Module `spectral`

Spectral

A library for analysing timeseries data, specifically for neural event identification, detection and classification.

Consists of three submodules, which can be used either independently or together:

1. Contrast: This module enables the contrasting between two categories of timeseries data (with multiple trials per category). This enables the identification of the frequency bins that have the most differnce between the categories.
2. Cluster: This module enables the clustering of the timeseries data based on the similarity of it's spectral decomposition using STFT. It finds the optimal number of clusters and returns a vector with labels of which class each STFT segment belongs to.
3. Classify: This module provides code to train classifiers for classifying timeseries data based on the clusters identified using the `spectral.cluster` module.

Author: Ali Zaidi

Version 0.1

Sub-modules

- spectral.classify
- spectral.cluster
- spectral.contrast

Module `spectral.classify`

Module for classifying timeseries data based on their spectral properties.

This module enables the classification of STFT transformed data by training an SVM classifier.

This facilitates the identification of various events (defined as transient spatio-temporal patterns of activity) present within the timeseries data, that have been identified.

Functions

Function `classifySVM`

```
def classifySVM(self, X, y)
```

Trains an SVM-classifier on the data.

Parameters

`X : array` Training data with shape: nobs x features
`y : array` vector with training labels

Returns

`scores : array` a vector of scores with length equal to number of CV-folds
`clf : python object` the trained classifier as a python object

Function `generate_features`

```
def generate_features(data, labels, **kwargs)
```

Generate a feature vector for training a classifier

Parameters

`data : array` array with structure
`labels : array` vector with class labels

Returns

`X : array` an array with the features in the 1st axis and trials on the 0-th axis
`y : array` a vector with the same number of rows as X containing class labels

Module `spectral.cluster`

Module for clustering timeseries data based on their spectral properties.

This module enables the clustering of STFT transformed data by mapping the STFT arrays to a low dimensional manifold and then clustering them using DBSCAN.

This facilitates the identification of various events (defined as transient spatio-temporal patterns of activity) present within the timeseries data.

Functions

Function `cluster`

```
def cluster(data, **kwargs)
```

Clusters the array using OPTICS and dbscan. Finds the best number of clusters.

Parameters

`data_array : array` STFT array or low-dimensional embedding from `embed()` [nchan x nobs x ntrials]

Returns

`res : array` results with res[0] having the
`nclust : int` number of clusters identified

Function `embed`

```
def embed(data_stft_norm, **kwargs)
```

Returns a low-dimensional embedding of an STFT array.

Parameters

`data_norm : array` normalized stft array [nchan x nfreqs x nobs x ntrials]

Returns

`embedding : array` low dimensional embedding of the STFT array

Function `stft_norm`

```
def stft_norm(data, **kwargs)
```

Returns the frequency-normalized STFT for timeseries data.

Parameters

`data_array : array` timeseries data [nchan x nobs x ntrials]
`*fs : int` sampling frequency in Hz
`*nperseg : int` number of timepoints for stft window
`*noverlap : int` number of timepoints for window overlap

Returns

`stft_norm() : array` STFT of the input array [nchan x nfreqs x nobs x trials]
`f : array` an array of the frequencies of the STFT transform

Module `spectral.contrast`

Module for contrasting timeseries data based on their spectral properties.

The set of methods are aimed at finding the frequency bands that enable the maximal seperability betwen two sets of timeseries data.

Functions

Function `contrast`

```
def contrast(data, y, **kwargs)
```

This method returns the SNR given a data array and vector of labels.

Ideally, this should be the only method that you need to call when contrasting timeseries' spectra.

Parameters

`data : array [nchans x nobs x ntrials]` an array with the LFP data organized into channels and trials.
`y : array [ntrials]` a binary vector with a label for each trial being either 0 or 1

Returns

`snr : array [nfreqs x nfreqs]` a matrix with the SNR for each combination of frequency bands
`f : array [nfreqs]` a vector that represents the frequencies for interpreting `snr`.

Function `filter`

```
def filter(data, low_pass, high_pass, fs, order=10)
```

Generates an n-th order butterworth filter and performs forward-backward pass on the signal.

Parameters

`data : array` same as data structure [nchans x nobs x ntrials]
`low_pass : param` low pass frequency
`high_pass : param` high pass frequency
`fs : param` sampling frequency
`order : param` filter order

Returns

`filt_data : array` array with same shape as `data` but bandpass filtered

Function `generate_ts`

```
def generate_ts(nsamples=10000, fs=1000, **kwargs)
```

Generates a 10s long LFP-like timeseries at 1kHz obeying the power law.

Function `get_bands`

```
def get_bands(target_stft_norm, baseline_stft_norm, f)
```

Calculates the mean power across all possible combinations of frequencies for each channel.

Parameters

`target_stft_norm : array` stft decomposed target array [nchan x nfreqs x nobs x ntrials]
`baseline_stft_norm : array` stft decomposed baseline array [nchan x nfreqs x nobs x ntrials]
`f : array` vector of frequencies obtained from STFT transform (see `get_stft()`).

Returns

`target_bands : array` array of mean power across all possible band permutations [nchan x nfreqs x nfreqs x nobs x ntrials]
`baseline_bands : array` same as `target_bands` [nchan x nfreqs x nfreqs x nobs x ntrials]

Function `get_norm_array`

```
def get_norm_array(data, **kwargs)
```

Returns the normalization array for timeseries data.

Parameters

`data : array, timeseries data [nchan x nobs x ntrials]`
`*fs : int, sampling frequency in Hz`
`*nperseg : int, number of timepoints for stft window`
`*noverlap : int, number of timepoints for window overlap`

Returns

`norm_array : normalized array with mean power per frequency [nchan x freqs]`

Function `get_snr`

```
def get_snr(target, baseline)
```

Returns the SNR given two vectors: target and baseline.

Parameters

`target : array [nchan x nfreqs x nfreqs x nobs x ntrials]` an array obtained by using `get_bands()`
`baseline : array (same as target)`

Returns

`snr : array [nfreqs x nfreqs]` a lower triangular matix representing the contrast between bands

Function `get_stft`

```
def get_stft(data_array, norm_array=[], normalize=True, **kwargs)
```

Returns the STFT for timeseries data.

Parameters

`data_array : array` timeseries data [nchan x nobs x ntrials]
`norm_array : array` for spectral normalization (see `get_norm_array()`)
`*fs : int` sampling frequency in Hz
`*nperseg : int` number of timepoints for stft window
`*noverlap : int` number of timepoints for window overlap

Returns

`stft_array : array` STFT of the input array [nchan x nfreqs x nobs x trials]
`f : array` an array of the frequencies of the STFT transform

Function `simulate_recording`

```
def simulate_recording(nchans=10, nsamples=10000, fs=1000, **kwargs)
```

Simulates an LFP recording with bursts in power of certain bands.

Function `test`

```
def test()
```

Simple test method to ensure that the pipeline and dependencies work.

Returns `True` if everything works.

Generated by *pdoc* 0.7.2 (https://pdoc3.github.io).