Surfboard

Jul 17, 2020

Contents

You can find our paper on arXiv.

CHAPTER 1

High level overview

1.1 Introduction

Surfboard is a package for audio-feature extraction written in Python. Information and tutorials can be found in our README on GitHub. Please see our paper for more details.

1.2 Installing Surfboard

Installing Surfboard is as easy as boogie boarding! You can install it from PyPi:

```
pip install surfboard
```

Alternatively, you can clone it and install it as such:

```
git clone https://github.com/novoic/surfboard
cd surfboard
pip install .
```

The package builds on LibROSA. You might need to install Libsndfile. On Linux:

```
sudo apt-get install libsndfile1-dev
```

On MacOS:

```
brew install libsndfile
```

CHAPTER 2

Core Surfboard classes: Waveform and Barrel

At the heart of Surfboard lie two classes: the `Waveform` class and the `Barrel` class.

2.1 Core Surfboard classes: Waveform and Barrel

2.1.1 Waveform class

This file contains the central Waveform class of the surfboard package, and all the corresponding methods

class surfboard.sound.**Waveform**(*path=None*, *signal=None*, *sample_rate=44100*)

The central class of the package. This class instantiates with a path to a sound file and a sample rate to load it or a signal and a sample rate. We can then use methods of this class to compute various components.

waveform

Properties written in this way prevent users to assign to self.waveform

sample_rate

Properties written in this way prevent users to assign to self.sample_rate

compute_components(*component_list*)

Compute components from self.waveform and self.sample_rate using a list of strings which identify which components to compute. You can pass in arguments to the components (e.g. frame_length_seconds) by passing in the components as dictionaries. For example: {'mfcc': {'n_mfcc': 26}}. See README.md for more details.

Parameters **component_list** (*list of str or dict*) – The methods to be computed. If elements are str, then the method uses default arguments. If dict, the arguments are passed to the methods.

Returns Dictionary mapping component names to computed components.

Return type dict

mfcc(*n_mfcc=13*, *n_fft_seconds=0.04*, *hop_length_seconds=0.01*)
Given a number of MFCCs, use the librosa.feature.mfcc method to compute the correct number of MFCCs on self.waveform and returns the array.

Parameters

- **n_mfcc** (*int*) – number of MFCCs to compute
- **n_fft_seconds** (*float*) – length of the FFT window in seconds.
- **hop_length_seconds** (*float*) – how much the window shifts for every timestep, in seconds.

Returns MFCCs.

Return type np.array, [n_mfcc, T / hop_length]

log_melspec(*n_mels=128*, *n_fft_seconds=0.04*, *hop_length_seconds=0.01*)
Given a number of filter banks, this uses the librosa.feature.melspectrogram method to compute the log melspectrogram of self.waveform.

Parameters

- **n_mels** (*int*) – Number of filter banks per time step in the log melspectrogram.
- **n_fft_seconds** (*float*) – Length of the FFT window in seconds.
- **hop_length_seconds** (*float*) – How much the window shifts for every timestep, in seconds.

Returns Log mel spectrogram.

Return type np.array, [n_mels, T_mels]

magnitude_spectrum(*n_fft_seconds=0.04*, *hop_length_seconds=0.01*)
Compute the STFT of self.waveform. This is used for further spectral analysis.

Parameters

- **n_fft_seconds** (*float*) – Length of the FFT window in seconds.
- **hop_length_seconds** (*float*) – How much the window shifts for every timestep, in seconds.

Returns The magnitude spectrogram

Return type np.array, [n_fft / 2 + 1, T / hop_length]

bark_spectrogram(*n_fft_seconds=0.04*, *hop_length_seconds=0.01*)
Compute the magnitude spectrum of self.waveform and arrange the frequency bins in the Bark scale. See https://en.wikipedia.org/wiki/Bark_scale

Parameters

- **n_fft_seconds** (*float*) – Length of the FFT window in seconds.
- **hop_length_seconds** (*float*) – How much the window shifts for every timestep, in seconds.

Returns The Bark spectrogram

Return type np.array, [n_bark_bands, T / hop_length]

morlet_cwt(*widths=None*)
Compute the Morlet Continuous Wavelet Transform of self.waveform. Note that this method returns a large matrix. Shown relevant in Vasquez-Correa et Al, 2016.

Parameters

- **wavelet** (*str*) – Wavelet to use. Currently only support "morlet".
- **widhts** (*None or list*) – If None, uses default of 32 evenly spaced widths as [i * sample_rate / 500 for i in range(1, 33)]

Returns The continuous wavelet transform

Return type np.array, [len(widths), T]

chroma_stft(*n_fft_seconds=0.04*, *hop_length_seconds=0.01*, *n_chroma=12*)

See librosa.feature documentation for more details on this component. This computes a chromagram from a waveform.

Parameters

- **n_fft_seconds** (*float*) – Length of the FFT window in seconds.
- **hop_length_seconds** (*float*) – How much the window shifts for every timestep, in seconds.
- **n_chroma** (*int*) – Number of chroma bins to compute.

Returns The chromagram

Return type np.array, [n_chroma, T / hop_length]

chroma_cqt(*hop_length_seconds=0.01*, *n_chroma=12*)

See librosa.feature documentation for more details on this component. This computes a constant-Q chromagram from a waveform.

Parameters

- **hop_length_seconds** (*float*) – How much the window shifts for every timestep, in seconds.
- **n_chroma** (*int*) – Number of chroma bins to compute.

Returns Constant-Q transform mode

Return type np.array, [n_chroma, T / hop_length]

chroma_cens(*hop_length_seconds=0.01*, *n_chroma=12*)

See librosa.feature documentation for more details on this component. This computes the CENS chroma variant from a waveform.

Parameters

- **hop_length_seconds** (*float*) – How much the window shifts for every timestep, in seconds.
- **n_chroma** (*int*) – Number of chroma bins to compute.

Returns CENS-chromagram

Return type np.array, [n_chroma, T / hop_length]

spectral_slope(*n_fft_seconds=0.04*, *hop_length_seconds=0.01*)

Compute the magnitude spectrum, and compute the spectral slope from that. This is a basic approximation of the spectrum by a linear regression line. There is one coefficient per timestep.

Parameters

- **n_fft_seconds** (*float*) – Length of the FFT window in seconds.

- **hop_length_seconds** (*float*) – How much the window shifts for every timestep, in seconds.

Returns Linear regression slope, for every timestep.

Return type np.array, [1, T / hop_length]

spectral_flux(*n_fft_seconds=0.04*, *hop_length_seconds=0.01*)
Compute the magnitude spectrum, and compute the spectral flux from that. This is a basic metric, measuring the rate of change of the spectrum.

Parameters

- **n_fft_seconds** (*float*) – Length of the FFT window in seconds.
- **hop_length_seconds** (*float*) – How much the window shifts for every timestep, in seconds.

Returns The spectral flux array.

Return type np.array, [1, T / hop_length]

spectral_entropy(*n_fft_seconds=0.04*, *hop_length_seconds=0.01*)
Compute the magnitude spectrum, and compute the spectral entropy from that. To compute that, simply normalize each frame of the spectrum, so that they are a probability distribution, then compute the entropy from that.

Parameters

- **n_fft_seconds** (*float*) – Length of the FFT window in seconds.
- **hop_length_seconds** (*float*) – How much the window shifts for every timestep, in seconds.

Returns The entropy of each normalized frame.

Return type np.array, [1, T / hop_length]

spectral_centroid(*n_fft_seconds=0.04*, *hop_length_seconds=0.01*)
Compute spectral centroid from magnitude spectrum. “First moment”.

Parameters

- **n_fft_seconds** (*float*) – Length of the FFT window in seconds.
- **hop_length_seconds** (*float*) – How much the window shifts for every timestep, in seconds.

Returns Spectral centroid of the magnitude spectrum (first moment).

Return type np.array, [1, T / hop_length]

spectral_spread(*n_fft_seconds=0.04*, *hop_length_seconds=0.01*)
Compute spectral spread (also spectral variance) from magnitude spectrum.

Parameters

- **n_fft_seconds** (*float*) – Length of the FFT window in seconds.
- **hop_length_seconds** (*float*) – How much the window shifts for every timestep, in seconds.

Returns Spectral skewness of the magnitude spectrum (second moment).

Return type np.array, [1, T / hop_length

spectral_skewness(*n_fft_seconds=0.04*, *hop_length_seconds=0.01*)
Compute spectral skewness from magnitude spectrum.

Parameters

- **n_fft_seconds** (*float*) – Length of the FFT window in seconds.
- **hop_length_seconds** (*float*) – How much the window shifts for every timestep, in seconds.

Returns Spectral skewness of the magnitude spectrum (third moment).

Return type np.array, [1, T / hop_length

spectral_kurtosis(*n_fft_seconds=0.04*, *hop_length_seconds=0.01*)
Compute spectral kurtosis from magnitude spectrum.

Parameters

- **n_fft_seconds** (*float*) – Length of the FFT window in seconds.
- **hop_length_seconds** (*float*) – How much the window shifts for every timestep, in seconds.

Returns Spectral kurtosis of the magnitude spectrum (fourth moment).

Return type np.array, [1, T / hop_length]

spectral_flatness(*n_fft_seconds=0.04*, *hop_length_seconds=0.01*)
Given an FFT window size and a hop length, uses the librosa feature package to compute the spectral flatness of self.waveform. This component is a measure to quantify how "noise-like" a sound is. The closer to 1, the closer the sound is to white noise.

Parameters

- **n_fft_seconds** (*float*) – Length of the FFT window in seconds.
- **hop_length_seconds** (*float*) – How much the window shifts for every timestep, in seconds.

Returns Spectral flatness vector computed over windows.

Return type np.array, [1, T/hop_length]

spectral_rolloff(*roll_percent=0.85*, *n_fft_seconds=0.04*, *hop_length_seconds=0.01*)
Given an FFT window size and a hop length, uses the librosa component package to compute the spectral roll-off of self.waveform. It is the point below which most energy of a signal is contained and is useful in distinguishing sounds with different energy distributions.

Parameters

- **roll_percent** (*float*) – The roll-off percentage: https://essentia.upf.edu/reference/streaming_RollOff.html
- **n_fft_seconds** (*float*) – Length of the FFT window in seconds.
- **hop_length_seconds** (*float*) – How much the window shifts for every timestep, in seconds.

Returns Spectral rolloff vector computed over windows.

Return type np.array, [1, T/hop_length]

loudness()
Compute the loudness of self.waveform using the pyloudnorm package. See https://github.com/csteinmetz1/pyloudnorm for more details on potential arguments to the functions below.

Returns The loudness of self.waveform

Return type float

loudness_slidingwindow(*frame_length_seconds=1*, *hop_length_seconds=0.25*)
Compute the loudness of self.waveform over time. See self.loudness for more details.

Parameters

- **frame_length_seconds** (*float*) – Length of the sliding window in seconds.
- **hop_length_seconds** (*float*) – How much the sliding window moves by

Returns The loudness on frames of self.waveform

Return type [1, T / hop_length]

shannon_entropy()
Compute the Shannon entropy of self.waveform, as per https://ijssst.info/Vol-16/No-4/data/8258a127.pdf

Returns Shannon entropy of the waveform.

Return type float

shannon_entropy_slidingwindow(*frame_length_seconds=0.04*, *hop_length_seconds=0.01*)
Compute the Shannon entropy of subblocks of a waveform into a newly created time series, as per https://ijssst.info/Vol-16/No-4/data/8258a127.pdf

Parameters

- **frame_length_seconds** (*float*) – Length of the sliding window, in seconds.
- **hop_length_seconds** (*float*) – How much the window shifts for every timestep, in seconds.

Returns Shannon entropy for each frame

Return type np.array, [1, T / hop_length]

zerocrossing()
Compute the zero crossing rate on self.waveform and return it as per https://journals.plos.org/plosone/article/file?id=10.1371/journal.pone.0162128&type=printable Note: can also compute zero crossing rate as a time series – see librosa.feature.zero_crossing_rate, and self.get_zcr_sequence.

Returns Keys "num_zerocrossings" and "rate" mapping to: zerocrossing["num_zerocrossings"]: number of zero crossings in self.waveform zerocrossing["rate"]: number of zero crossings divided by number of samples.

Return type dictionary

zerocrossing_slidingwindow(*frame_length_seconds=0.04*, *hop_length_seconds=0.01*)
Compute the zero crossing rate sequence on self.waveform and return it. This is now a sequence where every entry is computed on frame_length samples. There is a sliding window of length hop_length.

Parameters

- **frame_length_seconds** (*float*) – Length of the sliding window, in seconds.
- **hop_length_seconds** (*float*) – How much the window shifts for every timestep, in seconds.

Returns Fraction of zero crossings for each frame.

Return type np.array, [1, T / hop_length]

rms (*frame_length_seconds=0.04*, *hop_length_seconds=0.01*)
Get the root mean square value for each frame, with a specific frame length and hop length. This used to be called RMSE, or root mean square energy in the jargon?

Parameters

- **frame_length_seconds** (*float*) – Length of the sliding window, in seconds.
- **hop_length_seconds** (*float*) – How much the window shifts for every timestep, in seconds.

Returns RMS value for each frame.

Return type np.array, [1, T / hop_length]

intensity (*frame_length_seconds=0.04*, *hop_length_seconds=0.01*)
Get a value proportional to the intensity for each frame, with a specific frame length and hop length. Note that the intensity is proportional to the RMS amplitude squared.

Parameters

- **frame_length_seconds** (*float*) – Length of the sliding window, in seconds.
- **hop_length_seconds** (*float*) – How much the window shifts for every timestep, in seconds.

Returns Proportional intensity value for each frame.

Return type np.array, [1, T / hop_length]

crest_factor (*frame_length_seconds=0.04*, *hop_length_seconds=0.01*)
Get the crest factor of this waveform, on sliding windows. This value measures the local intensity of peaks in a waveform. Implemented as per: https://en.wikipedia.org/wiki/Crest_factor

Parameters

- **frame_length_seconds** (*float*) – Length of the sliding window, in seconds.
- **hop_length_seconds** (*float*) – How much the window shifts for every timestep, in seconds.

Returns Crest factor for each frame.

Return type np.array, [1, T / hop_length]

f0_contour (*hop_length_seconds=0.01*, *method='swipe'*, *f0_min=60*, *f0_max=300*)
Compute the F0 contour using PYSPTK: https://github.com/r9y9/pysptk/.

Parameters

- **hop_length_seconds** (*float*) – Hop size argument in pysptk. Corresponds to hop-size in the window sliding of the computation of f0. This is in seconds and gets converted.
- **method** (*str*) – One of 'swipe' or 'rapt'. Define which method to use for f0 calculation. See https://github.com/r9y9/pysptk
- **f0_min** (*float*) – minimum acceptable f0.
- **f0_max** (*float*) – maximum acceptable f0.

Returns

F0 contour of self.waveform. Contains unvoiced frames.

Return type np.array, [1, t1]

f0_statistics(*hop_length_seconds=0.01*, *method='swipe'*)
Compute the F0 mean and standard deviation of self.waveform. Note that we cannot simply rely on using statistics applied to the f0_contour since we do not want to include the zeros in the mean and standard deviation calculations.

Parameters

- **hop_length_seconds** (*float*) – Hop size argument in pysptk. Corresponds to hop-size in the window sliding of the computation of f0. This is in seconds and gets converted.
- **method** (*str*) – One of 'swipe' or 'rapt'. Define which method to use for f0 calculation. See https://github.com/r9y9/pysptk

Returns

Dictionary mapping: "mean": f0 mean of self.waveform. "std": f0 standard deviation of self.waveform.

Return type dict

ppe()
Compute pitch period entropy. This is an adaptation of the following Matlab code: https://github.com/Mak-Sim/Troparion/blob/5126f434b96e0c1a4a41fa99dd9148f3c959cfac/Perturbation_analysis/pitch_period_entropy.m Note that computing the PPE relies on the existence of voiced portions in the F0 trajectory.

Returns The pitch period entropy, as per http://www.maxlittle.net/students/thesis_tsanas.pdf

Return type float

jitters(*p_floor=0.0001*, *p_ceil=0.02*, *max_p_factor=1.3*)
Compute the jitters mathematically, according to certain conditions given by p_floor, p_ceil and max_p_factor. See jitters.py for more details.

Parameters

- **p_floor** (*float*) – Minimum acceptable period.
- **p_ceil** (*float*) – Maximum acceptable period.
- **max_p_factor** (*float*) – value to use for the period factor principle

Returns dictionary mapping strings to floats, with keys "localJitter", "localabsoluteJitter", "rapJitter", "ppq5Jitter", "ddpJitter"

Return type dict

shimmers(*max_a_factor=1.6*, *p_floor=0.0001*, *p_ceil=0.02*, *max_p_factor=1.3*)
Compute the shimmers mathematically, according to certain conditions given by max_a_factor, p_floor, p_ceil and max_p_factor. See shimmers.py for more details.

Parameters

- **max_a_factor** (*float*) – Value to use for amplitude factor principle
- **p_floor** (*float*) – Minimum acceptable period.
- **p_ceil** (*float*) – Maximum acceptable period.
- **max_p_factor** (*float*) – value to use for the period factor principle

Returns

Dictionary mapping strings to floats, with keys "localShimmer", "localdbShimmer", "apq3Shimmer", "apq5Shimmer", "apq11Shimmer"

Return type dict

hnr()

See https://www.ncbi.nlm.nih.gov/pubmed/12512635 for more thorough description of why HNR is important in the scope of healthcare.

Returns The harmonics to noise ratio computed on self.waveform.

Return type float

dfa(*window_lengths=[64, 128, 256, 512, 1024, 2048, 4096]*)

See Tsanas et al, 2011: Novel speech signal processing algorithms for high-accuracy classification of Parkinsons disease Detrended Fluctuation Analysis

Parameters **window_lengths** (*list of int > 0*) – List of L to use in DFA computation. See dfa.py for more details.

Returns The detrended fluctuation analysis alpha value.

Return type float

lpc(*order=4*, *return_np_array=False*)

This uses the librosa backend to get the Linear Prediction Coefficients via Burg's method. See librosa.core.lpc for more details.

Parameters

- **order** (*int > 0*) – Order of the linear filter
- **return_np_array** (*bool*) – If False, returns a dictionary. Otherwise a numpy array.

Returns Dictionary mapping 'LPC_{i}' to the i'th lpc coefficient, for i = 0...order. Or: LP prediction error coefficients (np array case)

Return type dict or np.array, [order + 1,]

lsf(*order=4*, *return_np_array=False*)

Compute the LPC coefficients, then convert them to LSP frequencies. The conversion is done using https://github.com/cokelaer/spectrum/blob/master/src/spectrum/linear_prediction.py

Parameters

- **order** (*int > 0*) – Order of the linear filter for LPC calculation
- **return_np_array** (*bool*) – If False, returns a dictionary. Otherwise a numpy array.

Returns

Dictionary mapping 'LPC_{i}' to the i'th lpc coefficient, for i = 0...order. Or LSP frequencies (np array case).

Return type dict or np.array, [order,]

formants()

Estimate the first four formant frequencies using LPC (see formants.py)

Returns Dictionary mapping {'f1', 'f2', 'f3', 'f4'} to corresponding {first, second, third, fourth} formant frequency.

Return type dict

formants_slidingwindow(*frame_length_seconds=0.04*, *hop_length_seconds=0.01*)

Estimate the first four formant frequencies using LPC (see formants.py) and apply the metric_slidingwindow decorator.

Parameters

- **frame_length_seconds** (*float*) – Length of the sliding window, in seconds.
- **hop_length_seconds** (*float*) – How much the window shifts for every timestep, in seconds.

Returns

Time series of the first four formant frequencies computed on windows of length frame_length_seconds, with sliding window of hop_length_seconds.

Return type np.array, [4, T / hop_length]

kurtosis_slidingwindow(*frame_length_seconds=0.04*, *hop_length_seconds=0.01*)

Computes the kurtosis on frames of the waveform with a sliding window

Parameters

- **frame_length_seconds** (*float*) – Length of the sliding window, in seconds.
- **hop_length_seconds** (*float*) – How much the window shifts for every timestep, in seconds.

Returns Kurtosis on each sliding window.

Return type np.array, [1, T / hop_length]

log_energy()

Compute the log energy of self.waveform as per Abeyrante et al. 2013.

Returns The log energy of self.waveform, computed as per the paper above.

Return type float

log_energy_slidingwindow(*frame_length_seconds=0.04*, *hop_length_seconds=0.01*)

Computes the log energy on frames of the waveform with a sliding window

Parameters

- **frame_length_seconds** (*float*) – Length of the sliding window, in seconds.
- **hop_length_seconds** (*float*) – How much the window shifts for every timestep, in seconds.

Returns Log energy on each sliding window.

Return type np.array, [1, T / hop_length]

2.1.2 Barrel class

This file contains the class which computes statistics from numpy arrays to turn components into features.

class surfboard.statistics.**Barrel**(*component*)

This class is used to instantiate components computed in the surfboard package. It helps us compute statistics on these components.

compute_statistics(*statistic_list*)

Compute statistics on self.component using a list of strings which identify which statistics to compute.

Parameters **statistic_list** (*list of str*) – list of strings representing Barrel methods to be called.

Returns Dictionary mapping str to float.

Return type dict

get_first_derivative()
Compute the "first derivative" of self.component. Remember that self.component is of the shape [n_feats, T].

Returns First empirical derivative.

Return type np.array, [n_feats, T - 1]

get_second_derivative()
Compute the "second derivative" of self.component. Remember that self.component is of the shape [n_feats, T].

Returns second empirical derivative.

Return type np.array, [n_feats, T - 2]

max()
Compute the max of self.component on the last dimensions.

Returns

The maximum of each individual dimension in self.component

Return type np.array, [n_feats,]

min()
Compute the min of self.component on the last dimension.

Returns

The minimum of each individual dimension in self.component

Return type np.array, [n_feats,]

mean()
Compute the mean of self.component on the last dimension (time).

Returns

The mean of each individual dimension in self.component

Return type np.array, [n_feats,]

first_derivative_mean()
Compute the mean of the first empirical derivative (delta coefficient) on the last dimension (time).

Returns

The mean of the first delta coefficient of each individual dimension in self.component

Return type np.array, [n_feats,]

second_derivative_mean()
Compute the mean of the second empirical derivative (2nd delta coefficient) on the last dimension (time).

Returns

The mean of the second delta coefficient of each individual dimension in self.component

Return type np.array, [n_feats,]

std()
Compute the standard deviation of self.component on the last dimension (time).

Returns

The standard deviation of each individual dimension in self.component

Return type np.array, [n_feats,]

first_derivative_std()
Compute the std of the first empirical derivative (delta coefficient) on the last dimension (time).

Returns

The std of the first delta coefficient of each individual dimension in self.component

Return type np.array, [n_feats,]

second_derivative_std()
Compute the std of the second empirical derivative (2nd delta coefficient) on the last dimension (time).

Returns

The std of the second delta coefficient of each individual dimension in self.component

Return type np.array, [n_feats,]

skewness()
Compute the skewness of self.component on the last dimension (time)

Returns

The skewness of each individual dimension in self.component

Return type np.array, [n_feats,]

first_derivative_skewness()
Compute the skewness of the first empirical derivative (delta coefficient) on the last dimension (time).

Returns

The skewness of the first delta coefficient of each individual dimension in self.component

Return type np.array, [n_feats,]

second_derivative_skewness()
Compute the skewness of the second empirical derivative (2nd delta coefficient) on the last dimension (time).

Returns

The skewness of the second delta coefficient of each individual dimension in self.component

Return type np.array, [n_feats,]

kurtosis()
Compute the kurtosis of self.component on the last dimension (time)

Returns

The kurtosis of each individual dimension in self.component

Return type np.array, [n_feats,]

first_derivative_kurtosis()
Compute the kurtosis of the first empirical derivative (delta coefficient) on the last dimension (time).

Returns

The kurtosis of the first delta coefficient of each individual dimension in self.component

Return type np.array, [n_feats,]

second_derivative_kurtosis()
Compute the kurtosis of the second empirical derivative (2nd delta coefficient) on the last dimension (time).

Returns

The kurtosis of the second delta coefficient of each individual dimension in self.component

Return type np.array, [n_feats,]

first_quartile()
Compute the first quartile on the last dimension (time).

Returns

The first quartile of each individual dimension in self.component

Return type np.array, [n_feats,]

second_quartile()
Compute the second quartile on the last dimension (time). Same as the median.

Returns

The second quartile of each individual dimension in self.component (same as the median)

Return type np.array, [n_feats,]

third_quartile()
Compute the third quartile on the last dimension (time)

Returns

The third quartile of each individual dimension in self.component

Return type np.array, [n_feats,]

q2_q1_range()
Compute second and first quartiles. Return q2 - q1

Returns

The q2 - q1 range of each individual dimension in self.component

Return type np.array, [n_feats,]

q3_q2_range()
Compute third and second quartiles. Return q3 - q2

Returns

The q3 - q2 range of each individual dimension in self.component

Return type np.array, [n_feats,]

q3_q1_range()
Compute third and first quartiles. Return q3 - q1

Returns

The q3 - q1 range of each individual dimension in self.component

Return type np.array, [n_feats,]

percentile_1()
Compute the 1% percentile.

Returns

The 1st percentile of each individual dimension in self.component

Return type np.array, [n_feats,]

percentile_99()
Compute the 99% percentile.

Returns

The 99th percentile of each individual dimension in self.component

Return type np.array, [n_feats,]

percentile_1_99_range()
Compute 99% percentile and 1% percentile. Return the range.

Returns

The 99th - 1st percentile range of each individual dimension in self.component

Return type np.array, [n_feats,]

linear_regression_offset()
Consider each row of self.component as a time series over which we fit a line. Return the offset of that fitted line.

Returns

The linear regression offset of each individual dimension in self.component

Return type np.array, [n_feats,]

linear_regression_slope()
Consider each row of self.component as a time series over which we fit a line. Return the slope of that fitted line.

Returns

The linear regression slope of each individual dimension in self.component

Return type np.array, [n_feats,]

linear_regression_mse()
Fit a line to the data. Compute the MSE.

Returns

The linear regression MSE of each individual dimension in self.component

Return type np.array, [n_feats,]

CHAPTER 3

Feature extraction

An alternative to extracting features with the `Waveform` class is to use functions specifically written for that purpose, either with the vanilla approach, or with the multiprocessing approach.

3.1 Feature Extraction

3.1.1 Vanilla feature extraction

This file contains functions to compute features.

`surfboard.feature_extraction.`**`load_waveforms_from_paths`**(*paths*, *sample_rate*)
Loads waveforms from paths using multiprocessing

`surfboard.feature_extraction.`**`extract_features_from_paths`**(*paths*, *components_list*, *statistics_list=None*, *sample_rate=44100*)
Function which loads waveforms, computes the components and statistics and returns them, without the need to store the waveforms in memory. This is to minimize the memory footprint when running over multiple files.

Parameters

- **`paths`** (*`list of str`*) – .wav to compute
- **`components_list`** (*`list of str/dict`*) – This is a list of the methods which should be applied to all the waveform objects in waveforms. If a dict, this also contains arguments to the sound.Waveform methods.
- **`statistics_list`** (*`list of str`*) – This is a list of the methods which should be applied to all the time-dependent features computed from the waveforms.
- **`sample_rate`** (*`int > 0`*) – sampling rate to load the waveforms

Returns

pandas dataframe where every row corresponds to features extracted for one of the waveforms and columns represent individual features.

Return type pandas DataFrame

surfboard.feature_extraction.**extract_features_from_waveform**(*components_list*, *statistics_list*, *waveform*)

Given one waveform, a list of components and statistics, extract the features from the waveform.

Parameters

- **components_list** (*list of str or dict*) – This is a list of the methods which should be applied to all the waveform objects in waveforms. If a dict, this also contains arguments to the sound.Waveform methods.
- **statistics_list** (*list of str*) – This is a list of the methods which should be applied to all the "time-dependent" components computed from the waveforms.
- **waveform** (Waveform) – the waveform object to extract components from.

Returns

Dictionary mapping names to numerical components extracted for this waveform.

Return type dict

surfboard.feature_extraction.**extract_features**(*waveforms*, *components_list*, *statistics_list=None*)

This is an important function. Given a list of Waveform objects, a list of Waveform methods in the form of strings and a list of Barrel methods in the form of strings, compute the time-independent features resulting. This function does multiprocessing.

Parameters

- **waveforms** (*list of Waveform*) – This is a list of waveform objects
- **components_list** (*list of str/dict*) – This is a list of the methods which should be applied to all the waveform objects in waveforms. If a dict, this also contains arguments to the sound.Waveform methods.
- **statistics_list** (*list of str*) – This is a list of the methods which should be applied to all the time-dependent features computed from the waveforms.

Returns

pandas dataframe where every row corresponds to features extracted for one of the waveforms and columns represent individual features.

Return type pandas DataFrame

3.1.2 Feature extraction with multiprocessing

This file contains functions to compute features with multiprocessing.

surfboard.feature_extraction_multiprocessing.**load_waveform_from_path**(*sample_rate*, *path*)

Helper function to access constructor with Pool

Parameters

- **sample_rate** (*int*) – The sample rate to load the Waveform object
- **path** (*str*) – The path to the audio file to load

Returns The loaded Waveform object

Return type *Waveform*

surfboard.feature_extraction_multiprocessing.**load_waveforms_from_paths**(*paths*, *sample_rate*, *num_proc=1*)

Loads waveforms from paths using multiprocessing

Parameters

- **paths** (*list of str*) – A list of paths to audio files
- **sample_rate** (*int*) – The sample rate to load the audio files
- **num_proc** (*int >= 1*) – The number of parallel processes to run

Returns List of loaded Waveform objects

Return type list of Waveform

surfboard.feature_extraction_multiprocessing.**extract_features_from_path**(*components_list*, *statistics_list*, *sample_rate*, *path*)

Function which loads a waveform, computes the components and statistics and returns them, without the need to store the waveforms in memory. This is to prevent accumulating too much memory.

Parameters

- **components_list** (*list of str/dict*) – This is a list of the methods which should be applied to all the waveform objects in waveforms. If a dict, this also contains arguments to the sound.Waveform methods.
- **statistics_list** (*list of str*) – This is a list of the methods which should be applied to all the "time-dependent" features computed from the waveforms.
- **sample_rate** (*int > 0*) – sampling rate to load the waveforms
- **path** (*str*) – path to audio file to extract features from

Returns Dictionary mapping feature names to values.

Return type dict

surfboard.feature_extraction_multiprocessing.**extract_features_from_paths**(*paths*, *components_list*, *statistics_list=None*, *sample_rate=44100*, *num_proc=1*)

Function which loads waveforms, computes the features and statistics and returns them, without the need to store the waveforms in memory. This is to prevent accumulating too much memory.

Parameters

- **paths** (*list of str*) – .wav to compute
- **components_list** (*list of str or dict*) – This is a list of the methods which should be applied to all the waveform objects in waveforms. If a dict, this also contains arguments to the sound.Waveform methods.

- **statistics_list** (*list of str*) – This is a list of the methods which should be applied to all the "time-dependent" features computed from the waveforms.
- **sample_rate** (*int > 0*) – sampling rate to load the waveforms

Returns

pandas dataframe where every row corresponds to features extracted for one of the waveforms and columns represent individual features.

Return type pandas DataFrame

surfboard.feature_extraction_multiprocessing.**extract_features**(*waveforms*, *components_list*, *statistics_list=None*, *num_proc=1*)

This is an important function. Given a list of Waveform objects, a list of Waveform methods in the form of strings and a list of Barrel methods in the form of strings, compute the time-independent features resulting. This function does multiprocessing.

Parameters

- **waveforms** (*list of Waveform*) – This is a list of waveform objects
- **components_list** (*list of str or dict*) – This is a list of the methods which should be applied to all the waveform objects in waveforms. If a dict, this also contains arguments to the sound.Waveform methods.
- **statistics_list** (*list of str*) – This is a list of the methods which should be applied to all the "time-dependent" features computed from the waveforms.
- **num_proc** (*int >= 1*) – The number of parallel processes to run

Returns

pandas dataframe where every row corresponds to features extracted for one of the waveforms and columns represent individual features.

Return type pandas DataFrame

CHAPTER 4

Under the hood

Under the hood lies a variety of files containing functions which are imported by the `Waveform` class. We split the code as such for the sake of readability.

4.1 Under the Hood

4.1.1 hnr.py

This function is inspired by the Speech Analysis repository at https://github.com/brookemosby/Speech_Analysis

`surfboard.hnr.`**`get_harmonics_to_noise_ratio`**(*waveform*, *sample_rate*, *min_pitch=75.0*, *silence_threshold=0.1*, *periods_per_window=4.5*)

Given a waveform, its sample rate, some conditions for voiced and unvoiced frames (including min pitch and silence threshold), and a "periods per window" argument, compute the harmonics to noise ratio. This is a good measure of voice quality and is an important metric in cognitively impaired patients. Compute the mean hnr_vector: harmonics to noise ratio.

Parameters

- **`waveform`** (*`np.array, [T, ]`*) – waveform signal
- **`sample_rate`** (*`int > 0`*) – sampling rate of the waveform
- **`min_pitch`** (*`float > 0`*) – minimum acceptable pitch. converts to maximum acceptable period.
- **`silence_threshold`** (*`1 >= float >= 0`*) – needs to be in [0, 1]. Below this amplitude, does not consider frames.
- **`periods_per_window`** (*`float > 0`*) – 4.5 is best for speech.

Returns

Harmonics to noise ratio of the entire considered waveform.

Return type float

4.1.2 jitters.py

This file contains all the functions needed to compute the jitters of a waveform.

surfboard.jitters.**validate_frequencies**(*frequencies*, *p_floor*, *p_ceil*, *max_p_factor*)

Given a sequence of frequencies, [f1, f2, ..., fn], a minimum period, maximum period, and maximum period factor, first remove all frequencies computed as 0. Then, if periods are the inverse frequencies, this function returns True if the sequence of periods satisfies the conditions, otherwise returns False. In order to satisfy the maximum period factor, the periods have to satisfy pi / pi+1 < max_p_factor and pi+1 / pi < max_p_factor.

Parameters

- **frequencies** (*sequence, eg list, of floats*) – sequence of frequencies == 1 / period.
- **p_floor** (*float*) – minimum acceptable period.
- **p_ceil** (*float*) – maximum acceptable period.
- **max_p_factor** (*float*) – value to use for the period factor principle

Returns True if the conditions are met, False otherwise.

Return type bool

surfboard.jitters.**get_mean_period**(*frequencies*, *p_floor*, *p_ceil*, *max_p_factor*)

Given a sequence of frequencies, passes these through the validation phase, then computes the mean of the remaining periods. Note period = 1/f.

Parameters

- **frequencies** (*sequence, eg list, of floats*) – sequence of frequencies
- **p_floor** (*float*) – minimum acceptable period.
- **p_ceil** (*float*) – maximum acceptable period.
- **max_p_factor** (*float*) – value to use for the period factor principle

Returns The mean of the acceptable periods.

Return type float

surfboard.jitters.**get_local_absolute_jitter**(*frequencies*, *p_floor*, *p_ceil*, *max_p_factor*)

Given a sequence of frequencies, and some period conditions, compute the local absolute jitter, as per https://royalsocietypublishing.org/action/downloadSupplement?doi=10.1098%2Frsif.2010.0456&file=rsif20100456supp1.pdf

Parameters

- **frequencies** (*sequence, eg list, of floats*) – sequence of estimated frequencies
- **p_floor** (*float*) – minimum acceptable period.
- **p_ceil** (*float*) – maximum acceptable period.
- **max_p_factor** (*float*) – value to use for the period factor principle

Returns the local absolute jitter.

Return type float

surfboard.jitters.**get_local_jitter**(*frequencies*, *p_floor*, *p_ceil*, *max_p_factor*)

Given a sequence of frequencies, and some period conditions, compute the local jitter, as per https://royalsocietypublishing.org/action/downloadSupplement?doi=10.1098%2Frsif.2010.0456&file=rsif20100456supp1.pdf

Parameters

- **frequencies** (*sequence, eg list, of floats*) – sequence of estimated frequencies
- **p_floor** (*float*) – minimum acceptable period.
- **p_ceil** (*float*) – maximum acceptable period.
- **max_p_factor** (*float*) – value to use for the period factor principle

Returns the local jitter.

Return type float

surfboard.jitters.**get_rap_jitter**(*frequencies*, *p_floor*, *p_ceil*, *max_p_factor*)

Given a sequence of frequencies, and some period conditions, compute the rap jitter, as per https://royalsocietypublishing.org/action/downloadSupplement?doi=10.1098%2Frsif.2010.0456&file=rsif20100456supp1.pdf

Parameters

- **frequencies** (*sequence, eg list, of floats*) – sequence of estimated frequencies
- **p_floor** (*float*) – minimum acceptable period.
- **p_ceil** (*float*) – maximum acceptable period.
- **max_p_factor** (*float*) – value to use for the period factor principle

Returns the rap jitter.

Return type float

surfboard.jitters.**get_ppq5_jitter**(*frequencies*, *p_floor*, *p_ceil*, *max_p_factor*)

Given a sequence of frequencies, and some period conditions, compute the ppq5 jitter, as per https://royalsocietypublishing.org/action/downloadSupplement?doi=10.1098%2Frsif.2010.0456&file=rsif20100456supp1.pdf

Parameters

- **frequencies** (*sequence, eg list, of floats*) – sequence of estimated frequencies
- **p_floor** (*float*) – minimum acceptable period.
- **p_ceil** (*float*) – maximum acceptable period.
- **max_p_factor** (*float*) – value to use for the period factor principle

Returns the ppq5 jitter.

Return type float

surfboard.jitters.**get_ddp_jitter**(*frequencies*, *p_floor*, *p_ceil*, *max_p_factor*)

Given a sequence of frequencies, and some period conditions, compute the ddp jitter, as per http://www.fon.hum.uva.nl/praat/manual/PointProcess__Get_jitter__ddp____.html

Parameters

- **frequencies** (*sequence, eg list, of floats*) – sequence of estimated frequencies
- **p_floor** (*float*) – minimum acceptable period.
- **p_ceil** (*float*) – maximum acceptable period.
- **max_p_factor** (*float*) – value to use for the period factor principle

Returns the ddp jitter.

Return type float

surfboard.jitters.**get_jitters**(*f0_contour*, *p_floor=0.0001*, *p_ceil=0.02*, *max_p_factor=1.3*)

Compute the jitters mathematically, according to certain conditions given by p_floor, p_ceil and max_p_factor.

Parameters

- **f0_contour** (*np.array [T / hop_length,]*) – the fundamental frequency contour.
- **p_floor** (*float*) – minimum acceptable period.
- **p_ceil** (*float*) – maximum acceptable period.
- **max_p_factor** (*float*) – value to use for the period factor principle

Returns

Dictionary mapping strings to floats, with keys ”localJitter”, “localabsoluteJitter”, “rapJitter”, “ppq5Jitter”, “ddpJitter”

Return type dict

4.1.3 shimmers.py

This file contains all the functions needed to compute the shimmers of a waveform.

surfboard.shimmers.**validate_amplitudes**(*amplitudes*, *frequencies*, *max_a_factor*, *p_floor*, *p_ceil*, *max_p_factor*)

First check that frequencies corresponding to this set of amplitudes are valid. Then Returns True if this set of amplitudes is validated as per the maximum amplitude factor principle, i.e. if amplitudes = [a1, a2, . . . , an], this functions returns false if any two successive amplitudes alpha, beta satisfy alpha / beta > max_a_factor or beta / alpha > max_a_factor. False otherwise.

Parameters

- **amplitudes** (*list*) – ordered list of amplitudes to run by this principle.
- **frequencies** (*sequence, eg list, of floats*) – sequence of frequencies == 1 / period.
- **max_a_factor** (*float*) – the threshold to run the principle.
- **p_floor** (*float*) – minimum acceptable period.
- **p_ceil** (*float*) – maximum acceptable period.
- **max_p_factor** (*float*) – value to use for the period factor principle

Returns

True if this set of amplitudes satisifies the principle and this set of frequencies satisfies the period condition, False otherwise.

Return type bool

surfboard.shimmers.**get_local_shimmer**(*amplitudes*, *frequencies*, *max_a_factor*, *p_floor*, *p_ceil*, *max_p_factor*)

Given a list of amplitudes, returns the localShimmer as per https://royalsocietypublishing.org/action/downloadSupplement?doi=10.1098%2Frsif.2010.0456&file=rsif20100456supp1.pdf

Parameters

- **amplitudes** (*list of floats*) – The list of peak amplitudes in each frame.
- **max_a_factor** (*float*) – The maximum A factor to validate amplitudes. See validate_amplitudes().

Returns The local shimmer computed over this sequence of amplitudes.

Return type float

surfboard.shimmers.**get_local_db_shimmer**(*amplitudes*, *frequencies*, *max_a_factor*, *p_floor*, *p_ceil*, *max_p_factor*)

Given a list of amplitudes, returns the localdbShimmer as per https://royalsocietypublishing.org/action/downloadSupplement?doi=10.1098%2Frsif.2010.0456&file=rsif20100456supp1.pdf

Parameters

- **amplitudes** (*list of floats*) – The list of peak amplitudes in each frame.
- **max_a_factor** (*float*) – The maximum A factor to validate amplitudes. See validate_amplitudes().

Returns The local DB shimmer computed over this sequence of amplitudes.

Return type float

surfboard.shimmers.**get_apq_shimmer**(*amplitudes*, *frequencies*, *max_a_factor*, *p_floor*, *p_ceil*, *max_p_factor*, *apq_no*)

Given a list of amplitudes, returns the apq{apq_no}Shimmer as per https://royalsocietypublishing.org/action/downloadSupplement?doi=10.1098%2Frsif.2010.0456&file=rsif20100456supp1.pdf

Parameters

- **amplitudes** (*list of floats*) – The list of peak amplitudes in each frame.
- **max_a_factor** (*float*) – The maximum A factor to validate amplitudes. See validate_amplitudes().
- **apq_no** (*int*) – an odd number which corresponds to the number of neighbors used to compute the shimmer.

Returns

The apqShimmer computed over this sequence of amplitudes with this APQ number.

Return type float

surfboard.shimmers.**get_shimmers**(*waveform*, *sample_rate*, *f0_contour*, *max_a_factor=1.6*, *p_floor=0.0001*, *p_ceil=0.02*, *max_p_factor=1.3*)

Compute five different types of shimmers using functions defined above.

Parameters

- **waveform** (*np.array, [T,]*) – waveform over which to compute shimmers
- **sample_rate** (*int*) – sampling rate of waveform.
- **f0_contour** (*np.array, [T / hop_length,]*) – the fundamental frequency contour.
- **max_a_factor** (*float*) – value to use for amplitude factor principle

- **p_floor** (*float*) – minimum acceptable period.
- **p_ceil** (*float*) – maximum acceptable period.
- **max_p_factor** (*float*) – value to use for the period factor principle

Returns

Dictionary mapping strings to floats, with keys "localShimmer", "localdbShimmer", "apq3Shimmer", "apq5Shimmer", "apq11Shimmer"

Return type dict

4.1.4 dfa.py

surfboard.dfa.**get_deviation_for_dfa**(*signal*, *window_length*)

Given a signal, compute the trend value for one window length, as per https://link.springer.com/article/10.1186/1475-925X-6-23 In order to get the overall DFA (detrended fluctuation analysis), compute this for a variety of window lengths, then plot that on a log-log graph, and get the slope.

Parameters

- **signal** (*np.array, [T,]*) – waveform
- **window_length** (*int > 0*) – L in the paper linked above. Length of windows for trend.

Returns

average rmse for fitting lines on chunks of window lengths on the cumulative sums of this signal.

Return type float

surfboard.dfa.**get_dfa**(*signal*, *window_lengths*)

Given a signal, compute the DFA (detrended fluctuation analysis) as per https://link.springer.com/article/10.1186/1475-925X-6-23 See paper equations (13) to (16) for more information.

4.1.5 spectrum.py

Spectrum features. The code in this file is inspired by audiocontentanalysis.org For more details, visit the pyACA package: https://github.com/alexanderlerch/pyACA

surfboard.spectrum.**get_spectral_centroid**(*magnitude_spectrum*, *sample_rate*)

Given the magnitude spectrum and the sample rate of the waveform from which it came, compute the spectral centroid.

Parameters

- **magnitude_spectrum** (*np.array, [n_frequencies, T / hop_length]*) – the spectrogram
- **sample_rate** (*int*) – The sample rate of the waveform

Returns the spectral centroid sequence in Hz.

Return type np.array [1, T / hop_length]

surfboard.spectrum.**get_spectral_slope**(*magnitude_spectrum*, *sample_rate*)

Given the magnitude spectrum and the sample rate of the waveform from which it came, compute the spectral slope.

Parameters

- **magnitude_spectrum** (*np.array, [n_frequencies, T / hop_length]*) – the spectrogram
- **sample_rate** (*int*) – The sample rate of the waveform

Returns the spectral slope sequence.

Return type np.array [1, T / hop_length]

surfboard.spectrum.**get_spectral_flux**(*magnitude_spectrum*, *sample_rate*)

Given the magnitude spectrum and the sample rate of the waveform from which it came, compute the spectral flux.

Parameters

- **magnitude_spectrum** (*np.array, [n_frequencies, T / hop_length]*) – the spectrogram
- **sample_rate** (*int*) – The sample rate of the waveform

Returns the spectral flux sequence.

Return type np.array [1, T / hop_length]

surfboard.spectrum.**get_spectral_spread**(*magnitude_spectrum*, *sample_rate*)

Given the magnitude spectrum and the sample rate of the waveform from which it came, compute the spectral spread.

Parameters

- **magnitude_spectrum** (*np.array, [n_frequencies, T / hop_length]*) – the spectrogram
- **sample_rate** (*int*) – The sample rate of the waveform

Returns the spectral spread (Hz).

Return type np.array [1, T / hop_length]

surfboard.spectrum.**get_spectral_skewness**(*magnitude_spectrum*, *sample_rate*)

Given the magnitude spectrum and the sample rate of the waveform from which it came, compute the spectral skewness.

Parameters

- **magnitude_spectrum** (*np.array, [n_frequencies, T / hop_length]*) – the spectrogram
- **sample_rate** (*int*) – The sample rate of the waveform

Returns the spectral skewness.

Return type np.array [1, T / hop_length]

surfboard.spectrum.**get_spectral_kurtosis**(*magnitude_spectrum*, *sample_rate*)

Given the magnitude spectrum and the sample rate of the waveform from which it came, compute the spectral skewness.

Parameters

- **magnitude_spectrum** (*np.array, [n_frequencies, T / hop_length]*) – the spectrogram
- **sample_rate** (*int*) – The sample rate of the waveform

Returns the spectral kurtosis.

Return type np.array [1, T / hop_length]

4.1.6 misc_components.py

This file contains components which do not fall under one category.

surfboard.misc_components.**get_crest_factor**(*waveform*, *sample_rate*, *rms*, *frame_length_seconds=0.04*, *hop_length_seconds=0.01*)

Get the crest factor of this waveform, on sliding windows. This value measures the local intensity of peaks in a waveform. Implemented as per: https://en.wikipedia.org/wiki/Crest_factor

Parameters

- **waveform** (*np.array, [T,]*) – waveform over which to compute crest factor
- **sample_rate** (*int > 0*) – number of samples per second in waveform
- **rms** (*np.array, [1, T / hop_length]*) – energy values.
- **frame_length_seconds** (*float*) – length of the sliding window, in seconds.
- **hop_length_seconds** (*float*) – how much the window shifts for every timestep, in seconds.

Returns Crest factor for each frame.

Return type np.array, [1, T / hop_length]

surfboard.misc_components.**get_f0**(*waveform*, *sample_rate*, *hop_length_seconds=0.01*, *method='swipe'*, *f0_min=60*, *f0_max=300*)

Compute the F0 contour using PYSPTK: https://github.com/r9y9/pysptk/.

Parameters

- **waveform** (*np.array, [T,]*) – waveform over which to compute f0
- **sample_rate** (*int > 0*) – number of samples per second in waveform
- **hop_length** (*int*) – hop size argument in pysptk.swipe. Corresponds to hopsize in the window sliding of the computation of f0.
- **method** (*str*) – is one of 'swipe' or 'rapt'. Define which method to use for f0 calculation. See https://github.com/r9y9/pysptk

Returns

Dictionary containing keys:

"contour" (np.array, [1, t1]): f0 contour of waveform. Contains unvoiced frames.

"values" (np.array, [1, t2]): nonzero f0 values waveform. Note that this discards all unvoiced frames. Use to compute mean, std, and other statistics.

"mean" (float): mean of the f0 contour. "std" (float): standard deviation of the f0 contour.

Return type dict

surfboard.misc_components.**get_ppe**(*rat_f0*)

Compute pitch period entropy. Here is a reference MATLAB implementation: https://github.com/Mak-Sim/Troparion/blob/5126f434b96e0c1a4a41fa99dd9148f3c959cfac/Perturbation_analysis/pitch_period_entropy.m Note that computing the PPE relies on the existence of voiced portions in the F0 trajectory.

Parameters **rat_f0** (*np.array*) – f0 voiced frames divided by f_min

Returns The pitch period entropy, as per http://www.maxlittle.net/students/thesis_tsanas.pdf

Return type float

surfboard.misc_components.**get_shannon_entropy**(*sequence*)
Given a sequence, compute the Shannon Entropy, defined in https://ijssst.info/Vol-16/No-4/data/8258a127.pdf

Parameters **sequence** (*np.array, [t,]*) – sequence over which to compute.

Returns shannon entropy.

Return type float

surfboard.misc_components.**get_shannon_entropy_slidingwindow**(*waveform*, *sample_rate*, *frame_length_seconds=0.04*, *hop_length_seconds=0.01*)
Same function as above, but decorated by the metric_slidingwindow decorator. See above for documentation on this.

Parameters

- **waveform** (*np.array, [T,]*) – waveform over which to compute the shannon entropy array
- **sample_rate** (*int > 0*) – number of samples per second in waveform
- **frame_length_seconds** (*float*) – length of the sliding window, in seconds.
- **hop_length_seconds** (*float*) – how much the window shifts for every timestep, in seconds.

Returns Shannon entropy over windows.

Return type np.array, [1, T/hop_length]

surfboard.misc_components.**get_loudness**(*waveform*, *sample_rate*)
Compute the loudness of waveform using the pyloudnorm package. See https://github.com/csteinmetz1/pyloudnorm for more details on potential arguments to the functions below.

Parameters

- **waveform** (*np.array, [T,]*) – waveform to compute loudness on
- **sample_rate** (*int > 0*) – sampling rate of waveform

Returns the loudness of self.waveform

Return type float

surfboard.misc_components.**get_loudness_slidingwindow**(*waveform*, *sample_rate*, *frame_length_seconds=0.04*, *hop_length_seconds=0.01*)
Same function as get_loudness, but decorated by the metric_slidingwindow decorator. See get_loudness documentation for this.

Parameters

- **waveform** (*np.array, [T,]*) – waveform over which to compute the kurtosis array
- **sample_rate** (*int > 0*) – number of samples per second in waveform
- **frame_length_seconds** (*float*) – length of the sliding window, in seconds.
- **hop_length_seconds** (*float*) – how much the window shifts for every timestep, in seconds.

Returns Frame level loudness

Return type np.array, [1, T / hop_length]

surfboard.misc_components.**get_kurtosis_slidingwindow**(*waveform*, *sample_rate*, *frame_length_seconds=0.04*, *hop_length_seconds=0.01*)

Same function as above, but decorated by the metric_slidingwindow decorator. See above documentation for this.

Parameters

- **waveform** (*np.array, [T,]*) – waveform over which to compute the kurtosis array
- **sample_rate** (*int > 0*) – number of samples per second in waveform
- **frame_length_seconds** (*float*) – length of the sliding window, in seconds.
- **hop_length_seconds** (*float*) – how much the window shifts for every timestep, in seconds.

Returns Kurtosis over windows

Return type np.array, [1, T/hop_length]

surfboard.misc_components.**get_log_energy**(*matrix*, *time_axis=-1*)

Compute the log energy of a matrix as per Abeyrante et al. 2013.

Parameters

- **matrix** (*np.array*) – matrix over which to compute. This has to be a 1 or 2-dimensional np.array
- **time_axis** (*int >= 0*) – the axis in matrix which corresponds to time.

Returns

The log energy of matrix, computed as per the paper above.

Return type float

surfboard.misc_components.**get_log_energy_slidingwindow**(*waveform*, *sample_rate*, *frame_length_seconds=0.04*, *hop_length_seconds=0.01*)

Same function as above, but decorated by the metric_slidingwindow decorator. See above documentation for this.

Parameters

- **waveform** (*np.array, [T,]*) – waveform over which to compute the log energy array
- **sample_rate** (*int > 0*) – number of samples per second in waveform
- **frame_length_seconds** (*float*) – length of the sliding window, in seconds.
- **hop_length_seconds** (*float*) – how much the window shifts for every timestep, in seconds.

Returns log_energy over windows

Return type np.array, [1, T/hop_length]

surfboard.misc_components.**get_bark_spectrogram**(*waveform*, *sample_rate*, *n_fft_seconds*, *hop_length_seconds*)

Convert a spectrogram to a bark-band spectrogram.

Parameters

- **waveform** (*np.array, [T,]*) – waveform over which to compute the bark spectrogram.
- **sample_rate** (*int > 0*) – number of samples per second in waveform.
- **n_fft_seconds** (*float > 0*) – length of the fft window, in seconds

Returns

The original spectrogram with bins converted into the Bark scale.

Return type np.array, [n_bark_bands, t]

4.1.7 utils.py

This file contains a variety of helper functions for the surfboard package.

surfboard.utils.**metric_slidingwindow**(*frame_length*, *hop_length*, *truncate_end=False*)

We use this decorator to decorate functions which take a sequence as an input and return a metric (float). For example the sum of a sequence. This decorator will enable us to quickly compute the metrics over a sliding window. Note the existence of the implicit decorator below which allows us to have arguments to the decorator.

Parameters

- **frame_length** (*int*) – The length of the sliding window
- **hop_length** (*int*) – How much to slide the window every time
- **truncate_end** (*bool*) – whether to drop frames which are shorter than frame_length (the end frames, typically)

Returns

The function which computes the metric over sliding windows.

Return type function

surfboard.utils.**numseconds_to_numsamples**(*numseconds*, *sample_rate*)

Convert a number of seconds a sample rate to the number of samples for n_fft, frame_length and hop_length computation. Find the closest power of 2 for efficient computations.

Parameters

- **numseconds** (*float*) – number of seconds that we want to convert
- **sample_rate** (*int*) – how many samples per second

Returns closest power of 2 to int(numseconds * sample_rate)

Return type int

surfboard.utils.**max_peak_amplitude**(*signal*)

Returns the maximum absolute value of a signal.

Parameters **[T,]** (*np.array*) – a waveform

Returns the maximum amplitude of this waveform, in absolute value

Return type float

surfboard.utils.**peak_amplitude_slidingwindow**(*signal*, *sample_rate*, *frame_length_seconds=0.04*, *hop_length_seconds=0.01*)

Apply the metric_slidingwindow decorator to the the peak amplitude computation defined above, effectively computing frequency from fft over sliding windows.

Parameters

- **signal** (*np.array [T,]*) – waveform over which to compute.
- **sample_rate** (*int*) – number of samples per second in the waveform
- **frame_length_seconds** (*float*) – how many seconds in one frame. This value is defined in seconds instead of number of samples.
- **hop_length_seconds** (*float*) – how many seconds frames shift each step. This value is defined in seconds instead of number of samples.

Returns peak amplitude on each window.

Return type np.array, [1, T / hop_length]

surfboard.utils.**shifted_sequence**(*sequence*, *num_sequences*)

Given a sequence (say a list) and an integer, returns a zipped iterator of sequence[:-num_sequences + 1], sequence[1:-num_sequences + 2], etc.

Parameters

- **sequence** (*list or other iteratable*) – the sequence over which to iterate in various orders
- **num_sequences** (*int*) – the number of sequences over which we iterate. Also the number of elements which come out of the output at each call.

Returns zipped shifted sequences.

Return type iterator

surfboard.utils.**lpc_to_lsf**(*lpc_polynomial*)

This code is inspired by the following: https://uk.mathworks.com/help/dsp/ref/lpctolsflspconversion.html

Parameters **lpc_polynomial** (*list*) – length n + 1 list of lpc coefficients. Requirements is that the polynomial is ordered so that lpc_polynomial[0] == 1

Returns length n list of line spectral frequencies.

Return type list

surfboard.utils.**parse_component**(*component*)

Parse the component coming from the .yaml file.

Parameters **component** (*str or dict*) – Can be either a str, or a dictionary. Comes from the .yaml config file. If it is a string, simply return, since its the component name without arguments. Otherwise, parse.

Returns

tuple containing: str: name of the method to be called from sound.Waveform dict: arguments to be unpacked. None if no arguments to

compute.

Return type tuple

surfboard.utils.**example_audio_file**(*which_file*)

Returns the path to one of sustained_a, sustained_o or sustained_e included with the Surfboard package.

Parameters **which_file** (*str*) – One of ‘a’, ‘o’ or ‘e’

Returns The path to the chosen file.

Return type str

exception surfboard.utils.**YamlFileException**(*message*)

CHAPTER 5

Indices and tables

- genindex
- search

Python Module Index

S

Index

H

I

J

K

L

M

N

P