

Let's assume that you wanted to implement an algorithm for determining whether something is speech, or, is just noise. How would/could you do this using a constant stream flow of sound data? It would be very difficult. Therefore, by storing this into an array (or list in your case) you can perform analysis on this data, for example RMS. Then you could have some threshold to determine if you want to keep the data, or the data is no good

One method to simulate Realtime data stream from CSV file:

use pandas read\_csv() function to read the big csv file in small chunks

<http://pandas.pydata.org/pandas-docs/stable/user_guide/io.html#io-chunking>

import pandas as pd

chunksize = 100

for chunk in pd.read\_csv(‘myfile.csv’, chunksize=chunksize):

print(chunk)

<http://zwmiller.com/projects/streamAudio.html>

Plot Streaming Audio with Python

Plot\_data(in\_data)

In\_data = stream.read(CHUNK)

More audio <https://github.com/khalil8500/ReSpeaker_Diarization>

<https://github.com/Spawns/Speaker_Diarization>

**pyBK**

1. Feature extraction 🡪 allData
2. UEM and SAD 🡪maskUEM and maskSAD
3. getSegmentTable 🡪 segmentTable
4. training the KBM 🡪 kbm, gmPool = trainKBM(data,config.getint('KBM','windowLength'),windowRate,kbmSize )
5. get Vg Matrix

Vg = getVgMatrix(data,gmPool,kbm,config.getint('BINARY\_KEY','topGaussiansPerFrame'))

1. compute binary key: segmentBKTable, segmentCVTable = getSegmentBKs(segmentTable, kbmSize, Vg, config.getfloat('BINARY\_KEY','bitsPerSegmentFactor'), speechMapping)
2. perform clustering based on the binary key
   1. initial clustering (based on number of segments etc)
   2. Perform agglomerative clustering (linkage / others)
   3. Select best clustering (elbow / spectral)
3. Perform resegmentation
4. Feature Extraction

Default: Features (Mel-Frequency Cepstral Coefficients (MFCCs)) are extracted using the librosa library (<https://librosa.github.io/librosa/>)

framelength = 0.025 # Window length for feature extraction in ms

frameshift = 0.01 # Window shift for feature extraction in ms

nfilters = 30 # Number of mel filters used

ncoeff = 30 # Number of MFCCs employed

Conduct the feature extraction on the input audio signal to obtain MFCC features.

y, sr = librosa.load(audioFile,sr=None) # by setting sr = None, it will use native sample rate)

y: audio time series, array with length = sr

sr: sample rate

frame\_length\_inSample=framelength\*sr

hop = int(frameshift\*sr)

NFFT=int(2\*\*np.ceil(np.log2(frame\_length\_inSample)))

features=librosa.feature.mfcc(y=y,

sr=sr,dct\_type=2,n\_mfcc=ncoeff,n\_mels=nfilters,n\_fft=NFFT,hop\_length=hop).T

n\_fft: length of the fft window

hop\_length: number of samples between successive frames

n\_mfcc: number of MFCCs to return

n\_mels: ???

features: np.ndarray shape = (n\_mfcc, t)

1. SAD

SAD:

Data, sr = load(file, sr = none)

# above steps are in SAD & feature extraction 🡪 can combine

现有的方法是 getSADfile 🡪 把SAD的结果(每一个segments的start time, end time) 写进 一个SAD file, 然后再用readSADfile 读maskSAD

但也可以： 直接得到maskSAD

File 🡪 data 🡪

va\_framed = py\_webrtcvad(data, fs=sr, fs\_vad=sr, hoplength=30, vad\_mode=0)

segments = get\_py\_webrtcvad\_segments(va\_framed, sr)

UEM file

SAD file: See if existing SAD file, otherwise get SAD file

2.1 webrtcvad from <https://github.com/wiseman/py-webrtcvad.git>

VAD: Voice Activity Detection

Input data ndarray (if int, -32768 < data < 32767)

Fs = fs\_vad = sr (sampling rate)

Hop\_length = 30: step size (milli second)

Vad\_mode = 0 (higher number, more aggressive

0 is the least aggressive about filtering out non-speech, 3 is the most aggressive.

py-webrtcvad

This is a python interface to the WebRTC Voice Activity Detector (VAD). It is compatible with Python 2 and Python 3.

A [VAD](https://en.wikipedia.org/wiki/Voice_activity_detection) classifies a piece of audio data as being voiced or unvoiced. It can be useful for telephony and speech recognition.

The VAD that Google developed for the [WebRTC](https://webrtc.org/) project is reportedly one of the best available, being fast, modern and free.

The VAD is unsupervised, use GMM (Gaussian Mixture Model) to model the voice and noise, determine the voice/noise based on the corresponding probability

VAD research

<https://www.microsoft.com/en-us/research/uploads/prod/2018/02/KoPhiliposeTashevZarar_ICASSP_2018.pdf>

this paper: new method to give better result and less delay

“Conventional VAD algorithms are generally based on statistical signal processing that make strong assumptions on the distributions of speech and background noise.”

Recently, another VAD algorithm based on the Gaussian mixture model was developed in line with the WebRTC project, including an open-source implementation that targets real-time performance [8]. This algorithm has found wide adoption and has recently become one of the gold-standards for delay-sensitive scenarios like web-based interaction.

[8] “WebRTC,” 2017. [Online]. Available: <https://webrtc.org/>.

Google WebRTC VAD which uses multiple frequency band features with pre-trained GMM classifier

\_\_\_

<https://arxiv.org/pdf/1712.01340.pdf>

Precision Scaling of Neural Networks for Efficient Audio Processing

While deep neural networks have shown powerful performance in many audio applications, their large computation and memory demand has been a challenge for real-time processing

With the rapid development of deep-learning technologies,

VAD and speech enhancement approaches based on deep neural networks (DNNs) have shown

powerful performance highly competitive to conventional methods [2, 3, 4]. However, DNNs are

inherently complex with high computation and memory demand [5], which is a critical challenge

in real-time speech applications

[5] ] J. H. Ko, D. Kim, T. Na, J. Kung, and S. Mukhopadhyay. Adaptive weight compression for memory-efficient neural networks.

3．getSegmentTable

在这一步之前：

maskUEM = np.ones([1,nFeatures])

mask = np.logical\_and(maskUEM,maskSAD)

mask = mask[0][0:nFeatures]

nSpeechFeatures=np.sum(mask)

speechMapping = np.zeros(nFeatures)

speechMapping[np.nonzero(mask)] = np.arange(1,nSpeechFeatures+1)

data=allData[np.where(mask==1)] (allData 从get features得来)

//不需要 input data

getSegmentTable(mask, speechMapping, wLength, wIncr, wShift):

// wLength = 100 (100 frames for one window)

// wIncr = 100 (Window increment after and before window in frames)

// wShift = 100 (Window shifting in frames)

changePoints,segBeg,segEnd,nSegs = unravelMask(mask)

Fast Speaker Diarization Using Python

<http://multimedia.icsi.berkeley.edu/speaker-diarization/fast-speaker-diarization-using-python/>

GPU