hiwenet Documentation

Release 0.2.5

Pradeep Reddy Raamana

Dec 13, 2017

Contents:

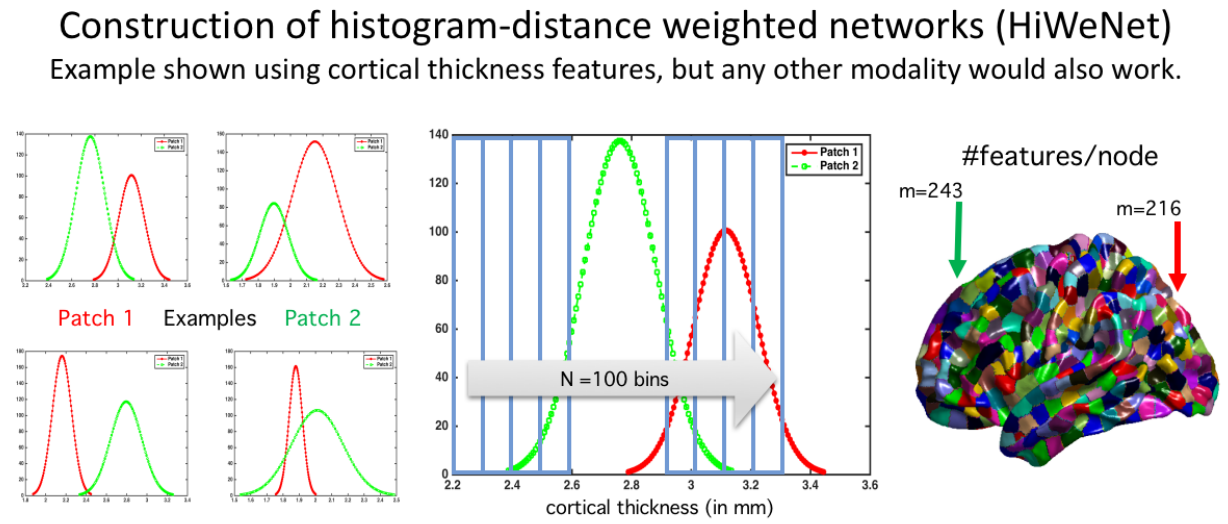
Hiwenet helps you extracts histogram-distance weighted networks for feature extraction and advanced analysis in neuroscience and other domains.

CHAPTER 1

About

Network-level analysis of various features, esp. if it can be individualized for a single-subject, is proving to be quite a valuable tool in many applications. This package extracts single-subject (individualized, or intrinsic) networks from node-wise data by computing the edge weights based on histogram distance between the distributions of values within each node. Individual nodes could be an ROI or a patch or a cube, or any other unit of relevance in your application. This is a great way to take advantage of the full distribution of values available within each node, relative to the simpler use of averages (or another summary statistic).

Rough scheme of computation is shown below:



The four smaller subpanels on the left show typical distributions of cortical thickness values for four random pairs of patches (in green and red) in a given subject (shown on cortical visualization on right). They demonstrate the means and shape of these distributions can vary substantially as you traverse across different pairs of cortical patches. The large panel in the middle illustrates the type of binning used to construct the histogram from each patch.

1.1 Applicability

Although this technique was originally developed for cortical thickness, this is a generic and powerful technique that could be applied to any features such as gray matter density, PET uptake values, functional activation data or EEG features. All that is needed is a set of nodes/parcellation that have one-to-one correspondence across samples/subjects in your dataset.

The target audience is users of almost all neuroimaging modalities who:

1. preprocessed dataset already,
2. have some base features extracted (node- or patch-wise, that are native to the given modality) using other packages (metioned above), and
3. who would like to analyze network-level (i.e. covariance-type or connectivity) relations among the base features (either in space across the cortex or a relevant domain, or across time).
4. This is similar to popular metrics of covariance like Correlation or LedoitWolf, and could be dropped in their place. Do you want to find out how histogram-based method compare to your own ideas?

1.2 What does hiwenet do?

- This packages takes in vector of features and their membership labels (denoting which features belong to which groups - alternatively referred to as nodes in a graph), and computes their pair-wise histogram distances, using a chosen method.
- This package is designed to be domain-agnostic, and hence a generic input format was chosen.
- However, we plan to add interfaces to tools that may be of interest to researchers in specific domains such as nilearn, MNE and the related. A scikit-learn compatible API/interface is also in the works.
- Refer to *Examples using API* and *API Reference* pages for more detailed and usage examples, and *examples* directory for sample files.

Thanks for checking out. Your feedback will be appreciated.

1.3 Citation

If you found this toolbox useful for your research, please cite one or more of these papers in that order:

- Raamana, P. R. and Strother, S.C., 2017, Histogram-weighted Networks for Feature Extraction, Connectivity and Advanced Analysis in Neuroscience. Journal of Open Source Software, 2(19), 380, doi:10.21105/joss.00380
- Raamana, P.R. and Strother, S.C., 2017, Impact of spatial scale and edge weight on predictive power of cortical thickness networks bioRxiv 170381 http://www.biorxiv.org/content/early/2017/07/31/170381.
- Raamana, P. R., Weiner, M. W., Wang, L., Beg, M. F., & Alzheimer's Disease Neuroimaging Initiative. (2015). Thickness network features for prognostic applications in dementia. Neurobiology of aging, 36, S91-S102.

1.4 Acknowledgements

I would like to thank Oscar Esteban (@oesteban) for his volunteer and attentive review of this package, which has been very helpful in improving the software.

CHAPTER 2

Installation

This package could easily be installed via:

```
pip install hiwenet
```

2.1 Requirements

- numpy
- medpy
- hypothesis
- networkx

CHAPTER 3

API Reference

A tutorial-like presentation is available at *Examples using API*, using the following API. # Histogram-weighted Networks (hiwenet)

hiwenet.**extract**(*features*, *groups*, *weight_method='manhattan'*, *num_bins=25*, *edge_range=None*, *trim_outliers=True*, *trim_percentile=5*, *use_original_distribution=False*, *relative_to_all=False*, *asymmetric=False*, *return_networkx_graph=False*, *out_weights_path=None*)

Extracts the histogram-distance weighted adjacency matrix.

Parameters **features** : ndarray or str

1d array of scalar values, either provided directly as a 1d numpy array, or as a path to a file containing these values

groups : ndarray or str

Membership array of same length as *features*, each value specifying which group that particular node belongs to. Input can be either provided directly as a 1d numpy array,or as a path to a file containing these values.

For example, if you have cortical thickness values for 1000 vertices (*features* is ndarray of length 1000), belonging to 100 patches, the groups array (of length 1000) could have numbers 1 to 100 (number of unique values) specifying which element belongs to which cortical patch.

Grouping with numerical values (contiguous from 1 to num_patches) is strongly recommended for simplicity, but this could also be a list of strings of length p, in which case a tuple is returned, identifying which weight belongs to which pair of patches.

weight_method : string or callable, optional

Type of distance (or metric) to compute between the pair of histograms. It can either be a string identifying one of the weights implemented below, or a valid callable.

If a string, it must be one of the following methods:

- 'chebyshev'

- 'chebyshev_neg'
- 'chi_square'
- 'correlate'
- 'correlate_1'
- 'cosine'
- 'cosine_1'
- 'cosine_2'
- 'cosine_alt'
- 'euclidean'
- 'fidelity_based'
- 'histogram_intersection'
- 'histogram_intersection_1'
- 'jensen_shannon'
- 'kullback_leibler'
- 'manhattan'
- 'minowski'
- 'noelle_1'
- 'noelle_2'
- 'noelle_3'
- 'noelle_4'
- 'noelle_5'
- 'relative_bin_deviation'
- 'relative_deviation'

Note only the following are *metrics*:

- 'manhattan'
- 'minowski'
- 'euclidean'
- 'noelle_2'
- 'noelle_4'
- 'noelle_5'

The following are *semi- or quasi-metrics*:

- 'kullback_leibler'
- 'jensen_shannon'
- 'chi_square'
- 'chebyshev'
- 'cosine_1'

- 'chebyshev_neg'
- 'correlate_1'
- 'histogram_intersection_1'
- 'relative_deviation'
- 'relative_bin_deviation'
- 'noelle_1'
- 'noelle_3'

The following are classified to be similarity functions:

- 'histogram_intersection'
- 'correlate'
- 'cosine'
- 'cosine_2'
- 'cosine_alt'
- 'fidelity_based'

Default choice: 'minowski'.

The method can also be one of the following identifying metrics that operate on the original data directly - e.g. difference in the medians coming from the distributions of the pair of ROIs.

- 'diff_medians'
- 'diff_means'
- 'diff_medians_abs'
- 'diff_means_abs'

Please note this can lead to adjacency matrices that may not be symmetric e.g. difference metric on two scalars is not symmetric). In this case, be sure to use the flag: allow_non_symmetric=True

If weight_method is a callable, it must two accept two arrays as input and return one scalar as output. Example: `diff_in_skew = lambda x, y: abs(scipy.stats.skew(x)-scipy.stats.skew(y))` NOTE: this method will be applied to histograms (not the original distribution of features from group/ROI). In order to apply this callable directly on the original distribution (without trimming and histogram binning), use `use_original_distribution=True`.

num_bins : scalar, optional

Number of bins to use when computing histogram within each patch/group.

Note:

1. Please ensure same number of bins are used across different subjects
2. histogram shape can vary widely with number of bins (esp with fewer bins in the range of 3-20), and hence the features extracted based on them vary also.
3. It is recommended to study the impact of this parameter on the final results of the experiment.

This could also be optimized within an inner cross-validation loop if desired.

edge_range : tuple or None

The range of edges within which to bin the given values. This can be helpful to ensure correspondence across multiple invocations of hiwenet (for different subjects), in terms of range across all bins as well as individual bin edges. Default is to automatically compute from the given values.

Accepted format:

- tuple of finite values: (range_min, range_max)
- None, triggering automatic calculation (default)

Notes : when controlling the `edge_range`, it is not possible trim the tails (e.g. using the parameters `trim_outliers` and `trim_percentile`) for the current set of features using its own range.

trim_outliers : bool, optional

Whether to trim a small percentile of outliers at the edges of feature range, when features are expected to contain extreme outliers (like 0 or eps or Inf). This is important to avoid numerical problems and also to stabilize the weight estimates.

trim_percentile : float

Small value specifying the percentile of outliers to trim. Default: 5 (5%). Must be in open interval (0, 100).

use_original_distribution : bool, optional

When using a user-defined callable, this flag 1) allows skipping of pre-processing (trimming outliers) and histogram construction, 2) enables the application of arbitrary callable (user-defined) on the original distributions coming from the two groups/ROIs/nodes directly.

Example: `diff_in_medians = lambda x, y: abs(np.median(x)-np.median(y))`

This option is valid only when weight_method is a valid callable, which must take two inputs (possibly of different lengths) and return a single scalar.

relative_to_all : bool

Flag to instruct the computation of a grand histogram (distribution pooled from values in all ROIs), and compute distances (based on distance specified by `weight_method`) by from each ROI to the grand mean. This would result in only N distances for N ROIs, instead of the usual N*(N-1) pair-wise distances.

asymmetric : bool

Flag to identify resulting adjacency matrix is expected to be non-symmetric. Note: this results in twice the computation time! Default: False , for histogram metrics implemented here are symmetric.

return_networkx_graph : bool, optional

Specifies the need for a networkx graph populated with weights computed. Default: False.

out_weights_path : str, optional

Where to save the extracted weight matrix. If networkx output is returned, it would be saved in GraphML format. Default: nothing saved unless instructed.

Returns **edge_weights** : ndarray

numpy 2d array of pair-wise edge-weights (of size: num_groups x num_groups), wherein num_groups is determined by the total number of unique values in *groups*.

Note:

- Only the upper triangular matrix is filled as the distance between node i and j would be the same as j and i.
- The edge weights from the upper triangular matrix can easily be obtained by

```
weights_array = edge_weights[ np.triu_indices_from(edge_weights,
↪ 1) ]
```

hiwenet.**run_cli**()

Command line interface to hiwenet.

CHAPTER 4

Examples using API

This package computes single-subject networks, hence you may need loop over samples/subjects in your dataset to extract them for all the samples/subjects. Them proceed to your typical subsequent analysis (such as classification etc).

Note:

- The *hiwenet.extract* could be used to extract advance covariance/connectome features in place of MNE.extract_label_time_course or nilearn.input_data.NiftiLabelsMasker.transform - see here and here.
- However, we plan to add interfaces to tools e.g. via a scikit-learn compatible API/interface is also in the works. Stay tuned.

A rough example of usage when using the hiwenet API can be:

```
import hiwenet
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score
import numpy as np
import nibabel
import os

# ------------------------------------------------------------------------------------
# see docs/example_thickness_hiwenet.py for a concrete example
# ------------------------------------------------------------------------------------

for ss, subject in enumerate(subject_list):
  features = get_features(subject)
  edge_weight_matrix = hiwenet.extract(features, groups,  weight_method = 'kullback_
↪leibler')
  edge_weights_vec[ss,:] = upper_tri_vec(edge_weight_matrix)

  out_file = os.path.join(out_folder, 'hiwenet_{}.txt'.format(subject))
  np.save(out_file, edge_weight_matrix)


# proceed to analysis
```

```
# very rough example for training/evaluating a classifier
rf = RandomForestClassifier(oob_score = True)
scores = cross_val_score(rf, edge_weights_vec, subject_labels)
```

A fuller example (still a bit rough) for cortical thickness applications can be shown below:

```
#~/usr/bin/env python

import hiwenet
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score
import numpy as np
import nibabel
import os

# -----------------------------------------------------
# toy examples - modify for your application/need
my_project = '/data/myproject'
subject_list = ['a1', 'b2', 'c3', 'd4']
subject_labels = [1, 1, -1, -1]

num_subjects = len(subject_list)
# number of features (imaging vertex-wise cortical thickness values over the whole␣
↪brain)
feature_dimensionality = 1000
num_ROIs = 50
edge_weights = np.empty(num_subjects, num_ROIs*(num_ROIs-1)/2.0)

atlas = 'fsaverage'
# -----------------------------------------------------

def get_parcellation(atlas, parcel_param):
    "Placeholder to insert your own function to return parcellation in reference␣
↪space."

    parc_path = os.path.join(atlas, 'parcellation_param{}.mgh'.format(parcel_param))
    parcel = nibabel.freesurfer.io.read_geometry(parc_path)

    return parcel


groups = get_parcellation(atlas, feature_dimensionality)

out_folder = os.path.join(my_project, 'hiwenet')

# choose a method from one from among the three groups (metrics, semi-metrics and␣
↪similarity functions)
metrics = [ 'manhattan', 'minowski', 'euclidean', 'noelle_2', 'noelle_4', 'noelle_5' ]

semi_metric_list = [
    'kullback_leibler', 'cosine_1',
    'jensen_shannon', 'chi_square',
    'chebyshev', 'chebyshev_neg',
    'histogram_intersection_1',
    'relative_deviation', 'relative_bin_deviation',
    'noelle_1', 'noelle_3',
```

```
    'correlate_1']
similarity_func = ['correlate', 'cosine', 'cosine_2', 'cosine_alt', 'fidelity_based']


def get_features(subject_id):
    "Placeholder to insert your own function to read subject-wise features."

    features_path = os.path.join(my_project,'base_features', subject_id, 'features.txt
↪')
    feature_vector = np.loadtxt(features_path)

    return feature_vector


def upper_tri_vec(matrix):
    "Returns the vectorized values of upper triangular part of a matrix"

    triu_idx = np.triu_indices_from(matrix, 1)
    return matrix[triu_idx]

num_links = num_ROIs*(num_ROIs-1)/2.0
edge_weights_vec = np.zeros(len(subject_list), num_links)
for ss, subject in enumerate(subject_list):
  features = get_features(subject)
  edge_weight_matrix = hiwenet.extract(features, groups,  weight_method = 'kullback_
↪leibler')
  edge_weights_vec[ss,:] = upper_tri_vec(edge_weight_matrix)

  out_file = os.path.join(out_folder, 'hiwenet_{}.txt'.format(subject))
  np.save(out_file, edge_weight_matrix)


# proceed to analysis

# very rough example for training/evaluating a classifier
rf = RandomForestClassifier(oob_score = True)
scores = cross_val_score(rf, edge_weights_vec, subject_labels)
```

CHAPTER 5

Command line interface

The command line interface for hiwenet (although I recommend using it via API) is shown below. Check the bottom of this page for examples.

A rough example of usage can be:

```
#!/bin/bash
#$ -l mf=4G -q abaqus.q -wd /work/project/PBS -j yes -o /work/project/output/job.log
cd /work/project/output
hiwenet -f thickness/features_1000.txt -g thickness/groups_1000.txt -w manhattan -n␣
↪50 -o thickness/hiwenet_manhatten_n50.csv
```

The default behaviour of hiwenet is to trim the outliers, as I suspect their existence in the feature distributions of different ROIs. But if you choose not to do it, you can disable it like this with `-t False` flag:

```
#!/bin/bash
#$ -l mf=4G -q abaqus.q -wd /work/project/PBS -j yes -o /work/project/output/job.log
cd /work/project/output
hiwenet -f thickness/features_1000.txt -g thickness/groups_1000.txt -w manhattan -n␣
↪50 -t False -o thickness/hiwenet_manhatten_n50.csv
```

Typical output can be seen in a file in the *examples* folder, called *pairwise_histogram_dist.csv*, which is shown below, wherein the upper triangular matrix is filled with the corresponding pair-wise distances:

```
0.000,0.903,0.882,0.859,0.865
0.000,0.000,0.910,0.916,0.914
0.000,0.000,0.000,0.902,0.903
0.000,0.000,0.000,0.000,0.945
0.000,0.000,0.000,0.000,0.000
```

CHAPTER 6

Guidelines

How to choose a histogram metric and various parameters associated.

Under construction.

Stay tuned.

CHAPTER 7

Metrics

Definition of the different distances between the histograms are presented here.

I plan to add more families of distances and metrics soon, your comments on what could be good additions will be appreciated, or links to similar repositories that I can link to will be appreciated.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Python Module Index

h

Index