BioSPPy Documentation

Release 0.2.1

Instituto de Telecomunicacoes

January 09, 2017

Contents

`BioSPPy` is a toolbox for biosignal processing written in Python. The toolbox bundles together various signal processing and pattern recognition methods geared torwards the analysis of biosignals.

Highlights:

- Support for various biosignals: BVP, ECG, EDA, EEG, EMG, Respiration
- Signal analysis primitives: filtering, frequency analysis
- Clustering
- Biometrics

Contents:

CHAPTER 1

Tutorial

In this tutorial we will describe how *biosppy* enables the development of Pattern Recognition and Machine Learning workflows for the analysis of biosignals. The major goal of this package is to make these tools easily available to anyone wishing to start playing around with biosignal data, regardless of their level of knowledge in the field of Data Science. Throughout this tutorial we will discuss the major features of *biosppy* and introduce the terminology used by the package.

1.1 What are Biosignals?

Biosignals, in the most general sense, are measurements of physical properties of biological systems. These include the measurement of properties at the cellular level, such as concentrations of molecules, membrane potentials, and DNA assays. On a higher level, for a group of specialized cells (i.e. an organ) we are able to measure properties such as cell counts and histology, organ secretions, and electrical activity (the electrical system of the heart, for instance). Finally, for complex biological systems like the human being, biosignals also include blood and urine test measurements, core body temperature, motion tracking signals, and imaging techniques such as CAT and MRI scans. However, the term biosignal is most often applied to bioelectrical, time-varying signals, such as the electrocardiogram.

The task of obtaining biosignals of good quality is time-consuming, and typically requires the use of costly hardware. Access to these instruments is, therefore, usually restricted to research institutes, medical centers, and hospitals. However, recent projects like BITalino or OpenBCI have lowered the entry barriers of biosignal acquisition, fostering the Do-It-Yourself and Maker communities to develop physiological computing applications. You can find a list of biosignal platform here.

The following sub-sections briefly describe the biosignals covered by *biosppy*.

1.1.1 Blood Volume Pulse

Blood Volume Pulse (BVP) signals are...

1.1.2 Electrocardiogram

Electrocardiogrm (ECG) signals are...

1.1.3 Electrodermal Activity

Electrodermal Activity (EDA) signals are...

1.1.4 Electroencephalogram

Electroencephalogram (EEG) signals are...

1.1.5 Electromyogram

Electromyogram (EMG) signals are...

1.1.6 Respiration

Respiration (Resp) signals are...

1.2 What is Pattern Recognition?

To do.

1.3 A Note on Return Objects

Before we dig into the core aspects of the package, you will quickly notice that many of the methods and functions defined here return a custom object class. This return class is defined in *biosppy.utils.ReturnTuple*. The goal of this return class is to strengthen the semantic relationship between a function's output variables, their names, and what is described in the documentation. Consider the following function definition:

```
def compute(a, b):
    """Simultaneously compute the sum, subtraction, multiplication and
    division between two integers.

    Args:
        a (int): First input integer.
        b (int): Second input integer.

    Returns:
        (tuple): containing:
            sum (int): Sum (a + b).
            sub (int): Subtraction (a - b).
            mult (int): Multiplication (a * b).
            div (int): Integer division (a / b).

    """

    if b == 0:
        raise ValueError("Input 'b' cannot be zero.")

    v1 = a + b
    v2 = a - b
    v3 = a * b
    v4 = a / b

    return v1, v2, v3, v4
```

Note that Python doesn't actually support returning multiple objects. In this case, the `return` statement packs the objects into a tuple.

```
>>> out = compute(4, 50)
>>> type(out)
<type 'tuple'>
>>> print out
(54, -46, 200, 0)
```

This is pretty straightforward, yet it shows one disadvantage of the native Python return pattern: the semantics of the output elements (i.e. what each variable actually represents) are only implicitly defined with the ordering of the docstring. If there isn't a dosctring available (yikes!), the only way to figure out the meaning of the output is by analyzing the code itself.

This is not necessarily a bad thing. One should always try to understand, at least in broad terms, how any given function works. However, the initial steps of the data analysis process encompass a lot of experimentation and interactive exploration of the data. This is important in order to have an initial sense of the quality of the data and what information we may be able to extract. In this case, the user typically already knows what a function does, but it is cumbersome to remember by heart the order of the outputs, without having to constantly check out the documentation.

For instance, does the numpy.histogram function first return the edges or the values of the histogram? Maybe it's the edges first, which correspond to the x axis. Oops, it's actually the other way around...

In this case, it could be useful to have an explicit reference directly in the return object to what each variable represents. Returning to the example above, we would like to have something like:

```
>>> out = compute(4, 50)
>>> print out
(sum=54, sub=-46, mult=200, div=0)
```

This is exactly what *biosppy.utils.ReturnTuple* accomplishes. Rewriting the *compute* function to work with *ReturnTuple* is simple. Just construct the return object with a tuple of strings with names for each output variable:

```
from biosppy import utils

def compute_new(a, b):
    """Simultaneously compute the sum, subtraction, multiplication and
    division between two integers.

    Args:
        a (int): First input integer.
        b (int): Second input integer.

    Returns:
        (ReturnTuple): containing:
            sum (int): Sum (a + b).
            sub (int): Subtraction (a - b).
            mult (int): Multiplication (a * b).
            div (int): Integer division (a / b).

    """

    if b == 0:
        raise ValueError("Input 'b' cannot be zero.")

    v1 = a + b
    v2 = a - b
    v3 = a * b
    v4 = a / b
```

```
    # build the return object
    output = utils.ReturnTuple((v1, v2, v3, v4), ('sum', 'sub', 'mult', 'div'))

    return output
```

The output now becomes:

```
>>> out = compute_new(4, 50)
>>> print out
ReturnTuple(sum=54, sub=-46, mult=200, div=0)
```

It allows to access a specific variable by key, like a dictionary:

```
>>> out['sum']
54
```

And to list all the available keys:

```
>>> out.keys()
['sum', 'sub', 'mult', 'div']
```

It is also possible to convert the object to a more traditional dictionary, specifically an OrderedDict:

```
>>> d = out.as_dict()
>>> print d
OrderedDict([('sum', 54), ('sub', -46), ('mult', 200), ('div', 0)])
```

Dictionary-like unpacking is supported:

```
>>> some_function(**out)
```

ReturnTuple is heavily inspired by namedtuple, but without the dynamic class generation at object creation. It is a subclass of *tuple*, therefore it maintains compatibility with the native return pattern. It is still possible to unpack the variables in the usual way:

```
>>> a, b, c, d = compute_new(4, 50)
>>> print a, b, c, d
54 -46 200 0
```

The behavior is slightly different when only one variable is returned. In this case it is necessary to explicitly unpack a one-element tuple:

```
from biosppy import utils

def foo():
    """Returns 'bar'."""

    out = 'bar'

    return utils.ReturnTuple((out, ), ('out', ))
```

```
>>> out, = foo()
>>> print out
'bar'
```

1.4 A First Approach

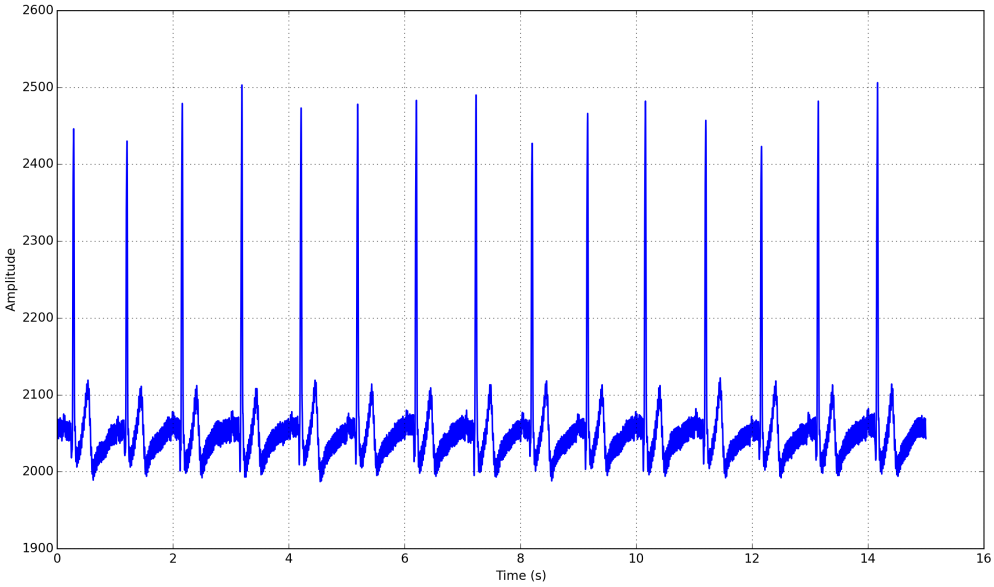
One of the major goals of *biosppy* is to provide an easy starting point into the world of biosignal processing. For that reason, we provide simple turnkey solutions for each of the supported biosignal types. These functions implement

typical methods to filter, transform, and extract signal features. Let's see how this works for the example of the ECG signal.

The GitHub repository includes a few example signals (see here). To load and plot the raw ECG signal follow:

```
>>> import numpy as np
>>> import pylab as pl
>>> from biosppy import storage
>>>
>>> signal, mdata = storage.load_txt('../examples/ecg.txt')
>>> Fs = mdata['sampling_rate']
>>> N = len(signal)  # number of samples
>>> T = (N - 1) / Fs  # duration
>>> ts = np.linspace(0, T, N, endpoint=False)  # relative timestamps
>>> pl.plot(ts, signal, lw=2)
>>> pl.grid()
>>> pl.show()
```

This should produce a similar output to the one shown below.

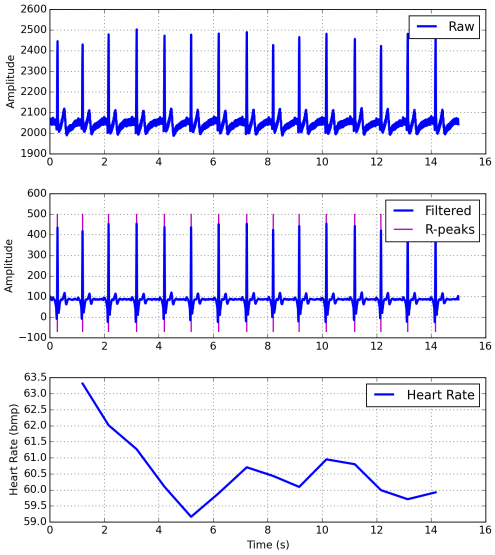


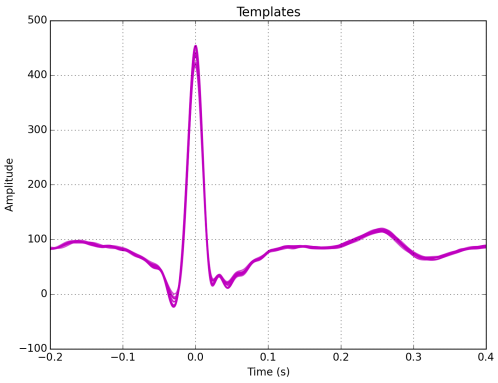
This signal is a Lead I ECG signal acquired at 1000 Hz, with a resolution of 12 bit. Although of good quality, it exhibits powerline noise interference, has a DC offset resulting from the acquisition device, and we can also observe the influence of breathing in the variability of R-peak amplitudes.

We can minimize the effects of these artifacts and extract a bunch of features with the *biosppy.signals.ecg.ecg* function:

```
>>> from biosppy.signals import ecg
>>> out = ecg.ecg(signal=signal, sampling_rate=Fs, show=True)
```

It should produce a plot like the one below.





1.5 Signal Processing

To do..

1.6 Clustering

To do..

1.7 Biometrics

To do..

1.8 What's Next?

To do..

1.9 References

To do.

CHAPTER 2

API Reference

This part of the documentation details the complete `BioSPPy` API.

2.1 Packages

2.1.1 biosppy.signals

This sub-package provides methods to process common physiological signals (biosignals).

Modules

biosppy.signals.bvp

This module provides methods to process Blood Volume Pulse (BVP) signals.

copyright

3. 2015 by Instituto de Telecomunicacoes

license BSD 3-clause, see LICENSE for more details.

`biosppy.signals.bvp.`**`bvp`**(*signal=None*, *sampling_rate=1000.0*, *show=True*)

Process a raw BVP signal and extract relevant signal features using default parameters.

Parameters

- **`signal`** (*`array`*) – Raw BVP signal.
- **`sampling_rate`** (*`int, float, optional`*) – Sampling frequency (Hz).
- **`show`** (*`bool, optional`*) – If True, show a summary plot.

Returns

- **ts** (*array*) – Signal time axis reference (seconds).
- **filtered** (*array*) – Filtered BVP signal.
- **onsets** (*array*) – Indices of BVP pulse onsets.
- **heart_rate_ts** (*array*) – Heart rate time axis reference (seconds).
- **heart_rate** (*array*) – Instantaneous heart rate (bpm).

biosppy.signals.bvp.**find_onsets**(*signal=None*, *sampling_rate=1000.0*)

Determine onsets of BVP pulses.

Skips corrupted signal parts.

Parameters

- **signal** (*array*) – Input filtered BVP signal.
- **sampling_rate** (*int, float, optional*) – Sampling frequency (Hz).

Returns **onsets** (*array*) – Indices of BVP pulse onsets.

biosppy.signals.ecg

This module provides methods to process Electrocardiographic (ECG) signals. Implemented code assumes a single-channel Lead I like ECG signal.

copyright

3. 2015 by Instituto de Telecomunicacoes

license BSD 3-clause, see LICENSE for more details.

biosppy.signals.ecg.**christov_segmenter**(*signal=None*, *sampling_rate=1000.0*)

ECG R-peak segmentation algorithm.

Follows the approach by Christov *[Chri04]*.

Parameters

- **signal** (*array*) – Input filtered ECG signal.
- **sampling_rate** (*int, float, optional*) – Sampling frequency (Hz).

Returns **rpeaks** (*array*) – R-peak location indices.

References

biosppy.signals.ecg.**compare_segmentation**(*reference=None*, *test=None*, *sampling_rate=1000.0*, *offset=0*, *minRR=None*, *tol=0.05*)

Compare the segmentation performance of a list of R-peak positions against a reference list.

Parameters

- **reference** (*array*) – Reference R-peak location indices.
- **test** (*array*) – Test R-peak location indices.
- **sampling_rate** (*int, float, optional*) – Sampling frequency (Hz).

- **offset** (*int, optional*) – Constant a priori offset (number of samples) between reference and test R-peak locations.
- **minRR** (*float, optional*) – Minimum admissible RR interval (seconds).
- **tol** (*float, optional*) – Tolerance between corresponding reference and test R-peak locations (seconds).

Returns

- **TP** (*int*) – Number of true positive R-peaks.
- **FP** (*int*) – Number of false positive R-peaks.
- **performance** (*float*) – Test performance; TP / len(reference).
- **acc** (*float*) – Accuracy rate; TP / (TP + FP).
- **err** (*float*) – Error rate; FP / (TP + FP).
- **match** (*list*) – Indices of the elements of 'test' that match to an R-peak from 'reference'.
- **deviation** (*array*) – Absolute errors of the matched R-peaks (seconds).
- **mean_deviation** (*float*) – Mean error (seconds).
- **std_deviation** (*float*) – Standard deviation of error (seconds).
- **mean_ref_ibi** (*float*) – Mean of the reference interbeat intervals (seconds).
- **std_ref_ibi** (*float*) – Standard deviation of the reference interbeat intervals (seconds).
- **mean_test_ibi** (*float*) – Mean of the test interbeat intervals (seconds).
- **std_test_ibi** (*float*) – Standard deviation of the test interbeat intervals (seconds).

biosppy.signals.ecg.**ecg**(*signal=None*, *sampling_rate=1000.0*, *show=True*)

Process a raw ECG signal and extract relevant signal features using default parameters.

Parameters

- **signal** (*array*) – Raw ECG signal.
- **sampling_rate** (*int, float, optional*) – Sampling frequency (Hz).
- **show** (*bool, optional*) – If True, show a summary plot.

Returns

- **ts** (*array*) – Signal time axis reference (seconds).
- **filtered** (*array*) – Filtered ECG signal.
- **rpeaks** (*array*) – R-peak location indices.
- **templates_ts** (*array*) – Templates time axis reference (seconds).
- **templates** (*array*) – Extracted heartbeat templates.
- **heart_rate_ts** (*array*) – Heart rate time axis reference (seconds).
- **heart_rate** (*array*) – Instantaneous heart rate (bpm).

biosppy.signals.ecg.**engzee_segmenter**(*signal=None*, *sampling_rate=1000.0*, *threshold=0.48*)

ECG R-peak segmentation algorithm.

Follows the approach by Engelse and Zeelenberg *[EnZe79]* with the modifications by Lourenco *et al.* *[LSLL12]*.

Parameters

- **signal** (*array*) – Input filtered ECG signal.
- **sampling_rate** (*int, float, optional*) – Sampling frequency (Hz).
- **threshold** (*float, optional*) – Detection threshold.

Returns **rpeaks** (*array*) – R-peak location indices.

References

biosppy.signals.ecg.**extract_heartbeats**(*signal=None*, *rpeaks=None*, *sampling_rate=1000.0*, *before=0.2*, *after=0.4*)

Extract heartbeat templates from an ECG signal, given a list of R-peak locations.

Parameters

- **signal** (*array*) – Input ECG signal.
- **rpeaks** (*array*) – R-peak location indices.
- **sampling_rate** (*int, float, optional*) – Sampling frequency (Hz).
- **before** (*float, optional*) – Window size to include before the R peak (seconds).
- **after** (*int, optional*) – Window size to include after the R peak (seconds).

Returns

- **templates** (*array*) – Extracted heartbeat templates.
- **rpeaks** (*array*) – Corresponding R-peak location indices of the extracted heartbeat templates.

biosppy.signals.ecg.**gamboa_segmenter**(*signal=None*, *sampling_rate=1000.0*, *tol=0.002*)

ECG R-peak segmentation algorithm.

Follows the approach by Gamboa.

Parameters

- **signal** (*array*) – Input filtered ECG signal.
- **sampling_rate** (*int, float, optional*) – Sampling frequency (Hz).
- **tol** (*float, optional*) – Tolerance parameter.

Returns **rpeaks** (*array*) – R-peak location indices.

biosppy.signals.ecg.**hamilton_segmenter**(*signal=None*, *sampling_rate=1000.0*)

ECG R-peak segmentation algorithm.

Follows the approach by Hamilton *[Hami02]*.

Parameters

- **signal** (*array*) – Input filtered ECG signal.
- **sampling_rate** (*int, float, optional*) – Sampling frequency (Hz).

Returns **rpeaks** (*array*) – R-peak location indices.

References

biosppy.signals.ecg.**ssf_segmenter**(*signal=None*, *sampling_rate=1000.0*, *threshold=20*, *before=0.03*, *after=0.01*)

ECG R-peak segmentation based on the Slope Sum Function (SSF).

Parameters

- **signal** (*array*) – Input filtered ECG signal.
- **sampling_rate** (*int, float, optional*) – Sampling frequency (Hz).
- **threshold** (*float, optional*) – SSF threshold.
- **before** (*float, optional*) – Search window size before R-peak candidate (seconds).
- **after** (*float, optional*) – Search window size after R-peak candidate (seconds).

Returns **rpeaks** (*array*) – R-peak location indices.

biosppy.signals.eda

This module provides methods to process Electrodermal Activity (EDA) signals, also known as Galvanic Skin Response (GSR).

copyright

3. 2015 by Instituto de Telecomunicacoes

license BSD 3-clause, see LICENSE for more details.

biosppy.signals.eda.**basic_scr**(*signal=None*, *sampling_rate=1000.0*)

Basic method to extract Skin Conductivity Responses (SCR) from an EDA signal.

Parameters

- **signal** (*array*) – Input filterd EDA signal.
- **sampling_rate** (*int, float, optional*) – Sampling frequency (Hz).

Returns

- **onsets** (*array*) – Indices of the SCR onsets.
- **peaks** (*array*) – Indices of the SRC peaks.
- **amplitudes** (*array*) – SCR pulse amplitudes.

biosppy.signals.eda.**eda**(*signal=None*, *sampling_rate=1000.0*, *show=True*)

Process a raw EDA signal and extract relevant signal features using default parameters.

Parameters

- **signal** (*array*) – Raw EDA signal.
- **sampling_rate** (*int, float, optional*) – Sampling frequency (Hz).
- **show** (*bool, optional*) – If True, show a summary plot.

Returns

- **ts** (*array*) – Signal time axis reference (seconds).
- **filtered** (*array*) – Filtered EDA signal.
- **onsets** (*array*) – Indices of SCR pulse onsets.

- **peaks** (*array*) – Indices of the SCR peaks.
- **amplitudes** (*array*) – SCR pulse amplitudes.

biosppy.signals.eda.**kbk_scr**(*signal=None*, *sampling_rate=1000.0*)
KBK method to extract Skin Conductivity Responses (SCR) from an EDA signal.

Follows the approach by Kim *et al.* *[KiBK04]*.

Parameters

- **signal** (*array*) – Input filterd EDA signal.
- **sampling_rate** (*int, float, optional*) – Sampling frequency (Hz).

Returns

- **onsets** (*array*) – Indices of the SCR onsets.
- **peaks** (*array*) – Indices of the SRC peaks.
- **amplitudes** (*array*) – SCR pulse amplitudes.

References

biosppy.signals.eeg

This module provides methods to process Electroencephalographic (EEG) signals.

copyright

3. 2015 by Instituto de Telecomunicacoes

license BSD 3-clause, see LICENSE for more details.

biosppy.signals.eeg.**car_reference**(*signal=None*)
Change signal reference to the Common Average Reference (CAR).

Parameters **signal** (*array*) – Input EEG signal matrix; each column is one EEG channel.

Returns **signal** (*array*) – Re-referenced EEG signal matrix; each column is one EEG channel.

biosppy.signals.eeg.**eeg**(*signal=None*, *sampling_rate=1000.0*, *labels=None*, *show=True*)
Process raw EEG signals and extract relevant signal features using default parameters.

Parameters

- **signal** (*array*) – Raw EEG signal matrix; each column is one EEG channel.
- **sampling_rate** (*int, float, optional*) – Sampling frequency (Hz).
- **labels** (*list, optional*) – Channel labels.
- **show** (*bool, optional*) – If True, show a summary plot.

Returns

- **ts** (*array*) – Signal time axis reference (seconds).
- **filtered** (*array*) – Filtered BVP signal.
- **features_ts** (*array*) – Features time axis reference (seconds).
- **theta** (*array*) – Average power in the 4 to 8 Hz frequency band; each column is one EEG channel.

- **alpha_low** (*array*) – Average power in the 8 to 10 Hz frequency band; each column is one EEG channel.
- **alpha_high** (*array*) – Average power in the 10 to 13 Hz frequency band; each column is one EEG channel.
- **beta** (*array*) – Average power in the 13 to 25 Hz frequency band; each column is one EEG channel.
- **gamma** (*array*) – Average power in the 25 to 40 Hz frequency band; each column is one EEG channel.
- **plf_pairs** (*list*) – PLF pair indices.
- **plf** (*array*) – PLF matrix; each column is a channel pair.

biosppy.signals.eeg.**get_plf_features**(*signal=None*, *sampling_rate=1000.0*, *size=0.25*, *overlap=0.5*)

Extract Phase-Locking Factor (PLF) features from EEG signals between all channel pairs.

Parameters

- **signal** (*array*) – Filtered EEG signal matrix; each column is one EEG channel.
- **sampling_rate** (*int, float, optional*) – Sampling frequency (Hz).
- **size** (*float, optional*) – Window size (seconds).
- **overlap** (*float, optional*) – Window overlap (0 to 1).

Returns

- **ts** (*array*) – Features time axis reference (seconds).
- **plf_pairs** (*list*) – PLF pair indices.
- **plf** (*array*) – PLF matrix; each column is a channel pair.

biosppy.signals.eeg.**get_power_features**(*signal=None*, *sampling_rate=1000.0*, *size=0.25*, *overlap=0.5*)

Extract band power features from EEG signals.

Computes the average signal power, with overlapping windows, in typical EEG frequency bands: * Theta: from 4 to 8 Hz, * Lower Alpha: from 8 to 10 Hz, * Higher Alpha: from 10 to 13 Hz, * Beta: from 13 to 25 Hz, * Gamma: from 25 to 40 Hz.

Parameters

- **array** (*signal*) – Filtered EEG signal matrix; each column is one EEG channel.
- **sampling_rate** (*int, float, optional*) – Sampling frequency (Hz).
- **size** (*float, optional*) – Window size (seconds).
- **overlap** (*float, optional*) – Window overlap (0 to 1).

Returns

- **ts** (*array*) – Features time axis reference (seconds).
- **theta** (*array*) – Average power in the 4 to 8 Hz frequency band; each column is one EEG channel.
- **alpha_low** (*array*) – Average power in the 8 to 10 Hz frequency band; each column is one EEG channel.
- **alpha_high** (*array*) – Average power in the 10 to 13 Hz frequency band; each column is one EEG channel.

- **beta** (*array*) – Average power in the 13 to 25 Hz frequency band; each column is one EEG channel.
- **gamma** (*array*) – Average power in the 25 to 40 Hz frequency band; each column is one EEG channel.

biosppy.signals.emg

This module provides methods to process Electromyographic (EMG) signals.

copyright

3. 2015 by Instituto de Telecomunicacoes

license BSD 3-clause, see LICENSE for more details.

biosppy.signals.emg.**emg**(*signal=None*, *sampling_rate=1000.0*, *show=True*)
Process a raw EMG signal and extract relevant signal features using default parameters.

Parameters

- **signal** (*array*) – Raw EMG signal.
- **sampling_rate** (*int, float, optional*) – Sampling frequency (Hz).
- **show** (*bool, optional*) – If True, show a summary plot.

Returns

- **ts** (*array*) – Signal time axis reference (seconds).
- **filtered** (*array*) – Filtered EMG signal.
- **onsets** (*array*) – Indices of EMG pulse onsets.

biosppy.signals.emg.**find_onsets**(*signal=None*, *sampling_rate=1000.0*, *size=0.05*, *threshold=None*)
Determine onsets of EMG pulses.

Skips corrupted signal parts.

Parameters

- **signal** (*array*) – Input filtered BVP signal.
- **sampling_rate** (*int, float, optional*) – Sampling frequency (Hz).
- **size** (*float, optional*) – Detection window size (seconds).
- **threshold** (*float, optional*) – Detection threshold.

Returns **onsets** (*array*) – Indices of BVP pulse onsets.

biosppy.signals.resp

This module provides methods to process Respiration (Resp) signals.

copyright

3. 2015 by Instituto de Telecomunicacoes

license BSD 3-clause, see LICENSE for more details.

biosppy.signals.resp.**resp**(*signal=None*, *sampling_rate=1000.0*, *show=True*)
Process a raw Respiration signal and extract relevant signal features using default parameters.

Parameters

- **signal** (*array*) – Raw Respiration signal.
- **sampling_rate** (*int, float, optional*) – Sampling frequency (Hz).
- **show** (*bool, optional*) – If True, show a summary plot.

Returns

- **ts** (*array*) – Signal time axis reference (seconds).
- **filtered** (*array*) – Filtered Respiration signal.
- **zeros** (*array*) – Indices of Respiration zero crossings.
- **resp_rate_ts** (*array*) – Respiration rate time axis reference (seconds).
- **resp_rate** (*array*) – Instantaneous respiration rate (Hz).

biosppy.signals.tools

This module provides various signal analysis methods in the time and frequency domains.

copyright

3. 2015 by Instituto de Telecomunicacoes

license BSD 3-clause, see LICENSE for more details.

biosppy.signals.tools.**analytic_signal**(*signal=None*, *N=None*)
Compute analytic signal, using the Hilbert Transform.

Parameters

- **signal** (*array*) – Input signal.
- **N** (*int, optional*) – Number of Fourier components; default is *len(signal)*.

Returns

- **amplitude** (*array*) – Amplitude envelope of the analytic signal.
- **phase** (*array*) – Instantaneous phase component of the analystic signal.

biosppy.signals.tools.**band_power**(*freqs=None*, *power=None*, *frequency=None*, *decibel=True*)
Compute the avearge power in a frequency band.

Parameters

- **freqs** (*array*) – Array of frequencies (Hz) at which the power was computed.
- **power** (*array*) – Input power spectrum.
- **frequency** (*list, array*) – Pair of frequencies defining the band.
- **decibel** (*bool, optional*) – If True, input power is in decibels.

Returns **avg_power** (*float*) – The average power in the band.

biosppy.signals.tools.**filter_signal**(*signal=None*, *ftype='FIR'*, *band='lowpass'*, *order=None*, *frequency=None*, *sampling_rate=1000.0*, ***kwargs*)
Filter a signal according to the given parameters.

Parameters

- **signal** (*array*) – Signal to filter.

- **ftype** (*str*) – Filter type: * Finite Impulse Response filter ('FIR'); * Butterworth filter ('butter'); * Chebyshev filters ('cheby1', 'cheby2'); * Elliptic filter ('ellip'); * Bessel filter ('bessel').
- **band** (*str*) – Band type: * Low-pass filter ('lowpass'); * High-pass filter ('highpass'); * Band-pass filter ('bandpass'); * Band-stop filter ('bandstop').
- **order** (*int*) – Order of the filter.
- **frequency** (*int, float, list, array*) – Cutoff frequencies; format depends on type of band: * 'lowpass' or 'bandpass': single frequency; * 'bandpass' or 'bandstop': pair of frequencies.
- **sampling_rate** (*int, float, optional*) – Sampling frequency (Hz).
- ****kwargs** (*dict, optional*) – Additional keyword arguments are passed to the underlying scipy.signal function.

Returns

- **signal** (*array*) – Filtered signal.
- **sampling_rate** (*float*) – Sampling frequency (Hz).
- **params** (*dict*) – Filter parameters.

Notes

•Uses a forward-backward filter implementation. Therefore, the combined filter has linear phase.

biosppy.signals.tools.**find_extrema**(*signal=None*, *mode='both'*)

Locate local extrema points in a signal.

Based on Fermat's Theorem *[Ferm]*.

Parameters

- **signal** (*array*) – Input signal.
- **mode** (*str, optional*) – Whether to find maxima ('max'), minima ('min'), or both ('both').

Returns

- **extrema** (*array*) – Indices of the extrama points.
- **values** (*array*) – Signal values at the extrema points.

References

biosppy.signals.tools.**find_intersection**(*x1=None*, *y1=None*, *x2=None*, *y2=None*, *alpha=1.5*, *xtol=1e-06*, *ytol=1e-06*)

Find the intersection points between two lines using piecewise polynomial interpolation.

Parameters

- **x1** (*array*) – Array of x-coordinates of the first line.
- **y1** (*array*) – Array of y-coordinates of the first line.
- **x2** (*array*) – Array of x-coordinates of the second line.
- **y2** (*array*) – Array of y-coordinates of the second line.

- **alpha** (*float, optional*) – Resolution factor for the x-axis; fraction of total number of x-coordinates.
- **xtol** (*float, optional*) – Tolerance for the x-axis.
- **ytol** (*float, optional*) – Tolerance for the y-axis.

Returns

- **roots** (*array*) – Array of x-coordinates of found intersection points.
- **values** (*array*) – Array of y-coordinates of found intersection points.

Notes

•If no intersection is found, returns the closest point.

biosppy.signals.tools.**get_filter**(*ftype='FIR'*, *band='lowpass'*, *order=None*, *frequency=None*, *sampling_rate=1000.0*, ***kwargs*)

Compute digital (FIR or IIR) filter coefficients with the given parameters.

Parameters

- **ftype** (*str*) – Filter type: * Finite Impulse Response filter ('FIR'); * Butterworth filter ('butter'); * Chebyshev filters ('cheby1', 'cheby2'); * Elliptic filter ('ellip'); * Bessel filter ('bessel').
- **band** (*str*) – Band type: * Low-pass filter ('lowpass'); * High-pass filter ('highpass'); * Band-pass filter ('bandpass'); * Band-stop filter ('bandstop').
- **order** (*int*) – Order of the filter.
- **frequency** (*int, float, list, array*) – Cutoff frequencies; format depends on type of band: * 'lowpass' or 'bandpass': single frequency; * 'bandpass' or 'bandstop': pair of frequencies.
- **sampling_rate** (*int, float, optional*) – Sampling frequency (Hz).
- ****kwargs** (*dict, optional*) – Additional keyword arguments are passed to the underlying scipy.signal function.

Returns

- **b** (*array*) – Numerator coefficients.
- **a** (*array*) – Denominator coefficients.
- *See Also* – scipy.signal

biosppy.signals.tools.**get_heart_rate**(*beats=None*, *sampling_rate=1000.0*, *smooth=False*, *size=3*)

Compute instantaneous heart rate from an array of beat indices.

Parameters

- **beats** (*array*) – Beat location indices.
- **sampling_rate** (*int, float, optional*) – Sampling frequency (Hz).
- **smooth** (*bool, optional*) – If True, perform smoothing on the resulting heart rate.
- **size** (*int, optional*) – Size of smoothing window; ignored if *smooth* is False.

Returns

- **index** (*array*) – Heart rate location indices.
- **heart_rate** (*array*) – Instantaneous heart rate (bpm).

Notes

•Assumes normal human heart rate to be between 40 and 190 bpm.

biosppy.signals.tools.**normalize**(*signal=None*)
Normalize a signal.

Parameters **signal** (*array*) – Input signal.

Returns **signal** (*array*) – Normalized signal.

biosppy.signals.tools.**phase_locking**(*signal1=None*, *signal2=None*, *N=None*)
Compute the Phase-Locking Factor (PLF) between two signals.

Parameters

- **signal1** (*array*) – First input signal.
- **signal2** (*array*) – Second input signal.
- **N** (*int, optional*) – Number of Fourier components.

Returns **plf** (*float*) – The PLF between the two signals.

biosppy.signals.tools.**power_spectrum**(*signal=None*, *sampling_rate=1000.0*, *pad=None*, *pow2=False*, *decibel=True*)
Compute the power spectrum of a signal (one-sided).

Parameters

- **signal** (*array*) – Input signal.
- **sampling_rate** (*int, float, optional*) – Sampling frequency (Hz).
- **pad** (*int, optional*) – Padding for the Fourier Transform (number of zeros added).
- **pow2** (*bool, optional*) – If True, rounds the number of points *N = len(signal) + pad* to the nearest power of 2 greater than N.
- **decibel** (*bool, optional*) – If True, returns the power in decibels.

Returns

- **freqs** (*array*) – Array of frequencies (Hz) at which the power was computed.
- **power** (*array*) – Power spectrum.

biosppy.signals.tools.**signal_stats**(*signal=None*)
Compute various metrics describing the signal.

Parameters **signal** (*array*) – Input signal.

Returns

- **mean** (*float*) – Mean of the signal.
- **median** (*float*) – Median of the signal.
- **max** (*float*) – Maximum signal amplitude.
- **var** (*float*) – Signal variance (unbiased).
- **std_dev** (*float*) – Standard signal deviation (unbiased).

- **abs_dev** (*float*) – Absolute signal deviation.
- **kurtosis** (*float*) – Signal kurtosis (unbiased).
- **skew** (*float*) – Signal skewness (unbiased).

biosppy.signals.tools.**smoother**(*signal=None*, *kernel='boxzen'*, *size=10*, *mirror=True*, ***kwargs*)

Smooth a signal using an N-point moving average *[MAvg]* filter.

This implementation uses the convolution of a filter kernel with the input signal to compute the smoothed signal *[Smit97]*.

Availabel kernels: median, boxzen, boxcar, triang, blackman, hamming, hann, bartlett, flattop, parzen, bohman, blackmanharris, nuttall, barthann, kaiser (needs beta), gaussian (needs std), general_gaussian (needs power, width), slepian (needs width), chebwin (needs attenuation).

Parameters

- **signal** (*array*) – Signal to smooth.
- **kernel** (*str, array, optional*) – Type of kernel to use; if array, use directly as the kernel.
- **size** (*int, optional*) – Size of the kernel; ignored if kernel is an array.
- **mirror** (*bool, optional*) – If True, signal edges are extended to avoid boundary effects.
- ****kwargs** (*dict, optional*) – Additional keyword arguments are passed to the underlying scipy.signal.windows function.

Returns

- **signal** (*array*) – Smoothed signal.
- **params** (*dict*) – Smoother parameters.

Notes

•When the kernel is 'median', mirror is ignored.

References

biosppy.signals.tools.**synchronize**(*signal1=None*, *signal2=None*)

Align two signals based on cross-correlation.

Parameters

- **signal1** (*array*) – First input signal.
- **signal2** (*array*) – Second input signal.

Returns

- **delay** (*int*) – Delay (number of samples) of 'signal1' in relation to 'signal2'; if 'delay' < 0 , 'signal1' is ahead in relation to 'signal2'; if 'delay' > 0 , 'signal1' is delayed in relation to 'signal2'.
- **corr** (*float*) – Value of maximum correlation.
- **synch1** (*array*) – Biggest possible portion of 'signal1' in synchronization.

- **synch2** (*array*) – Biggest possible portion of 'signal2' in synchronization.

biosppy.signals.tools.**windower**(*signal=None*, *size=None*, *step=None*, *fcn=None*, *fcn_kwargs=None*, *kernel='boxcar'*, *kernel_kwargs=None*)

Apply a function to a signal in sequential windows, with optional overlap.

Availabel window kernels: boxcar, triang, blackman, hamming, hann, bartlett, flattop, parzen, bohman, blackmanharris, nuttall, barthann, kaiser (needs beta), gaussian (needs std), general_gaussian (needs power, width), slepian (needs width), chebwin (needs attenuation).

Parameters

- **signal** (*array*) – Input signal.
- **size** (*int*) – Size of the signal window.
- **step** (*int, optional*) – Size of window shift; if None, there is no overlap.
- **fcn** (*callable*) – Function to apply to each window.
- **fcn_kwargs** (*dict, optional*) – Additional keyword arguments to pass to 'fcn'.
- **kernel** (*str, array, optional*) – Type of kernel to use; if array, use directly as the kernel.
- **kernel_kwargs** (*dict, optional*) – Additional keyword arguments to pass on window creation; ignored if 'kernel' is an array.

Returns

- **index** (*array*) – Indices characterizing window locations (start of the window).
- **values** (*array*) – Concatenated output of calling 'fcn' on each window.

biosppy.signals.tools.**zero_cross**(*signal=None*, *detrend=False*)

Locate the indices where the signal crosses zero.

Parameters

- **signal** (*array*) – Input signal.
- **detrend** (*bool, optional*) – If True, remove signal mean before computation.

Returns **zeros** (*array*) – Indices of zero crossings.

Notes

•When the signal crosses zero between samples, the first index is returned.

2.2 Modules

2.2.1 biosppy.biometrics

This module provides classifier interfaces for identity recognition (biometrics) applications. The core API methods are: * enroll: add a new subject; * dismiss: remove an existing subject; * identify: determine the identity of collected biometric dataset; * authenticate: verify the identity of collected biometric dataset.

copyright

3. 2015 by Instituto de Telecomunicacoes

license BSD 3-clause, see LICENSE for more details.

class biosppy.biometrics.**BaseClassifier**
Bases: object

Base biometric classifier class.

This class is a skeleton for actual classifier classes. The following methods must be overridden or adapted to build a new classifier:

- __init__
- _authenticate
- _get_thresholds
- _identify
- _prepare
- _train
- _update

EER_IDX
int

Reference index for the Equal Error Rate.

EER_IDX = 0

authenticate(*data*, *subject*, *threshold=None*)
Authenticate a set of feature vectors, allegedly belonging to the given subject.

Parameters

- **data** (*array*) – Input test data.
- **subject** (*hashable*) – Subject identity.
- **threshold** (*int, float, optional*) – Authentication threshold.

Returns **decision** (*array*) – Authentication decision for each input sample.

batch_train(*data=None*)
Train the classifier in batch mode.

Parameters **data** (*dict*) – Dictionary holding training data for each subject; if the object for a subject is *None*, performs a *dismiss*.

check_subject(*subject*)
Check if a subject is enrolled.

Parameters **subject** (*hashable*) – Subject identity.

Returns **check** (*bool*) – If True, the subject is enrolled.

classmethod **cross_validation**(*data*, *labels*, *cv*, *thresholds=None*, ***kwargs*)
Perform Cross Validation (CV) on a data set.

Parameters

- **data** (*array*) – An m by n array of m data samples in an n-dimensional space.
- **labels** (*list, array*) – A list of m class labels.
- **cv** (*CV iterator*) – A *sklearn.cross_validation* iterator.
- **thresholds** (*array, optional*) – Classifier thresholds to use.
- ****kwargs** (*dict, optional*) – Classifier parameters.

Returns

- **runs** (*list*) – Evaluation results for each CV run.
- **assessment** (*dict*) – Final CV biometric statistics.

dismiss(*subject=None*, *deferred=False*)
Remove a subject.

Parameters

- **subject** (*hashable*) – Subject identity.
- **deferred** (*bool, optional*) – If True, computations are delayed until *flush* is called.

Notes

•When using deferred calls, a dismiss overrides a previous enroll for the same subject.

enroll(*data=None*, *subject=None*, *deferred=False*)
Enroll new data for a subject.

If the subject is already enrolled, new data is combined with existing data.

Parameters

- **data** (*array*) – Data to enroll.
- **subject** (*hashable*) – Subject identity.
- **deferred** (*bool, optional*) – If True, computations are delayed until *flush* is called.

Notes

•When using deferred calls, an enroll overrides a previous dismiss for the same subject.

evaluate(*data*, *thresholds=None*, *show=False*)
Assess the performance of the classifier in both authentication and identification scenarios.

Parameters

- **data** (*dict*) – Dictionary holding test data for each subject.
- **thresholds** (*array, optional*) – Classifier thresholds to use.
- **show** (*bool, optional*) – If True, show a summary plot.

Returns

- **classification** (*dict*) – Classification results.
- **assessment** (*dict*) – Biometric statistics.

flush()
Flush deferred computations.

get_auth_thr(*subject*, *ready=False*)
Get the authentication threshold of a subject.

Parameters

- **subject** (*hashable*) – Subject identity.
- **ready** (*bool, optional*) – If True, *subject* is the internal classifier label.

Returns **threshold** (*int, float*) – Threshold value.

get_id_thr(*subject*, *ready=False*)
Get the identification threshold of a subject.

Parameters

- **subject** (*hashable*) – Subject identity.
- **ready** (*bool, optional*) – If True, *subject* is the internal classifier label.

Returns **threshold** (*int, float*) – Threshold value.

get_thresholds(*force=False*)
Get an array of reasonable thresholds.

Parameters **force** (*bool, optional*) – If True, forces generation of thresholds.

Returns **ths** (*array*) – Generated thresholds.

identify(*data*, *threshold=None*)
Identify a set of feature vectors.

Parameters

- **data** (*array*) – Input test data.
- **threshold** (*int, float, optional*) – Identification threshold.

Returns **subjects** (*list*) – Identity of each input sample.

io_del(*label*)
Delete subject data.

Parameters **label** (*str*) – Internal classifier subject label.

io_load(*label*)
Load enrolled subject data.

Parameters **label** (*str*) – Internal classifier subject label.

Returns **data** (*array*) – Subject data.

io_save(*label*, *data*)
Save subject data.

Parameters

- **label** (*str*) – Internal classifier subject label.
- **data** (*array*) – Subject data.

list_subjects()
List all the enrolled subjects.

Returns **subjects** (*list*) – Enrolled subjects.

classmethod load(*path*)
Load classifier instance from a file.

Parameters **path** (*str*) – Source file path.

Returns **clf** (*object*) – Loaded classifier instance.

save(*path*)
Save classifier instance to a file.

Parameters **path** (*str*) – Destination file path.

set_auth_thr(*subject*, *threshold*, *ready=False*)
Set the authentication threshold of a subject.

Parameters

- **subject** (*hashable*) – Subject identity.
- **threshold** (*int, float*) – Threshold value.
- **ready** (*bool, optional*) – If True, *subject* is the internal classifier label.

set_id_thr(*subject*, *threshold*, *ready=False*)
Set the identification threshold of a subject.

Parameters

- **subject** (*hashable*) – Subject identity.
- **threshold** (*int, float*) – Threshold value.
- **ready** (*bool, optional*) – If True, *subject* is the internal classifier label.

update_thresholds(*fraction=1.0*)
Update subject-specific thresholds based on the enrolled data.

Parameters **fraction** (*float, optional*) – Fraction of samples to select from training data.

exception biosppy.biometrics.**CombinationError**
Bases: exceptions.Exception

Exception raised when the combination method fails.

class biosppy.biometrics.**KNN**(*k=3*, *metric='euclidean'*, *metric_args=None*)
Bases: *biosppy.biometrics.BaseClassifier*

K Nearest Neighbors (k-NN) biometric classifier.

Parameters

- **k** (*int, optional*) – Number of neighbors.
- **metric** (*str, optional*) – Distance metric.
- **metric_args** (*dict, optional*) – Additional keyword arguments are passed to the distance function.

EER_IDX
int

Reference index for the Equal Error Rate.

EER_IDX = 0

class biosppy.biometrics.**SVM**(*C=1.0*, *kernel='linear'*, *degree=3*, *gamma=0.0*, *coef0=0.0*, *shrinking=True*, *tol=0.001*, *cache_size=200*, *max_iter=-1*, *random_state=None*)

Bases: *biosppy.biometrics.BaseClassifier*

Support Vector Machines (SVM) biometric classifier.

Wraps the 'OneClassSVM' and 'SVC' classes from 'scikit-learn'.

Parameters

- **C** (*float, optional*) – Penalty parameter C of the error term.
- **kernel** (*str, optional*) – Specifies the kernel type to be used in the algorithm. It must be one of 'linear', 'poly', 'rbf', 'sigmoid', 'precomputed' or a callable. If none is given, 'rbf' will be used. If a callable is given it is used to precompute the kernel matrix.
- **degree** (*int, optional*) – Degree of the polynomial kernel function ('poly'). Ignored by all other kernels.
- **gamma** (*float, optional*) – Kernel coefficient for 'rbf', 'poly' and 'sigmoid'. If gamma is 0.0 then 1/n_features will be used instead.
- **coef0** (*float, optional*) – Independent term in kernel function. It is only significant in 'poly' and 'sigmoid'.
- **shrinking** (*bool, optional*) – Whether to use the shrinking heuristic.
- **tol** (*float, optional*) – Tolerance for stopping criterion.
- **cache_size** (*float, optional*) – Specify the size of the kernel cache (in MB).
- **max_iter** (*int, optional*) – Hard limit on iterations within solver, or -1 for no limit.
- **random_state** (*int, RandomState, optional*) – The seed of the pseudo random number generator to use when shuffling the data for probability estimation.

EER_IDX

int

Reference index for the Equal Error Rate.

EER_IDX = -1

exception biosppy.biometrics.**SubjectError**(*subject=None*)

Bases: exceptions.Exception

Exception raised when the subject is unknown.

exception biosppy.biometrics.**UntrainedError**

Bases: exceptions.Exception

Exception raised when classifier is not trained.

biosppy.biometrics.**assess_classification**(*results=None*, *thresholds=None*)

Assess the performance of a biometric classification test.

Parameters

- **results** (*dict*) – Classification results.
- **thresholds** (*array*) – Classifier thresholds.

Returns **assessment** (*dict*) – Classification assessment.

biosppy.biometrics.**assess_runs**(*results=None*, *subjects=None*)
Assess the performance of multiple biometric classification runs.

Parameters

- **results** (*list*) – Classification results for each run.
- **subjects** (*list*) – Common target subject classes.

Returns **assessment** (*dict*) – Global classification assessment.

biosppy.biometrics.**combination**(*results=None*, *weights=None*)
Combine results from multiple classifiers.

Parameters

- **results** (*dict*) – Results for each classifier.
- **weights** (*dict, optional*) – Weight for each classifier.

Returns

- **decision** (*object*) – Consensus decision.
- **confidence** (*float*) – Confidence estimate of the decision.
- **counts** (*array*) – Weight for each possible decision outcome.
- **classes** (*array*) – List of possible decision outcomes.

biosppy.biometrics.**cross_validation**(*labels*, *n_iter=10*, *test_size=0.1*, *train_size=None*, *random_state=None*)
Return a Cross Validation (CV) iterator.

Wraps the StratifiedShuffleSplit iterator from sklearn.cross_validation. This iterator returns stratified randomized folds, which preserve the percentage of samples for each class.

Parameters

- **labels** (*list, array*) – List of class labels for each data sample.
- **n_iter** (*int, optional*) – Number of splitting iterations.
- **test_size** (*float, int, optional*) – If float, represents the proportion of the dataset to include in the test split; if int, represents the absolute number of test samples.
- **train_size** (*float, int, optional*) – If float, represents the proportion of the dataset to include in the train split; if int, represents the absolute number of train samples.
- **random_state** (*int, RandomState, optional*) – The seed of the pseudo random number generator to use when shuffling the data.

Returns **cv** (*CV iterator*) – Cross Validation iterator.

biosppy.biometrics.**get_auth_rates**(*TP=None*, *FP=None*, *TN=None*, *FN=None*, *thresholds=None*)
Compute authentication rates from the confusion matrix.

Parameters

- **TP** (*array*) – True Positive counts for each classifier threshold.
- **FP** (*array*) – False Positive counts for each classifier threshold.
- **TN** (*array*) – True Negative counts for each classifier threshold.
- **FN** (*array*) – False Negative counts for each classifier threshold.
- **thresholds** (*array*) – Classifier thresholds.

Returns

- **Acc** (*array*) – Accuracy at each classifier threshold.
- **TAR** (*array*) – True Accept Rate at each classifier threshold.
- **FAR** (*array*) – False Accept Rate at each classifier threshold.
- **FRR** (*array*) – False Reject Rate at each classifier threshold.
- **TRR** (*array*) – True Reject Rate at each classifier threshold.
- **EER** (*array*) – Equal Error Rate points, with format (threshold, rate).

biosppy.biometrics.**get_id_rates**(*H=None*, *M=None*, *R=None*, *N=None*, *thresholds=None*)
Compute identification rates from the confusion matrix.

Parameters

- **H** (*array*) – Hit counts for each classifier threshold.
- **M** (*array*) – Miss counts for each classifier threshold.
- **R** (*array*) – Reject counts for each classifier threshold.
- **N** (*int*) – Number of test samples.
- **thresholds** (*array*) – Classifier thresholds.

Returns

- **Acc** (*array*) – Accuracy at each classifier threshold.
- **Err** (*array*) – Error rate at each classifier threshold.
- **MR** (*array*) – Miss Rate at each classifier threshold.
- **RR** (*array*) – Reject Rate at each classifier threshold.
- **EID** (*array*) – Error of Identification points, with format (threshold, rate).
- **EER** (*array*) – Equal Error Rate points, with format (threshold, rate).

biosppy.biometrics.**get_subject_results**(*results=None*, *subject=None*, *thresholds=None*, *subjects=None*, *subject_dict=None*, *subject_idx=None*)
Compute authentication and identification performance metrics for a given subject.

Parameters

- **results** (*dict*) – Classification results.
- **subject** (*hashable*) – True subject label.
- **thresholds** (*array*) – Classifier thresholds.
- **subjects** (*list*) – Target subject classes.
- **subject_dict** (*SubjectDict*) – Subject-label conversion dictionary.
- **subject_idx** (*list*) – Subject index.

Returns **assessment** (*dict*) – Authentication and identification results.

biosppy.biometrics.**majority_rule**(*labels=None*, *random=True*)
Determine the most frequent class label.

Parameters

- **labels** (*array, list*) – List of clas labels.

- **random** (*bool, optional*) – If True, will choose randomly in case of tied classes, otherwise the first element is chosen.

Returns

- **decision** (*object*) – Consensus decision.
- **count** (*int*) – Number of elements of the consensus decision.

2.2.2 biosppy.clustering

This module provides various unsupervised machine learning (clustering) algorithms.

copyright

3. 2015 by Instituto de Telecomunicacoes

license BSD 3-clause, see LICENSE for more details.

biosppy.clustering.**centroid_templates**(*data=None*, *clusters=None*, *ntemplates=1*)
Template selection based on cluster centroids.

Parameters

- **data** (*array*) – An m by n array of m data samples in an n-dimensional space.
- **clusters** (*dict*) – Dictionary with the sample indices (rows from 'data') for each cluster.
- **ntemplates** (*int, optional*) – Number of templates to extract; if more than 1, k-means is used to obtain more templates.

Returns **templates** (*array*) – Selected templates from the input data.

biosppy.clustering.**coassoc_partition**(*coassoc=None*, *k=0*, *linkage='average'*)
Extract the consensus partition from a co-association matrix using hierarchical agglomerative methods.

Parameters

- **coassoc** (*array*) – Co-association matrix.
- **k** (*int, optional*) – Number of clusters to extract; if 0 uses the life-time criterion.
- **linkage** (*str, optional*) – Linkage criterion for final partition extraction; one of 'average', 'centroid', 'complete', 'median', 'single', 'ward', or 'weighted'.

Returns **clusters** (*dict*) – Dictionary with the sample indices (rows from 'data') for each found cluster; outliers have key -1; clusters are assigned integer keys starting at 0.

biosppy.clustering.**consensus**(*data=None*, *k=0*, *linkage='average'*, *fcn=None*, *grid=None*)
Perform clustering based in an ensemble of partitions.

Parameters

- **data** (*array*) – An m by n array of m data samples in an n-dimensional space.
- **k** (*int, optional*) – Number of clusters to extract; if 0 uses the life-time criterion.
- **linkage** (*str, optional*) – Linkage criterion for final partition extraction; one of 'average', 'centroid', 'complete', 'median', 'single', 'ward', or 'weighted'.
- **fcn** (*function*) – A clustering function.
- **grid** (*dict, list, optional*) – A (list of) dictionary with parameters for each run of the clustering method (see sklearn.grid_search.ParameterGrid).

Returns **clusters** (*dict*) – Dictionary with the sample indices (rows from 'data') for each found cluster; outliers have key -1; clusters are assigned integer keys starting at 0.

biosppy.clustering.**consensus_kmeans**(*data=None*, *k=0*, *linkage='average'*, *nensemble=100*, *kmin=None*, *kmax=None*)

Perform clustering based on an ensemble of k-means partitions.

Parameters

- **data** (*array*) – An m by n array of m data samples in an n-dimensional space.
- **k** (*int, optional*) – Number of clusters to extract; if 0 uses the life-time criterion.
- **linkage** (*str, optional*) – Linkage criterion for final partition extraction; one of 'average', 'centroid', 'complete', 'median', 'single', 'ward', or 'weighted'.
- **nensemble** (*int, optional*) – Number of partitions in the ensemble.
- **kmin** (*int, optional*) – Minimum k for the k-means partitions; defaults to $\sqrt{m}/2$.
- **kmax** (*int, optional*) – Maximum k for the k-means partitions; defaults to $\sqrt{m}$.

Returns **clusters** (*dict*) – Dictionary with the sample indices (rows from 'data') for each found cluster; outliers have key -1; clusters are assigned integer keys starting at 0.

biosppy.clustering.**create_coassoc**(*ensemble=None*, *N=None*)

Create the co-association matrix from a clustering ensemble.

Parameters

- **ensemble** (*list*) – Clustering ensemble partitions.
- **N** (*int*) – Number of data samples.

Returns **coassoc** (*array*) – Co-association matrix.

biosppy.clustering.**create_ensemble**(*data=None*, *fcn=None*, *grid=None*)

Create an ensemble of partitions of the data using the given clustering method.

Parameters

- **data** (*array*) – An m by n array of m data samples in an n-dimensional space.
- **fcn** (*function*) – A clustering function.
- **grid** (*dict, list, optional*) – A (list of) dictionary with parameters for each run of the clustering method (see sklearn.grid_search.ParameterGrid).

Returns **ensemble** (*list*) – Obtained ensemble partitions.

biosppy.clustering.**dbscan**(*data=None*, *min_samples=5*, *eps=0.5*, *metric='euclidean'*, *metric_args=None*)

Perform clustering using the DBSCAN algorithm *[EKSX96]*.

The algorithm works by grouping data points that are closely packed together (with many nearby neighbors), marking as outliers points that lie in low-density regions.

Parameters

- **data** (*array*) – An m by n array of m data samples in an n-dimensional space.
- **min_samples** (*int, optional*) – Minimum number of samples in a cluster.
- **eps** (*float, optional*) – Maximum distance between two samples in the same cluster.
- **metric** (*str, optional*) – Distance metric (see scipy.spatial.distance).

- **metric_args** (*dict, optional*) – Additional keyword arguments to pass to the distance function.

Returns clusters (*dict*) – Dictionary with the sample indices (rows from 'data') for each found cluster; outliers have key -1; clusters are assigned integer keys starting at 0.

References

biosppy.clustering.**hierarchical**(*data=None*, *k=0*, *linkage='average'*, *metric='euclidean'*, *metric_args=None*)

Perform clustering using hierarchical agglomerative algorithms.

Parameters

- **data** (*array*) – An m by n array of m data samples in an n-dimensional space.
- **k** (*int, optional*) – Number of clusters to extract; if 0 uses the life-time criterion.
- **linkage** (*str, optional*) – Linkage criterion; one of 'average', 'centroid', 'complete', 'median', 'single', 'ward', or 'weighted'.
- **metric** (*str, optional*) – Distance metric (see scipy.spatial.distance).
- **metric_args** (*dict, optional*) – Additional keyword arguments to pass to the distance function.

Returns clusters (*dict*) – Dictionary with the sample indices (rows from 'data') for each found cluster; outliers have key -1; clusters are assigned integer keys starting at 0.

biosppy.clustering.**kmeans**(*data=None*, *k=None*, *init='random'*, *max_iter=300*, *n_init=10*, *tol=0.0001*)

Perform clustering using the k-means algorithm.

Parameters

- **data** (*array*) – An m by n array of m data samples in an n-dimensional space.
- **k** (*int*) – Number of clusters to extract.
- **init** (*str, array, optional*) – If string, one of 'random' or 'k-means++'; if array, it should be of shape (n_clusters, n_features), specifying the initial centers.
- **max_iter** (*int, optional*) – Maximum number of iterations.
- **n_init** (*int, optional*) – Number of initializations.
- **tol** (*float, optional*) – Relative tolerance to declare convergence.

Returns clusters (*dict*) – Dictionary with the sample indices (rows from 'data') for each found cluster; outliers have key -1; clusters are assigned integer keys starting at 0.

biosppy.clustering.**mdist_templates**(*data=None*, *clusters=None*, *ntemplates=1*, *metric='euclidean'*, *metric_args=None*)

Template selection based on the MDIST method *[UlRJ04]*.

Extends the original method with the option of also providing a data clustering, in which case the MDIST criterion is applied for each cluster *[LCSF14]*.

Parameters

- **data** (*array*) – An m by n array of m data samples in an n-dimensional space.
- **clusters** (*dict, optional*) – Dictionary with the sample indices (rows from *data*) for each cluster.

- **ntemplates** (*int, optional*) – Number of templates to extract.
- **metric** (*str, optional*) – Distance metric (see scipy.spatial.distance).
- **metric_args** (*dict, optional*) – Additional keyword arguments to pass to the distance function.

Returns **templates** (*array*) – Selected templates from the input data.

References

biosppy.clustering.**outliers_dbscan**(*data=None*, *min_samples=5*, *eps=0.5*, *metric='euclidean'*, *metric_args=None*)

Perform outlier removal using the DBSCAN algorithm.

Parameters

- **data** (*array*) – An m by n array of m data samples in an n-dimensional space.
- **min_samples** (*int, optional*) – Minimum number of samples in a cluster.
- **eps** (*float, optional*) – Maximum distance between two samples in the same cluster.
- **metric** (*str, optional*) – Distance metric (see scipy.spatial.distance).
- **metric_args** (*dict, optional*) – Additional keyword arguments to pass to the distance function.

Returns

- **clusters** (*dict*) – Dictionary with the sample indices (rows from 'data') for the outliers (key -1) and the normal (key 0) groups.
- **templates** (*dict*) – Elements from 'data' for the outliers (key -1) and the normal (key 0) groups.

biosppy.clustering.**outliers_dmean**(*data=None*, *alpha=0.5*, *beta=1.5*, *metric='euclidean'*, *metric_args=None*, *max_idx=None*)

Perform outlier removal using the DMEAN algorithm *[LCSF13]*.

A sample is considered valid if it cumulatively verifies:

- distance to average template smaller than a (data derived) threshold 'T';
- sample minimum greater than a (data derived) threshold 'M';
- sample maximum smaller than a (data derived) threshold 'N';
- position of the sample maximum is the same as the given index [optional].

For a set of $\{X_1, ..., X_n\}$ n samples:

$$\widetilde{X} = \frac{1}{n} \sum_{i=1}^{n} X_i$$

$$d_i = dist(X_i, \widetilde{X})$$

$$D_m = \frac{1}{n} \sum_{i=1}^{n} d_i$$

$$D_s = \sqrt{\frac{1}{n-1} \sum_{i=1}^{n} (d_i - D_m)^2}$$

$$T = D_m + \alpha * D_s$$

$$M = \beta * median(\{\max X_i, i = 1, ..., n\})$$

$$N = \beta * median(\{\min X_i, i = 1, ..., n\})$$

Parameters

- **data** (*array*) – An m by n array of m data samples in an n-dimensional space.
- **alpha** (*float, optional*) – Parameter for the distance threshold.
- **beta** (*float, optional*) – Parameter for the maximum and minimum thresholds.
- **metric** (*str, optional*) – Distance metric (see scipy.spatial.distance).
- **metric_args** (*dict, optional*) – Additional keyword arguments to pass to the distance function.
- **max_idx** (*int, optional*) – Index of the expected maximum.

Returns

- **clusters** (*dict*) – Dictionary with the sample indices (rows from 'data') for the outliers (key -1) and the normal (key 0) groups.
- **templates** (*dict*) – Elements from 'data' for the outliers (key -1) and the normal (key 0) groups.

References

2.2.3 biosppy.metrics

This module provides pairwise distance computation methods.

copyright

3. 2015 by Instituto de Telecomunicacoes

license BSD 3-clause, see LICENSE for more details.

biosppy.metrics.**cdist**(*XA*, *XB*, *metric='euclidean'*, *p=2*, *V=None*, *VI=None*, *w=None*)

Computes distance between each pair of the two collections of inputs.

Wraps scipy.spatial.distance.cdist.

Parameters

- **XA** (*array*) – An m_A by n array of m_A original observations in an n-dimensional space.
- **XB** (*array*) – An m_B by n array of m_B original observations in an n-dimensional space.

- **metric** (*str, function, optional*) – The distance metric to use; the distance can be 'braycurtis', 'canberra', 'chebyshev', 'cityblock', 'correlation', 'cosine', 'dice', 'euclidean', 'hamming', 'jaccard', 'kulsinski', 'mahalanobis', 'matching', 'minkowski', 'pcosine', 'rogerstanimoto', 'russellrao', 'seuclidean', 'sokalmichener', 'sokalsneath', 'sqeuclidean', 'yule'.
- **p** (*float, optional*) – The p-norm to apply (for Minkowski, weighted and unweighted).
- **w** (*array, optional*) – The weight vector (for weighted Minkowski).
- **V** (*array, optional*) – The variance vector (for standardized Euclidean).
- **VI** (*array, optional*) – The inverse of the covariance matrix (for Mahalanobis).

Returns **Y** (*array*) – An m_A by m_B distance matrix is returned. For each i and j, the metric `dist(u=XA[i], v=XB[j])` is computed and stored in the ij th entry.

biosppy.metrics.**pcosine**(*u*, *v*)

Computes the Cosine distance (positive space) between 1-D arrays.

The Cosine distance (positive space) between *u* and *v* is defined as

$$d(u, v) = 1 - abs\left(\frac{u \cdot v}{||u||_2||v||_2}\right)$$

where $u \cdot v$ is the dot product of u and v.

Parameters

- **u** (*array*) – Input array.
- **v** (*array*) – Input array.

Returns **cosine** (*float*) – Cosine distance between *u* and *v*.

biosppy.metrics.**pdist**(*X*, *metric='euclidean'*, *p=2*, *w=None*, *V=None*, *VI=None*)

Pairwise distances between observations in n-dimensional space.

Wraps scipy.spatial.distance.pdist.

Parameters

- **X** (*array*) – An m by n array of m original observations in an n-dimensional space.
- **metric** (*str, function, optional*) – The distance metric to use; the distance can be 'braycurtis', 'canberra', 'chebyshev', 'cityblock', 'correlation', 'cosine', 'dice', 'euclidean', 'hamming', 'jaccard', 'kulsinski', 'mahalanobis', 'matching', 'minkowski', 'pcosine', 'rogerstanimoto', 'russellrao', 'seuclidean', 'sokalmichener', 'sokalsneath', 'sqeuclidean', 'yule'.
- **p** (*float, optional*) – The p-norm to apply (for Minkowski, weighted and unweighted).
- **w** (*array, optional*) – The weight vector (for weighted Minkowski).
- **V** (*array, optional*) – The variance vector (for standardized Euclidean).
- **VI** (*array, optional*) – The inverse of the covariance matrix (for Mahalanobis).

Returns **Y** (*array*) – Returns a condensed distance matrix Y. For each i and j (where $i < j < n$), the metric `dist(u=X[i], v=X[j])` is computed and stored in entry `ij`.

biosppy.metrics.**squareform**(*X*, *force='no'*, *checks=True*)

Converts a vector-form distance vector to a square-form distance matrix, and vice-versa.

Wraps scipy.spatial.distance.squareform.

Parameters

- **X** (*array*) – Either a condensed or redundant distance matrix.
- **force** (*str, optional*) – As with MATLAB(TM), if force is equal to 'tovector' or 'tomatrix', the input will be treated as a distance matrix or distance vector respectively.
- **checks** (*bool, optional*) – If *checks* is set to False, no checks will be made for matrix symmetry nor zero diagonals. This is useful if it is known that `X - X.T1` is small and `diag(X)` is close to zero. These values are ignored any way so they do not disrupt the squareform transformation.

Returns **Y** (*array*) – If a condensed distance matrix is passed, a redundant one is returned, or if a redundant one is passed, a condensed distance matrix is returned.

2.2.4 biosppy.plotting

This module provides utilities to plot data.

copyright

3. 2015 by Instituto de Telecomunicacoes

license BSD 3-clause, see LICENSE for more details.

biosppy.plotting.**plot_biometrics**(*assessment=None*, *eer_idx=None*, *path=None*, *show=False*)
Create a summary plot of a biometrics test run.

Parameters

- **assessment** (*dict*) – Classification assessment results.
- **eer_idx** (*int, optional*) – Classifier reference index for the Equal Error Rate.
- **path** (*str, optional*) – If provided, the plot will be saved to the specified file.
- **show** (*bool, optional*) – If True, show the plot immediately.

biosppy.plotting.**plot_bvp**(*ts=None*, *raw=None*, *filtered=None*, *onsets=None*, *heart_rate_ts=None*, *heart_rate=None*, *path=None*, *show=False*)
Create a summary plot from the output of signals.bvp.bvp.

Parameters

- **ts** (*array*) – Signal time axis reference (seconds).
- **raw** (*array*) – Raw BVP signal.
- **filtered** (*array*) – Filtered BVP signal.
- **onsets** (*array*) – Indices of BVP pulse onsets.
- **heart_rate_ts** (*array*) – Heart rate time axis reference (seconds).
- **heart_rate** (*array*) – Instantaneous heart rate (bpm).
- **path** (*str, optional*) – If provided, the plot will be saved to the specified file.
- **show** (*bool, optional*) – If True, show the plot immediately.

biosppy.plotting.**plot_clustering**(*data=None*, *clusters=None*, *path=None*, *show=False*)
Create a summary plot of a data clustering.

Parameters

- **data** (*array*) – An m by n array of m data samples in an n-dimensional space.
- **clusters** (*dict*) – Dictionary with the sample indices (rows from *data*) for each cluster.
- **path** (*str, optional*) – If provided, the plot will be saved to the specified file.
- **show** (*bool, optional*) – If True, show the plot immediately.

biosppy.plotting.**plot_ecg**(*ts=None*, *raw=None*, *filtered=None*, *rpeaks=None*, *templates_ts=None*, *templates=None*, *heart_rate_ts=None*, *heart_rate=None*, *path=None*, *show=False*)

Create a summary plot from the output of signals.ecg.ecg.

Parameters

- **ts** (*array*) – Signal time axis reference (seconds).
- **raw** (*array*) – Raw ECG signal.
- **filtered** (*array*) – Filtered ECG signal.
- **rpeaks** (*array*) – R-peak location indices.
- **templates_ts** (*array*) – Templates time axis reference (seconds).
- **templates** (*array*) – Extracted heartbeat templates.
- **heart_rate_ts** (*array*) – Heart rate time axis reference (seconds).
- **heart_rate** (*array*) – Instantaneous heart rate (bpm).
- **path** (*str, optional*) – If provided, the plot will be saved to the specified file.
- **show** (*bool, optional*) – If True, show the plot immediately.

biosppy.plotting.**plot_eda**(*ts=None*, *raw=None*, *filtered=None*, *onsets=None*, *peaks=None*, *amplitudes=None*, *path=None*, *show=False*)

Create a summary plot from the output of signals.eda.eda.

Parameters

- **ts** (*array*) – Signal time axis reference (seconds).
- **raw** (*array*) – Raw EDA signal.
- **filtered** (*array*) – Filtered EDA signal.
- **onsets** (*array*) – Indices of SCR pulse onsets.
- **peaks** (*array*) – Indices of the SCR peaks.
- **amplitudes** (*array*) – SCR pulse amplitudes.
- **path** (*str, optional*) – If provided, the plot will be saved to the specified file.
- **show** (*bool, optional*) – If True, show the plot immediately.

biosppy.plotting.**plot_eeg**(*ts=None*, *raw=None*, *filtered=None*, *labels=None*, *features_ts=None*, *theta=None*, *alpha_low=None*, *alpha_high=None*, *beta=None*, *gamma=None*, *plf_pairs=None*, *plf=None*, *path=None*, *show=False*)

Create a summary plot from the output of signals.eeg.eeg.

Parameters

- **ts** (*array*) – Signal time axis reference (seconds).
- **raw** (*array*) – Raw EEG signal.
- **filtered** (*array*) – Filtered EEG signal.

- **labels** (*list*) – Channel labels.
- **features_ts** (*array*) – Features time axis reference (seconds).
- **theta** (*array*) – Average power in the 4 to 8 Hz frequency band; each column is one EEG channel.
- **alpha_low** (*array*) – Average power in the 8 to 10 Hz frequency band; each column is one EEG channel.
- **alpha_high** (*array*) – Average power in the 10 to 13 Hz frequency band; each column is one EEG channel.
- **beta** (*array*) – Average power in the 13 to 25 Hz frequency band; each column is one EEG channel.
- **gamma** (*array*) – Average power in the 25 to 40 Hz frequency band; each column is one EEG channel.
- **plf_pairs** (*list*) – PLF pair indices.
- **plf** (*array*) – PLF matrix; each column is a channel pair.
- **path** (*str, optional*) – If provided, the plot will be saved to the specified file.
- **show** (*bool, optional*) – If True, show the plot immediately.

biosppy.plotting.**plot_emg**(*ts=None*, *raw=None*, *filtered=None*, *onsets=None*, *path=None*, *show=False*)

Create a summary plot from the output of signals.emg.emg.

Parameters

- **ts** (*array*) – Signal time axis reference (seconds).
- **raw** (*array*) – Raw EMG signal.
- **filtered** (*array*) – Filtered EMG signal.
- **onsets** (*array*) – Indices of EMG pulse onsets.
- **path** (*str, optional*) – If provided, the plot will be saved to the specified file.
- **show** (*bool, optional*) – If True, show the plot immediately.

biosppy.plotting.**plot_filter**(*ftype='FIR'*, *band='lowpass'*, *order=None*, *frequency=None*, *sampling_rate=1000.0*, *path=None*, *show=True*, ***kwargs*)

Plot the frequency response of the filter specified with the given parameters.

Parameters

- **ftype** (*str*) – Filter type: * Finite Impulse Response filter ('FIR'); * Butterworth filter ('butter'); * Chebyshev filters ('cheby1', 'cheby2'); * Elliptic filter ('ellip'); * Bessel filter ('bessel').
- **band** (*str*) – Band type: * Low-pass filter ('lowpass'); * High-pass filter ('highpass'); * Band-pass filter ('bandpass'); * Band-stop filter ('bandstop').
- **order** (*int*) – Order of the filter.
- **frequency** (*int, float, list, array*) – Cutoff frequencies; format depends on type of band: * 'lowpass' or 'bandpass': single frequency; * 'bandpass' or 'bandstop': pair of frequencies.
- **sampling_rate** (*int, float, optional*) – Sampling frequency (Hz).
- **path** (*str, optional*) – If provided, the plot will be saved to the specified file.

- **show** (*bool, optional*) – If True, show the plot immediately.
- ****kwargs** (*dict, optional*) – Additional keyword arguments are passed to the underlying scipy.signal function.

biosppy.plotting.**plot_resp**(*ts=None*, *raw=None*, *filtered=None*, *zeros=None*, *resp_rate_ts=None*, *resp_rate=None*, *path=None*, *show=False*)

Create a summary plot from the output of signals.bvp.bvp.

Parameters

- **ts** (*array*) – Signal time axis reference (seconds).
- **raw** (*array*) – Raw Resp signal.
- **filtered** (*array*) – Filtered Resp signal.
- **zeros** (*array*) – Indices of Respiration zero crossings.
- **resp_rate_ts** (*array*) – Respiration rate time axis reference (seconds).
- **resp_rate** (*array*) – Instantaneous respiration rate (Hz).
- **path** (*str, optional*) – If provided, the plot will be saved to the specified file.
- **show** (*bool, optional*) – If True, show the plot immediately.

biosppy.plotting.**plot_spectrum**(*signal=None*, *sampling_rate=1000.0*, *path=None*, *show=True*)

Plot the power spectrum of a signal (one-sided).

Parameters

- **signal** (*array*) – Input signal.
- **sampling_rate** (*int, float, optional*) – Sampling frequency (Hz).
- **path** (*str, optional*) – If provided, the plot will be saved to the specified file.
- **show** (*bool, optional*) – If True, show the plot immediately.

2.2.5 biosppy.storage

This module provides several data storage methods.

copyright

3. 2015 by Instituto de Telecomunicacoes

license BSD 3-clause, see LICENSE for more details.

class biosppy.storage.**HDF**(*path=None*, *mode='a'*)

Bases: object

Wrapper class to operate on BioSPPy HDF5 files.

Parameters

- **path** (*str*) – Path to the HDF5 file.
- **mode** (*str, optional*) – File mode; one of:
  - 'a': read/write, creates file if it does not exist;
  - 'r+': read/write, file must exist;
  - 'r': read only, file must exist;
  - 'w': create file, truncate if it already exists;

- 'w-': create file, fails if it already esists.

__enter__()
Method for with statement.

__exit__(*exc_type*, *exc_value*, *traceback*)
Method for with statement.

add_event(*ts=None*, *values=None*, *mdata=None*, *group=''*, *name=None*, *compress=False*)
Add an event to the file.

Parameters

- **ts** (*array*) – Array of time stamps.
- **values** (*array, optional*) – Array with data for each time stamp.
- **mdata** (*dict, optional*) – Event metadata.
- **group** (*str, optional*) – Destination event group.
- **name** (*str, optional*) – Name of the dataset to create.
- **compress** (*bool, optional*) – If True, the data will be compressed with gzip.

Returns

- **group** (*str*) – Destination group.
- **name** (*str*) – Name of the created event dataset.

add_header(*header=None*)
Add header metadata.

Parameters **header** (*dict*) – Header metadata.

add_signal(*signal=None*, *mdata=None*, *group=''*, *name=None*, *compress=False*)
Add a signal to the file.

Parameters

- **signal** (*array*) – Signal to add.
- **mdata** (*dict, optional*) – Signal metadata.
- **group** (*str, optional*) – Destination signal group.
- **name** (*str, optional*) – Name of the dataset to create.
- **compress** (*bool, optional*) – If True, the signal will be compressed with gzip.

Returns

- **group** (*str*) – Destination group.
- **name** (*str*) – Name of the created signal dataset.

close()
Close file descriptor.

del_event(*group=''*, *name=None*)
Delete an event from the file.

Parameters

- **group** (*str, optional*) – Event group.
- **name** (*str*) – Name of the event dataset.

del_event_group(*group=''*)
Delete all events in a file group.

Parameters **str, optional** (*group*) – Event group.

del_signal(*group=''*, *name=None*)
Delete a signal from the file.

Parameters

- **group** (*str, optional*) – Signal group.
- **name** (*str*) – Name of the dataset.

del_signal_group(*group=''*)
Delete all signals in a file group.

Parameters **group** (*str, optional*) – Signal group.

get_event(*group=''*, *name=None*)
Retrieve an event from the file.

Parameters

- **group** (*str, optional*) – Event group.
- **name** (*str*) – Name of the event dataset.

Returns

- **ts** (*array*) – Array of time stamps.
- **values** (*array*) – Array with data for each time stamp.
- **mdata** (*dict*) – Event metadata.

Notes

Loads the entire event data into memory.

get_header()
Retrieve header metadata.

Returns **header** (*dict*) – Header metadata.

get_signal(*group=''*, *name=None*)
Retrieve a signal from the file.

Parameters

- **group** (*str, optional*) – Signal group.
- **name** (*str*) – Name of the signal dataset.

Returns

- **signal** (*array*) – Retrieved signal.
- **mdata** (*dict*) – Signal metadata.

Notes

•Loads the entire signal data into memory.

list_events (*group=''*, *recursive=False*)
List events in the file.

Parameters

- **group** (*str, optional*) – Event group.
- **recursive** (*bool, optional*) – It True, also lists events in sub-groups.

Returns **events** (*list*) – List of (group, name) tuples of the found events.

list_signals (*group=''*, *recursive=False*)
List signals in the file.

Parameters

- **group** (*str, optional*) – Signal group.
- **recursive** (*bool, optional*) – It True, also lists signals in sub-groups.

Returns **signals** (*list*) – List of (group, name) tuples of the found signals.

biosppy.storage.**alloc_h5** (*path*)
Prepare an HDF5 file.

Parameters **path** (*str*) – Path to file.

biosppy.storage.**deserialize** (*path*)
Deserialize data from a file using sklearn's joblib.

Parameters **path** (*str*) – Source path.

Returns **data** (*object*) – Deserialized object.

biosppy.storage.**dumpJSON** (*data*, *path*)
Save JSON data to a file.

Parameters

- **data** (*dict*) – The JSON data to dump.
- **path** (*str*) – Destination path.

biosppy.storage.**loadJSON** (*path*)
Load JSON data from a file.

Parameters **path** (*str*) – Source path.

Returns **data** (*dict*) – The loaded JSON data.

biosppy.storage.**load_h5** (*path*, *label*)
Load data from an HDF5 file.

Parameters

- **path** (*str*) – Path to file.
- **label** (*hashable*) – Data label.

Returns **data** (*array*) – Loaded data.

biosppy.storage.**load_txt** (*path*)
Load data from a text file.

Parameters **path** (*str*) – Path to file.

Returns

- **data** (*array*) – Loaded data.
- **mdata** (*dict*) – Metadata.

biosppy.storage.**pack_zip**(*files*, *path*, *recursive=True*, *forceExt=True*)
Pack files into a zip archive.

Parameters

- **files** (*iterable*) – List of files or directories to pack.
- **path** (*str*) – Destination path.
- **recursive** (*bool, optional*) – If True, sub-directories and sub-folders are also written to the archive.
- **forceExt** (*bool, optional*) – Append default extension.

Returns **zip_path** (*str*) – Full path to created zip archive.

biosppy.storage.**serialize**(*data*, *path*, *compress=3*)
Serialize data and save to a file using sklearn's joblib.

Parameters

- **data** (*object*) – Object to serialize.
- **path** (*str*) – Destination path.
- **compress** (*int, optional*) – Compression level; from 0 to 9 (highest compression).

biosppy.storage.**store_h5**(*path*, *label*, *data*)
Store data to HDF5 file.

Parameters

- **path** (*str*) – Path to file.
- **label** (*hashable*) – Data label.
- **data** (*array*) – Data to store.

biosppy.storage.**store_txt**(*path*, *data*, *sampling_rate=1000.0*, *resolution=None*, *date=None*, *labels=None*, *precision=6*)
Store data to a simple text file.

Parameters

- **path** (*str*) – Path to file.
- **data** (*array*) – Data to store (up to 2 dimensions).
- **sampling_rate** (*int, float, optional*) – Sampling frequency (Hz).
- **resolution** (*int, optional*) – Sampling resolution.
- **date** (*datetime, str, optional*) – Datetime object, or an ISO 8601 formatted date-time string.
- **labels** (*list, optional*) – Labels for each column of *data*.
- **precision** (*int, optional*) – Precision for string conversion.

Raises

- ValueError – If the number of data dimensions is greater than 2.

- `ValueError` – If the number of labels is inconsistent with the data.

`biosppy.storage.`**`unpack_zip`**(*zip_path*, *path*)
Unpack a zip archive.

Parameters

- **`zip_path`** (*`str`*) – Path to zip archive.
- **`path`** (*`str`*) – Destination path (directory).

`biosppy.storage.`**`zip_write`**(*fid*, *files*, *recursive=True*, *root=None*)
Write files to zip archive.

Parameters

- **`fid`** (*`file-like object`*) – The zip file to write into.
- **`files`** (*`iterable`*) – List of files or directories to pack.
- **`recursive`** (*`bool, optional`*) – If True, sub-directories and sub-folders are also written to the archive.
- **`root`** (*`str, optional`*) – Relative folder path.

Notes

•Ignores non-existent files and directories.

2.2.6 biosppy.utils

This module provides several frequently used functions and hacks.

copyright

3. 2015 by Instituto de Telecomunicacoes

license BSD 3-clause, see LICENSE for more details.

class `biosppy.utils.`**`ReturnTuple`**(*values*, *names=None*)
Bases: `tuple`

A named tuple to use as a hybrid tuple-dict return object.

Parameters

- **`values`** (*`iterable`*) – Return values.
- **`names`** (*`iterable, optional`*) – Names for return values.

Raises

- `ValueError` – If the number of values differs from the number of names.
- `ValueError` – If any of the items in names: * contain non-alphanumeric characters; * are Python keywords; * start with a number; * are duplicates.

`__getitem__`(*key*)
Get item as an index or keyword.

Returns **out** (*object*) – The object corresponding to the key, if it exists.

Raises

- KeyError – If the key is a string and it does not exist in the mapping.
- IndexError – If the key is an int and it is out of range.

__getnewargs__()
Return self as a plain tuple; used for copy and pickle.

__repr__()
Return representation string.

as_dict()
Convert to an ordered dictionary.

Returns **out** (*OrderedDict*) – An OrderedDict representing the return values.

keys()
Return the value names.

Returns **out** (*list*) – The keys in the mapping.

biosppy.utils.**highestAveragesAllocator**(*votes*, *k*, *divisor='dHondt'*, *check=False*)
Allocate k seats proportionally using the Highest Averages Method.

Parameters

- **votes** (*list*) – Number of votes for each class/party/cardinal.
- **k** (*int*) – Total number o seats to allocate.
- **divisor** (*str, optional*) – Divisor method; one of 'dHondt', 'Huntington-Hill', 'Sainte-Lague', 'Imperiali', or 'Danish'.
- **check** (*bool, optional*) – If True, limits the number of seats to the total number of votes.

Returns **seats** (*list*) – Number of seats for each class/party/cardinal.

biosppy.utils.**normpath**(*path*)
Normalize a path.

Parameters **path** (*str*) – The path to normalize.

Returns **npath** (*str*) – The normalized path.

biosppy.utils.**random_fraction**(*indx*, *fraction*, *sort=True*)
Select a random fraction of an input list of elements.

Parameters

- **indx** (*list, array*) – Elements to partition.
- **fraction** (*int, float*) – Fraction to select.
- **sort** (*bool, optional*) – If True, output lists will be sorted.

Returns

- **use** (*list, array*) – Selected elements.
- **unuse** (*list, array*) – Remaining elements.

biosppy.utils.**remainderAllocator**(*votes*, *k*, *reverse=True*, *check=False*)
Allocate k seats proportionally using the Remainder Method.

Also known as Hare-Niemeyer Method. Uses the Hare quota.

Parameters

- **votes** (*list*) – Number of votes for each class/party/cardinal.
- **k** (*int*) – Total number o seats to allocate.
- **reverse** (*bool, optional*) – If True, allocates remaining seats largest quota first.
- **check** (*bool, optional*) – If True, limits the number of seats to the total number of votes.

Returns **seats** (*list*) – Number of seats for each class/party/cardinal.

CHAPTER 3

Installation

Installation can be easily done with `pip`:

```
$ pip install biosppy
```

CHAPTER 4

Simple Example

The code below loads an ECG signal from the `examples` folder, filters it, performs R-peak detection, and computes the instantaneous heart rate.

```
import numpy as np
from biosppy.signals import ecg

# load raw ECG signal
signal = np.loadtxt('./examples/ecg.txt')

# process it and plot
out = ecg.ecg(signal=signal, sampling_rate=1000., show=True)
```

CHAPTER 5

Index

- genindex
- modindex
- search

Bibliography

[Chri04] Ivaylo I. Christov, "Real time electrocardiogram QRS detection using combined adaptive threshold", BioMedical Engineering OnLine 2004, vol. 3:28, 2004

[EnZe79] W. Engelse and C. Zeelenberg, "A single scan algorithm for QRS detection and feature extraction", IEEE Comp. in Cardiology, vol. 6, pp. 37-42, 1979

[LSLL12] A. Lourenco, H. Silva, P. Leite, R. Lourenco and A. Fred, "Real Time Electrocardiogram Segmentation for Finger Based ECG Biometrics", BIOSIGNALS 2012, pp. 49-54, 2012

[Hami02] P.S. Hamilton, "Open Source ECG Analysis Software Documentation", E.P.Limited, 2002

[KiBK04] K.H. Kim, S.W. Bang, and S.R. Kim, "Emotion recognition system using short-term monitoring of physiological signals", Med. Biol. Eng. Comput., vol. 42, pp. 419-427, 2004

[Ferm] Wikipedia, "Fermat's theorem (stationary points)", https://en.wikipedia.org/wiki/Fermat%27s_theorem_(stationary_points)

[MAvg] Wikipedia, "Moving Average", http://en.wikipedia.org/wiki/Moving_average

[Smit97] S. W. Smith, "Moving Average Filters - Implementation by Convolution", http://www.dspguide.com/ch15/1.htm, 1997

[EKSX96] M. Ester, H. P. Kriegel, J. Sander, and X. Xu, "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise", Proceedings of the 2nd International Conf. on Knowledge Discovery and Data Mining, pp. 226-231, 1996.

[UlRJ04] U. Uludag, A. Ross, A. Jain, "Biometric template selection and update: a case study in fingerprints", Pattern Recognition 37, 2004

[LCSF14] A. Lourenco, C. Carreiras, H. Silva, A. Fred, "ECG biometrics: A template selection approach", 2014 IEEE International Symposium on Medical Measurements and Applications (MeMeA), 2014

[LCSF13] A. Lourenco, H. Silva, C. Carreiras, A. Fred, "Outlier Detection in Non-intrusive ECG Biometric System", Image Analysis and Recognition, vol. 7950, pp. 43-52, 2013

b

Symbols

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

U

W

Z