Tutorial of NeuroRA Version 1.1.6

Updated by 2021-12-25

This tutorial of NeuroRA provides information on how to use the NeuroRA. It contains very detailed description of each function in NeuroRA.

Before you read it, you only need to spend a little time learning the basic Python syntax, and this toolkit is very easy to understand. Hope that NeuroRA can help you!

If there is anything wrong, difficult to understand or having any useful advice during reading it, you can contact me (zitonglu1996@gmail.com), and I will be happy and thankful to know about it.

Written by **Zitong Lu**

PhD Student

Cognitive Neuroscience Graduate Program, Department of Psychology

The Ohio State University, Columbus, USA

Vision and Cognitive Neuroscience Lab https://u.osu.edu/golomblab/

Personal Website: https://zitonglu1996.github.io

GitHub Website: https://github.com/ZitongLu1996

This tutorial consists of these parts:

Content

Part 1: Introduction

NeurorRA is a Python toolbox for multimode neural data representational analysis.

Overview

Representational Similarity Analysis (RSA) has become a popular and effective method to measure the representation of multivariable neural activity in different modes.

NeuroRA is a novel and easy-to-use toolbox based on Python, which can do some works about RSA among nearly all kinds of neural data, including behavioral, EEG, MEG, fNIRS, fMRI and some other neuroelectrophysiological data.

In addition, users can conduct Neural Pattern Similarity (NPS), Spatiotemporal Pattern Similarity (STPS), Inter-Subject Correlation (ISC), Classification-based EEG Decoding and a novel cross-temporal RSA (CTRSA) on NeuroRA.

Using NeuroRA, you can realize multiple functions from analysis to statistics to drawing in just a few lines of code.

Installation

- "pip install neurora"

Required Dependencies

Numpy: A fundamental package for scientific computing.
SciPy: A package that provides many user-friendly and efficient numerical routines.
Scikit-learn: An open-source machine learning library.
Matplotlib: A Python 2D plotting library.
NiBabel: A package prividing read +/- write access to some common medical and neuroimaging file formats.
Nilearn: A Python module for fast and easy statistical learning on NeuroImaging data.
Scikit-image: a collection of algorithms for image processing.
MNE-Python: A Python software for exploring, visualizing, and analyzing human neurophysiological data.

Paper

Please cite the paper below while using NeuroRA:

Lu, Z., & Ku, Y. (2020) NeuroRA: A Python toolbox of representational analysis from multi-modal neural data. *Frontiers in Neuroinformatics*. 14:563669. doi: 10.3389/fninf.2020.563669

Part 2: Data Conversion

For EEG/MEG data

Users can use MATLAB toolbox such as EEGLab (sccn.ucsd.edu/eeglab/) to do preprocessing and obtain .mat files, then use SciPy (www.scipy.org) to load EEG data (.mat) as ndarray-type data. Sample codes:

```
>>> import scipy.io as sio
>>> data = sio.loadmat(filename)["data"])
```

Or users can use MNE (mne-tools.github.io) to do preprocessing and return ndarray-type data. Sample codes:

```
>>> # here epoch should be an Epoch object in MNE-Python
>>> data = epoch.get_data()
```

Also, users can use Neo (Garcia et al., 2014) (neuralensemble.org/neo/) for EEG data to do preprocessing and return ndarray-type data. See more detail in Neo io module, and it provides many methods for reading different formats from different EEG acquisition systems.

For fMRI data

We strongly recommend users to use Nibabel (nipy.org/nibabel/) to load fMRI data as ndarray-type data. Sample codes:

```
>>> import nibabel as nib
>>> data = nib.load(fmrifilename).get_fdata()
```

For fNIRS data

For raw data from device, users can use Numpy (www.numpy.org) to load fNIRS data (.txt or .csv) as ndarray-type data. Sample codes:

```
>>> import numpy as np
>>> # load fNIRS data of .txt file as ndarray
>>> data = np.loadtxt(txtfilename)
>>> # load fNIRS data of .csv file as ndarray
>>> data = np.loadtxt(csvfilename, delimiter, usecols, unpack)
```

For some other neuroelectrophysiological data

Users can use Brainstorm (neuroimage.usc.edu/brainstorm/) to do preprocessing and obtain .mat files, then use SciPy to load ECoG data (.mat) as ndarray-type data.

Or users can use Neo (neuralensemble.org/neo/) to do preprocessing and return ndarray-type data. See more detail in Neo io module, and it provides many methods for reading different formats from different neuroelectrophysiology acquisition systems.

Also, users can use pyABF (github.com/swharden/pyABF) for Axon system, to load electrophysiology data (.abf) as ndarray-type data. Sample codes:

```
>>> import pyabf
>>> # the electrophysiology data file name with full address
>>> abf = pyabf.ABF("demo.abf")
>>> # access sweep data
>>> abf.setSweep(sweepNumber, channel)
>>> # get sweep data with sweepY
>>> data = abf.sweepY
```

Notes

Two functions, *NumPy.reshape*() & *NumPy.transpose*(), are recommended for further data transformation.

Part 3: To calculate the Neural Pattern Similarity

Module: *nps_cal.py*

- a module for calculating the neural pattern similarity based on neural data

'a function for calculating the neural pattern similarity for EEG-like data'

- **nps(data, time_win=5, time_step=5, sub_opt=1)**

Parameters

data : array
The EEG-like neural data.
The shape of data must be [2, n_subs, n_trials, n_chls, n_ts]. 2 presents 2 different conditions. n_subs, n_trials, n_chls & n_ts represent the number of subjects, the number of trials, the number of channels & the number of time-points, respectively.

time_win : int. Default is 5.
Set a time-window for calculating the NPS for different time-points.
If time_win=5, that means each calculation process based on 5 time-points.

time_step : int. Default is 5.
The time step size for each time of calculating.

sub_opt : int 0 or 1. Default is 1.
Calculate the NPS for each subject or not.
If sub_opt=0, calculate the NPS based on all data. If sub_opt=1, calculate the NPS based on each subject's data

Returns

nps : array
The EEG-like NPS.
If sub_opt=0, the shape of NPS is [n_chls, int((n_ts-time_win)/time_step)+1, 2]. If sub_opt=1, the shape of NPS is [n_subs, n_chls, int((n_ts-time_win)/time_step)+1, 2]. 2 representation a r-value and a p-value.

'a function for calculating the neural pattern similarity for fMRI data (searchlight)'

- **nps_fmri(fmri_data, ksize=[3, 3, 3], strides=[1, 1, 1])**

Parameters

fmri_data : array
The fmri data.

The shape of fmri_data must be [2, n_subs, nx, ny, nz]. 2 presents 2 different conditions. nx, ny, nz represent the size of fMRI-img, respectively.

ksize : array or list [kx, ky, kz]. Default is [3, 3, 3].
The size of the calculation unit for searchlight.
kx, ky, kz represent the number of voxels along the x, y, z axis. kx, ky, kz should be odd.

strides : array or list [sx, sy, sz]. Default is [1, 1, 1].
The strides for calculating along the x, y, z axis.

Returns

nps : array
The fMRI NPS for searchlight.
The shape of NPS is [n_subs, n_x, n_y, n_z, 2]. n_subs, n_x, n_y, n_z represent the number of subjects, the number of calculation units for searchlight along the x, y, z axis. 2 represent a r-value and a p-value.

Notes

The size of the calculation units should at least be [3, 3, 3].

'a function for calculating the neural pattern similarity for fMRI data (for ROI)'

- **nps_fmri_roi(fmri_data, mask_data)**

Parameters

fmri_data : array
The fmri data.
The shape of fmri_data must be [2, n_subs, nx, ny, nz]. 2 presents 2 different conditions. n_subs, nx, ny, nz represent the number of channels & the size of fMRI-img, respectively.

mask_data : array [nx, ny, nz].
The mask data for region of interest (ROI)
The size of the fMRI-img. nx, ny, nz represent the number of voxels along the x, y, z axis

Returns

subNPS : array
The fMRI NPS for ROI.
The shape of NPS is [n_subs, 2]. n_subs represents the number of subjects. 2 represents a r-value and a p-value.

Notes

The size of the calculation units should at least be [3, 3, 3].

Part 4: To calculate the Spatiotemporal Pattern Similarity

Module: *stps_cal.py*

- a module for calculating the spatiotemporal pattern similarity based on neural data

'a function for calculating the spatiotemporal pattern similarities (STPS)'

- **stps(data, label_item, label_rf, time_win=20, time_step=1)**

Parameters

data : array
The neural data.
The shape of data must be [n_subs, n_trials, n_chls, n_ts]. n_subs, n_trials, n_chls and n_ts represent the number of subjects, the number of trials, the number of channels or regions and the number of time-points.

label_item : array or list.
The label of trials.
The shape of label_wibi must be [n_trials]. n_trials represents the number of trials.

label_rf : array or list.
The label of trials: Remembered (0) or Forgot (1).
The shape of label_rf must be [n_trials]. n_trials represents the number of trials. If the trial i is a remembered trial, label_rf[i]=0. If the trial j is a forgot trial, label_rf[j]=0.

time_win : int. Default is 20.
Set a time-window for calculating the STPS for different time-points.
If time_win=20, that means each calculation process based on 20 time-points.

time_step : int. Default is 1.
The time step size for each time of calculating.

Returns

stps : array.
The STPS.
The shape of stps is [n_subs, 8, n_chls, int((n_ts-time_win)/time_step)+1]. 8 represents eight different conditions: 0: Within-Item, 1: Between-Item, 2: Remembered, 3: Forgot, 4: Within-Item&Remembered, 5: Within-Item&Forgot, 6: Between-Item&Remembered, 7: Between-Item&Forgot.

'a function for calculating the spatiotemporal pattern similarities (STPS) for fMRI

(searchlight)'

- **stps_fmri(fmri_data, label_item, label_rf, ksize=[3, 3, 3], strides=[1, 1, 1])**

Parameters

fmri_data : array
The fMRI data.
The shape of fmri_data must be [n_subs, n_trials, nx, ny, nz]. n_subs, n_trials, nx, ny, nz represent the number of subjects, the number of trials & the size of fMRI-img, respectively.

label_item : array or list.
The label of trials.
The shape of label_item must be [n_trials]. n_trials represents the number of trials.

label_rf : array or list.
The label of trials: Remembered (0) or Forgot (1).
The shape of label_rf must be [n_trials]. n_trials represents the number of trials. If the trial i is a remembered trial, label_rf[i]=0. If the trial j is a forgot trial, label_rf[j]=0.

ksize : array or list [kx, ky, kz]. Default is [3, 3, 3].
The size of the calculation unit for searchlight
kx, ky, kz represent the number of voxels along the x, y, z axis. kx, ky, kz should be odd.

strides : array or list [sx, sy, sz]. Default is [1, 1, 1].
The strides for calculating along the x, y, z axis.

Returns

stps : array.
The STPS.
The shape of stps is [n_subs, 8, n_x, n_y, n_z]. 8 represents eight different conditions: 0: Within-Item, 1: Between-Item, 2: Remembered, 3: Forgot, 4: Within-Item&Remembered, 5: Within-Item&Forgot, 6: Between-Item&Remembered, 7: Between-Item&Forgot. n_x, n_y, n_z represent the number of calculation units for searchlight along the x, y, z axis.

Notes

The size of the calculation units should at least be [3, 3, 3].

'a function for calculating the spatiotemporal pattern similarities (STPS) for fMRI (for ROI)'

- **stps_fmri_roi(fmri_data, mask_data, label_item, label_rf)**

Parameters

fmri_data : array
The fmri data.
The shape of fmri_data must be [n_subs, n_trials, nx, ny, nz]. n_subs, n_trials, nx, ny, nz represent the number of subjects, the number of trials

& the size of fMRI-img, respectively.

mask_data : array [nx, ny, nz].
The mask data for region of interest (ROI).
The size of the fMRI-img. nx, ny, nz represent the number of voxels along the x, y, z axis.

label_item : array or list.
The label of trials.
The shape of label_wibi must be [n_trials]. n_trials represents the number of trials.

label_rf : array or list.
The label of trials: Remembered (0) or Forgot (1).
The shape of label_rf must be [n_trials]. n_trials represents the number of trials. If the trial i is a emembered trial, label_rf[i]=0. If the trial j is a forgot trial, label_rf[j]=0.

Returns

stps : array.
The STPS.
The shape of stps is [n_subs, 8, n_x, n_y, n_z]. 8 represents eight different conditions: 0: Within-Item, 1: Between-Item, 2: Remembered, 3: Forgot, 4: Within-Item&Remembered, 5: Within-Item&Forgot, 6: Between-Item&Remembered, 7: Between-Item&Forgot. n_x, n_y, n_z represent the number of calculation units for searchlight along the x, y, z axis.

Notes

The size of the calculation units should at least be [3, 3, 3].

Part 5: To calculate the Inter-Subject Correlation

Module: *isc_cal.py*

- a module for calculating the inter-subject correlation based on neural data

'a function for calculating the inter subject correlation (ISC)'

- **isc(data, time_win=5, time_step=5)**

Parameters

data : array
The neural data.
The shape of data must be [n_subs, n_chls, n_ts]. n_subs, n_chls, n_ts represent the number of subjects, the number of channels and the number of time-points.

time_win : int. Default is 5.
Set a time-window for calculating the STPS for different time-points.
If time_win=5, that means each calculation process based on 5 time-points.

time_step : int. Default is 5.
The time step size for each time of calculating.

Returns

isc : array
The ISC.
The shape of isc is [n_subs!/(2!(n_subs-2)!), n_chls, int((n_ts-time_win)/time_step)+1, 2]. n_subs, n_chls, n_ts represent the number of subjects, the number of channels and the number of time-points. 2 represents a r-value and a p-value.*

Notes

In ISC, correlation computing process will be done for each pair of subjects.

'a function for calculating the inter subject correlation (ISC) for fMRI (searchlight)'

- **isc_fmri(fmri_data, ksize=[3, 3, 3], strides=[1, 1, 1])**

Parameters

fmri_data : array
The fmri data.
The shape of fmri_data must be [n_ts, n_subs, nx, ny, nz]. n_ts,

nx, ny, nz represent the number of time-points, the number of subs & the size of fMRI-img, respectively.

ksize : array or list [kx, ky, kz]. Default is [3, 3, 3].
The size of the calculation unit for searchlight.
kx, ky, kz represent the number of voxels along the x, y, z axis. kx, ky, kz should be odd.

strides : array or list [sx, sy, sz]. Default is [1, 1, 1].
The strides for calculating along the x, y, z axis.

Returns

isc : array
The ISC.
The shape of isc is [n_ts, n_subs!/(2!(n_subs-2)!), n_x, n_y, n_z, 2]. n_ts, n_subs, n_x, n_y, n_z represent the number of time-points, the number of subjects, the number of calculation units for searchlight along the x, y, z axis. 2 represent a r-value and a p-value.*

Notes

The size of the calculation units should at least be [3, 3, 3].
In ISC, correlation computing process will be done for each pair of subjects.

'a function for calculating the inter subject correlation (ISC) for fMRI (ROI)'

- **isc_fmri_roi(fmri_data, mask_data)**

Parameters

fmri_data : array
The fmri data.
The shape of fmri_data must be [n_ts, n_subs, nx, ny, nz]. n_ts, nx, ny, nz represent the number of time-points, the number of subs & the size of fMRI-img, respectively.

mask_data : array [nx, ny, nz].
The mask data for region of interest (ROI).
The size of the fMRI-img. nx, ny, nz represent the number of voxels along the x, y, z axis.

Returns

isc : array
The ISC.
The shape of corrs is [n_ts, n_subs!/(2!(n_subs-2)!), 2]. n_ts, n_subs represent the number of time-points, the number of subjects. 2 represent a r-value and a p-value.*

Notes

In ISC, correlation computing process will be done for each pair of subjects.

Part 6: To calculate the Representational Dissimilarity Matrix

Module: *rdm_cal.py*

- a module for calculating the RDM based on multimode neural data

'a function for calculating the RDM(s) based on behavioral data'

- **bhvRDM(bhv_data, sub_opt=1, method="correlation", abs=False)**

Parameters

bhv_data : array
The behavioral data.
The shape of bhv_data must be [n_cons, n_subs, n_trials]. n_cons, n_subs & n_trials represent the number of conidtions, the number of subjects & the number of trials, respectively.

sub_opt : int 0 or 1. Default is 1.
Return the results for each subject or after averaging.
If sub_opt=1, calculate the results of each subject (using the absolute distance). If sub_opt=0, calculate the results averaging the trials and taking the subjects as the features.

method : string 'correlation' or 'euclidean'. Default is 'correlation'.
The method to calculate the dissimilarities.
If method='correlation', the dissimilarity is calculated by Pearson Correlation. If method='euclidean', the dissimilarity is calculated by Euclidean Distance, the results will be normalized.

abs : boolean True or False. Default is True.
Calculate the absolute value of Pearson r or not.
Only works when method='correlation'.

Returns

RDM(s) : array
The behavioral RDM.
If sub_opt=1, return n_subs RDMs. The shape is [n_subs, n_cons, n_cons]. If sub_opt=0, return only one RDM. The shape is [n_cons, n_cons].

Notes

This function can also be used to calculate the RDM for computational simulation data. For example, users can extract the activations for a certain layer i which includes Nn nodes in a deep convolutional neural network (DCNN) corresponding to Ni images. Thus, the input could be a [Ni, Nn, 1] matrix, M.

Using "bhvRDM(M, sub_opt=0)", users can obtain the DCNN RDM for layer i.

'a function for calculating the RDM(s) based on EEG/MEG/fNIRS & other EEG-like data'

- **eegRDM(EEG_data, sub_opt=1, chl_opt=0, time_opt=0, time_win=5, time_step=5, method="correlation", abs=False)**

Parameters

EEG_data : array
The EEG/MEG/fNIRS data.
The shape of EEGdata must be [n_cons, n_subs, n_trials, n_chls, n_ts]. n_cons, n_subs, n_trials, n_chls & n_ts represent the number of conidtions, the number of subjects, the number of trials, the number of channels & the number of time-points, respectively.

sub_opt: int 0 or 1. Default is 1.
Return the subject-result or average-result.
If sub_opt=0, return the average result. If sub_opt=1, return the results of each subject.

chl_opt : int 0 or 1. Default is 0.
Calculate the RDM for each channel or not.
If chl_opt=0, calculate the RDM based on all channels'data. If chl_opt=1, calculate the RDMs based on each channel's data respectively.

time_opt : int 0 or 1. Default is 0.
Calculate the RDM for each time-point or not
If time_opt=0, calculate the RDM based on whole time-points' data. If time_opt=1, calculate the RDMs based on each time-points respectively.

time_win : int. Default is 5.
Set a time-window for calculating the RDM for different time-points.
Only when time_opt=1, time_win works. If time_win=5, that means each calculation process based on 5 time-points.

time_step : int. Default is 5.
The time step size for each time of calculating.
Only when time_opt=1, time_step works.

method : string 'correlation' or 'euclidean'. Default is 'correlation'.
The method to calculate the dissimilarities.
If method='correlation', the dissimilarity is calculated by Pearson Correlation. If method='euclidean', the dissimilarity is calculated by Euclidean Distance, the results will be normalized.

abs : boolean True or False. Default is True.
Calculate the absolute value of Pearson r or not.

Returns

RDM(s) : array

The EEG/MEG/fNIR/other EEG-like RDM.

If sub_opt=0 & chl_opt=0 & time_opt=0, return only one RDM. The shape is [n_cons, n_cons]. If sub_opt=0 & chl_opt=0 & time_opt=1, return int((n_ts-time_win)/time_step)+1 RDM. The shape is [int((n_ts-time_win)/time_step)+1, n_cons, n_cons]. If sub_opt=0 & chl_opt=1 & time_opt=0, return n_chls RDM. The shape is [n_chls, n_cons, n_cons]. If sub_opt=0 & chl_opt=1 & time_opt=1, return n_chls(int((n_ts-time_win)/time_step)+1) RDM. The shape is [n_chls, int((n_ts-time_win)/time_step)+1, n_cons, n_cons]. If sub_opt=1 & chl_opt=0 & time_opt=0, return n_subs RDM. The shape is [n_subs, n_cons, n_cons]. If sub_opt=1 & chl_opt=0 & time_opt=1, return n_subs*(int((n_ts-time_win)/time_step)+1) RDM. The shape is [n_subs, int((n_ts-time_win)/time_step)+1, n_cons, n_cons]. If sub_opt=1 & chl_opt=1 & time_opt=0, return n_subs*n_chls RDM. The shape is [n_subs, n_chls, n_cons, n_cons]. If sub_opt=1 & chl_opt=1 & time_opt=1, return n_subs*n_chls*(int((n_ts-time_win)/time_step)+1) RDM. The shape is [n_subs, n_chls, int((n_ts-time_win)/time_step)+1, n_cons, n_cons].*

Notes

Sometimes, the numbers of trials under different conditions are not same. In NeuroRA, we recommend users to average the trials under a same condition firstly in this situation. Thus, the shape of input (EEG_data) should be [n_cons, n_subs, 1, n_chls, n_ts].

'a function for calculating the RDM(s) using classification-based neural decoding based on EEG/MEG/fNIRS & other EEG-like data'

- **eegRDM_bydecoding(EEG_data, sub_opt=1, time_win=5, time_step=5, navg=5, time_opt="average", nfolds=5, nrepeats=2, normalization=False)**

Parameters

EEG_data : array

The EEG/MEG/fNIRS data.

The shape of EEGdata must be [n_cons, n_subs, n_trials, n_chls, n_ts]. n_cons, n_subs, n_trials, n_chls & n_ts represent the number of conidtions, the number of subjects, the number of trials, the number of channels & the number of time-points, respectively.

sub_opt: int 0 or 1. Default is 1.

Return the subject-result or average-result.

If sub_opt=0, return the average result. If sub_opt=1, return the results of each subject.

time_win : int. Default is 5.

Set a time-window for calculating the RDM for different time-points.

Only when time_opt=1, time_win works.

If time_win=5, that means each calculation process based on 5 time-points.

time_step : int. Default is 5.
The time step size for each time of calculating.
Only when time_opt=1, time_step works.

navg : int. Default is 5.
The number of trials used to average.

time_opt : string "average" or "features". Default is "average".
Average the time-points or regard the time points as features for classification
If time_opt="average", the time-points in a certain time-window will be averaged.
If time_opt="features", the time-points in a certain time-window will be used as features for classification.

nfolds : int. Default is 5.
The number of folds.
k should be at least 2.

nrepeats : int. Default is 2.
The times for iteration.

normalization : boolean True or False. Default is False.
Normalize the data or not.

Returns

RDM(s) : array
The EEG/MEG/fNIR/other EEG-like RDM.
*If sub_opt=0, return int((n_ts-time_win)/time_step)+1 RDMs. The shape is [int((n_ts-time_win)/time_step)+1, n_cons, n_cons]. If sub_opt=1, return n_subs*int((n_ts-time_win)/time_step)+1 RDM. The shape is [n_subs, int((n_ts-time_win)/time_step)+1, n_cons, n_cons].*

Notes

Sometimes, the numbers of trials under different conditions are not same. In NeuroRA, we recommend users to sample randomly from the trials under each condition to keep the numbers of trials under different conditions same, and you can iterate multiple times.

'a function for calculating the RDMs based on fMRI data (searchlight)'

- **fmriRDM(fmri_data, ksize=[3, 3, 3], strides=[1, 1, 1], sub_opt=1, method="correlation", abs=False)**

Parameters

fmri_data : array
The fmri data.
The shape of fmri_data must be [n_cons, n_subs, nx, ny, nz]. n_cons, nx, ny, nz represent the number of onditions, the number of subs & the size

of fMRI-img, respectively.

ksize : array or list [kx, ky, kz]. Default is [3, 3, 3].
The size of the calculation unit for searchlight.
kx, ky, kz represent the number of voxels along the x, y, z axis. kx, ky, kz should be odd.

strides : array or list [sx, sy, sz]. Default is [1, 1, 1].
The strides for calculating along the x, y, z axis.

sub_opt: int 0 or 1. Default is 1.
Return the subject-result or average-result.
If sub_opt=0, return the average result. If sub_opt=1, return the results of each subject.

method : string 'correlation' or 'euclidean'. Default is 'correlation'.
The method to calculate the dissimilarities.
If method='correlation', the dissimilarity is calculated by Pearson Correlation. If method='euclidean', the dissimilarity is calculated by Euclidean Distance, the results will be normalized.

abs : boolean True or False. Default is True.
Calculate the absolute value of Pearson r or not.

Returns

RDM : array
The fMRI-Searchlight RDM.
If sub_opt=0, the shape of RDMs is [n_x, n_y, n_z, n_cons, n_cons]. If sub_opt=1, the shape of RDMs is [n_subs, n_x, n_y, n_cons, n_cons] n_subs, n_x, n_y, n_z represent the number of subjects & the number of calculation units for searchlight along the x, y, z axis.

'a function for calculating the RDM based on fMRI data of an ROI'

- **fmriRDM_roi(fmri_data, mask_data, sub_opt=1, method="correlation", abs=False)**

Parameters

fmri_data : array
The fmri data.
The shape of fmri_data must be [n_cons, n_subs, nx, ny, nz]. n_cons, nx, ny, nz represent the number of conditions, the number of subs & the size of fMRI-img, respectively.

mask_data : array [nx, ny, nz].
The mask data for region of interest (ROI)
The size of the fMRI-img. nx, ny, nz represent the number of voxels along the x, y, z axis.

sub_opt: int 0 or 1. Default is 1.
Return the subject-result or average-result.
If sub_opt=0, return the average result. If sub_opt=1, return

the results of each subject.

method : string 'correlation' or 'euclidean' or 'mahalanobis'. Default is 'correlation'.

The method to calculate the dissimilarities.

If method='correlation', the dissimilarity is calculated by Pearson Correlation. If method='euclidean', the dissimilarity is calculated by Euclidean Distance, the results will be normalized.

abs : boolean True or False. Default is True.

Calculate the absolute value of Pearson r or not.

Returns

RDM : array

The fMRI-ROI RDM.

If sub_opt=0, the shape of RDM is [n_cons, n_cons]. If sub_opt=1, the shape of RDM is [n_subs, n_cons, n_cons].

Notes

The sizes (nx, ny, nz) of fmri_data and mask_data should be same.

Part 7: To calculate the Cross-Temporal Representational Dissimilarity Matrix

Module: *ctrdm_cal.py*

- a module for calculating the cross-temporal RDM based on EEG-like data

'a function for calculating the cross-temporal RDM(s) based on EEG-like data'

- **ctRDM(data, sub_opt=1, chl_opt=0, time_opt=0, time_win=5, time_step=5)**

Parameters

data : array
EEG/MEG/fNIRS data from a time window.
The shape of data must be [n_cons, n_subs, n_chls, n_ts]. n_cons, n_subs, n_chls & n_ts represent the number of conidtions, the number of subjects, the number of channels & the number of time-points, respectively.

sub_opt: int 0 or 1. Default is 1.
Return the subject-result or average-result.
If sub_opt=0, return the average result. If sub_opt=1, return the results of each subject.

chl_opt : int 0 or 1. Default is 0.
Calculate the CTRDMs for each channel or not.
If chl_opt=1, calculate the CTRDMs for each channel. If chl_opt=0, calculate the CTRDMs after averaging the channels.

time_win : int. Default is 5.
Set a time-window for calculating the CTRDM for different time-points.
Only when time_opt=1, time_win works. If time_win=5, that means each calculation process based on 5 time-points.

time_step : int. Default is 5.
The time step size for each time of calculating.

Returns

CTRDM(s) : array
Cross-temporal RDMs.
If chl_opt=1, the shape of CTRDMs is [n_subs, n_chls, int((n_ts-time_win)/time_step)+1, int((n_ts-time_win)/time_step)+1, n_cons, n_cons]. If chl_opt=0, the shape of CTRDMs is [n_subs, n_chls, int((n_ts-time_win)/time_step)+1, int((n_ts-time_win)/time_step)+1, n_cons, n_cons].

Part 8: To Conduct Representational Similarity Analysis

Module: *rdm_corr.py*

- a module for calculating the Similarity/Correlation Coefficient between two RDMs

'a function for calculating the Spearman correlation coefficient between two RDMs'

- **rdm_correlation_spearman(RDM1, RDM2, rescale=False, permutation=False, iter=1000)**

Parameters

RDM1 : array [ncons, ncons]
The RDM 1.
The shape of RDM1 must be [n_cons, n_cons]. n_cons represent the number of conidtions.

RDM2 : array [ncons, ncons].
The RDM 2.
The shape of RDM2 must be [n_cons, n_cons]. n_cons represent the number of conidtions.

rescale : bool True or False. Default is False.
Rescale the values in RDM or not.
Here, the maximum-minimum method is used to rescale the values except for the values on the diagonal.

permutation : bool True or False. Default is False.
Conduct permutation test or not.

iter : int. Default is 1000.
The times for iteration.

Returns

corr : array [r, p].
The Spearman Correlation result.
The shape of corr is [2], including a r-value and a p-value.

'a function for calculating the Pearson correlation coefficient between two RDMs'

- **rdm_correlation_pearson(RDM1, RDM2, rescale=False, permutation=False, iter=1000)**

Parameters

RDM1 : array [ncons, ncons]

The RDM 1.

The shape of RDM1 must be [n_cons, n_cons]. n_cons represent the number of conidtions.

RDM2 : array [ncons, ncons].

The RDM 2.

The shape of RDM2 must be [n_cons, n_cons]. n_cons represent the number of conidtions.

rescale : bool True or False. Default is False.

Rescale the values in RDM or not.

Here, the maximum-minimum method is used to rescale the values except for the values on the diagonal.

permutation : bool True or False. Default is False.

Conduct permutation test or not.

iter : int. Default is 1000.

The times for iteration.

Returns

corr : array [r, p].

The Pearson Correlation result.

The shape of corr is [2], including a r-value and a p-value.

'a function for calculating the Kendalls tau correlation coefficient between two RDMs'

- **rdm_correlation_kendall(RDM1, RDM2, rescale=False, permutation=False, iter=1000)**

Parameters

RDM1 : array [ncons, ncons]

The RDM 1.

The shape of RDM1 must be [n_cons, n_cons]. n_cons represent the number of conidtions.

RDM2 : array [ncons, ncons].

The RDM 2.

The shape of RDM2 must be [n_cons, n_cons]. n_cons represent the number of conidtions.

rescale : bool True or False. Default is False.

Rescale the values in RDM or not.

Here, the maximum-minimum method is used to rescale the values except for the values on the diagonal.

permutation : bool True or False. Default is False.

Conduct permutation test or not.

iter : int. Default is 1000.

The times for iteration.

Returns

corr : array [r, p].
The Kendall tau Correlation result.
The shape of corr is [2], including a r-value and a p-value.

'a function for calculating the Cosine Similarity between two RDMs'

- **rdm_similarity(RDM1, RDM2, rescale=False)**

Parameters

RDM1 : array [ncons, ncons]
The RDM 1.
The shape of RDM1 must be [n_cons, n_cons]. n_cons represent the number of conidtions.

RDM2 : array [ncons, ncons].
The RDM 2.
The shape of RDM2 must be [n_cons, n_cons]. n_cons represent the number of conidtions.

rescale : bool True or False. Default is False.
Rescale the values in RDM or not.
Here, the maximum-minimum method is used to rescale the values except for the values on the diagonal.

Returns

similarity : float.
The Cosine Similarity result.

'a fuction for calculating the Euclidean Distance between two RDMs'

- **rdm_distance(RDM1, RDM2, rescale=False)**

Parameters

RDM1 : array [ncons, ncons]
The RDM 1.
The shape of RDM1 must be [n_cons, n_cons]. n_cons represent the number of conidtions.

RDM2 : array [ncons, ncons].
The RDM 2.
The shape of RDM2 must be [n_cons, n_cons]. n_cons represent the number of conidtions.

rescale : bool True or False. Default is False.
Rescale the values in RDM or not.
Here, the maximum-minimum method is used to rescale the values except for the values on the diagonal.

Returns

dist : float
The Euclidean Distance result.

Module: *corr_cal.py*

- a module for calculating the similarity between two different modes' data

'a function for calculating the similarity between behavioral data and EEG-like data'

- **bhvANDeeg_corr(bhv_data, eeg_data, sub_opt=1, chl_opt=0, time_opt=0, time_win=5, time_step=5, method="spearman", rescale=False, permutation=False, iter=1000)**

Parameters

bhv_data : array
The behavioral data.
The shape of bhv_data must be [n_cons, n_subs, n_trials]. n_cons, n_subs & n_trials represent the number of conidtions, the number of subjects & the number of trials, respectively.

eeg_data : array
The EEG/MEG/fNIRS data.
The shape of EEGdata must be [n_cons, n_subs, n_trials, n_chls, n_ts]. n_cons, n_subs, n_trials, n_chls & n_ts represent the number of conidtions, the number of subjects, the number of trials, the number of channels & the number of time-points, respectively.

sub_opt : int 0 or 1. Default is 0.
Calculate the RDM & similarities for each subject or not.
If sub_opt=0, calculating based on all data. If sub_opt=1, calculating based on each subject's data, respectively.

chl_opt : int 0 or 1. Default is 0.
Calculate the RDM & similarities for each channel or not.
If chl_opt=0, calculating based on all channels' data. If chl_opt=1, calculating based on each channel's data respectively.

time_opt : int 0 or 1. Default is 0.
Calculate the RDM & similarities for each time-point or not
If time_opt=0, calculating based on whole time-points' data. If time_opt=1, calculating based on each time-points respectively.

time_win : int. Default is 5.
Set a time-window for calculating the RDM & similarities for different time-points.
Only when time_opt=1, time_win works. If time_win=5, that means each calculation process based on 5 time-points.

time_step : int. Default is 5.
The time step size for each time of calculating.
Only when time_opt=1, time_step works.

method : string 'spearman' or 'pearson' or 'kendall' or 'similarity' or 'distance'. Default is 'spearman'.
The method to calculate the similarities.
If method='spearman', calculate the Spearman Correlations. If method='pearson', calculate the Pearson Correlations. If methd='kendall', calculate the Kendall tau Correlations. If method='similarity', calculate the Cosine Similarities. If method='distance', calculate the Euclidean Distances.

rescale : bool True or False.
Rescale the values in RDM or not.
Here, the maximum-minimum method is used to rescale the values except for the values on the diagonal.

permutation : bool True or False. Default is False.
Use permutation test or not.

iter : int. Default is 5000.
The times for iteration.

Returns

corrs : array
The similarities between behavioral data and EEG/MEG/fNIRS data.
If sub_opt=0 & chl_opt=0 & time_opt=0, return one corr result. The shape of corrs is [2], a r-value and a p-value. If method='similarity' or method='distance', the p-value is 0. If sub_opt=0 & chl_opt=0 & time_opt=1, return int((n_ts-time_win)/time_step)+1 corrs result. The shape of corrs is [int((n_ts-time_win)/time_step)+1, 2]. 2 represents a r-value and a p-value. If method='similarity' or method='distance', the p-values are all 0. If sub_opt=0 & chl_opt=1 & time_opt=0, return n_chls corrs result. The shape of corrs is [n_chls, 2]. 2 represents a r-value and a p-value. If method='similarity' or method='distance', the p-values are all 0. If sub_opt=0 & chl_opt=1 & time_opt=1, return n_chls(int((n_ts-time_win)/time_step)+1) corrs result. The shape of corrs is [n_chls, int((n_ts-time_win)/time_step)+1, 2]. 2 represents a r-value and a p-value. If method='similarity' or method='distance', the p-values are all 0. If sub_opt=1 & chl_opt=0 & time_opt=0, return n_subs corr result. The shape of corrs is [n_subs, 2], a r-value and a p-value. If method='similarity' or method='distance', the p-values are all 0. If sub_opt=1 & chl_opt=0 & time_opt=1, return n_subs*(int((n_ts-time_win)/time_step)+1) corrs result. The shape of corrs is [n_subs, int((n_ts-time_win)/time_step)+1, 2]. 2 represents a r-value and a p-value. If method='similarity' or method='distance', the p-values are all 0. If sub_opt=1 & chl_opt=1 & time_opt=0, return n_subs*n_chls corrs result. The shape of corrs is [n_subs, n_chls, 2]. 2 represents a r-value and a p-value. If method='similarity' or method='distance', the p-values are all 0. If sub_opt=1 & chl_opt=1 & time_opt=1, return n_subs*n_chls*(int((n_ts-time_win)/time_step)+1) corrs result. The shape*

of corrs is [n_subs, n_chls, int((n_ts-time_win)/time_step)+1, 2]. 2 represents a r-value and a p-value. If method='similarity' or method='distance', the p-values are all 0.

'a function for calculating the similarity between behavioral data and fMRI data (searchlight)'

- **bhvANDfmri_corr(bhv_data, fmri_data, ksize=[3, 3, 3], strides=[1, 1, 1], sub_opt=1, method="spearman", rescale=False, permutation=False, iter=1000)**

Parameters

bhv_data : array
The behavioral data.
The shape of bhv_data must be [n_cons, n_subs, n_trials]. n_cons, n_subs & n_trials represent the number of conidtions, the number of subjects & the number of trials, respectively.

fmri_data : array
The fmri data.
The shape of fmri_data must be [n_cons, n_subs, nx, ny, nz]. n_cons, nx, ny, nz represent the number of conditions, the number of subs & the size of fMRI-img, respectively. ksize : array or list [kx, ky, kz]. Default is [3, 3, 3]. The size of the calculation unit for searchlight. kx, ky, kz represent the number of voxels along the x, y, z axis. kx, ky, kz should be odd.

strides : array or list [sx, sy, sz]. Default is [1, 1, 1].
The strides for calculating along the x, y, z axis.

sub_opt: int 0 or 1. Default is 1.
Return the subject-result or average-result.
If sub_opt=0, return the average result. If sub_opt=1, return the results of each subject.

method : string 'spearman' or 'pearson' or 'kendall' or 'similarity' or 'distance'. Default is 'spearman'.
The method to calculate the similarities.
If method='spearman', calculate the Spearman Correlations. If method='pearson', calculate the Pearson Correlations. If methd='kendall', calculate the Kendall tau Correlations. If method='similarity', calculate the Cosine Similarities. If method='distance', calculate the Euclidean Distances.

rescale : bool True or False.
Rescale the values in RDM or not.
Here, the maximum-minimum method is used to rescale the values except for the values on the diagonal.

permutation : bool True or False. Default is False.
Use permutation test or not.

iter : int. Default is 1000.

The times for iteration.

Returns

corrs : array

The similarities between behavioral data and fMRI data for searchlight.

If sub_result=0, the shape of corrs is [n n_x, n_y, n_z represent the number of calculation units for searchlight along the x, y, z axis and 2 represents a r-value and a p-value.

'a function for calculating the similarity between behavioral data and fMRI data (searchlight)'

- **bhvANDfmri_corr(bhv_data, fmri_data, ksize=[3, 3, 3], strides=[1, 1, 1], sub_opt=1, method="spearman", rescale=False, permutation=False, iter=1000)**

Parameters

bhv_data : array

The behavioral data.

The shape of bhv_data must be [n_cons, n_subs, n_trials]. n_cons, n_subs & n_trials represent the number of conidtions, the number of subjects & the number of trials, respectively.

fmri_data : array

The fmri data.

The shape of fmri_data must be [n_cons, n_subs, nx, ny, nz]. n_cons, nx, ny, nz represent the number of conditions, the number of subs & the size of fMRI-img, respectively. ksize : array or list [kx, ky, kz]. Default is [3, 3, 3]. The size of the calculation unit for searchlight. kx, ky, kz represent the number of voxels along the x, y, z axis. kx, ky, kz should be odd.

strides : array or list [sx, sy, sz]. Default is [1, 1, 1].

The strides for calculating along the x, y, z axis.

sub_opt: int 0 or 1. Default is 1.

Return the subject-result or average-result.

If sub_opt=0, return the average result. If sub_opt=1, return the results of each subject.

method : string 'spearman' or 'pearson' or 'kendall' or 'similarity' or 'distance'. Default is 'spearman'.

The method to calculate the similarities.

If method='spearman', calculate the Spearman Correlations. If method='pearson', calculate the Pearson Correlations. If methd='kendall', calculate the Kendall tau Correlations. If method='similarity', calculate the Cosine Similarities. If method='distance', calculate the Euclidean Distances.

rescale : bool True or False.

Rescale the values in RDM or not.

Here, the maximum-minimum method is used to rescale the values except for the values on the diagonal.

permutation : bool True or False. Default is False.
Use permutation test or not.

iter : int. Default is 1000.
The times for iteration.

Returns

corrs : array
The similarities between behavioral data and fMRI data for searchlight.

If sub_result=0, the shape of corrs is [n n_x, n_y, n_z represent the number of calculation units for searchlight along the x, y, z axis and 2 represents a r-value and a p-value.

'a function for calculating the similarity EEG-like data and fMRI data (searchlight)'

- **eegANDfmri_corr(eeg_data, fmri_data, chl_opt=0, ksize=[3, 3, 3], stride=[1, 1, 1], sub_opt=1, method="spearman", rescale=False, permutation=False, iter=1000)**

Parameters

eeg_data : array
The EEG/MEG/fNIRS data.

The shape of EEGdata must be [n_cons, n_subs, n_trials, n_chls, n_ts]. n_cons, n_subs, n_trials, n_chls & n_ts represent the number of conidtions, the number of subjects, the number of trials, the number of channels & the number of time-points, respectively.

fmri_data : array
The fmri data.

The shape of fmri_data must be [n_cons, n_subs, nx, ny, nz]. n_cons, nx, ny, nz represent the number of conditions, the number of subs & the size of fMRI-img, respectively.

chl_opt : int 0 or 1. Default is 0.
Calculate the RDM & similarities for each channel or not.

If chl_opt=0, calculating based on all channels' data. If chl_opt=1, calculating based on each channel's data respectively.

ksize : array or list [kx, ky, kz]. Default is [3, 3, 3].
The size of the calculation unit for searchlight.

kx, ky, kz represent the number of voxels along the x, y, z axis. kx, ky, kz should be odd.

strides : array or list [sx, sy, sz]. Default is [1, 1, 1].
The strides for calculating along the x, y, z axis.

sub_opt: int 0 or 1. Default is 1.

Return the subject-result or average-result.

If sub_opt=0, return the average result. If sub_opt=1, return the results of each subject.

method : string 'spearman' or 'pearson' or 'kendall' or 'similarity' or 'distance'. Default is 'spearman'.

The method to calculate the similarities.

If method='spearman', calculate the Spearman Correlations. If method='pearson', calculate the Pearson Correlations. If methd='kendall', calculate the Kendall tau Correlations. If method='similarity', calculate the Cosine Similarities. If method='distance', calculate the Euclidean Distances.

rescale : bool True or False.

Rescale the values in RDM or not.

Here, the maximum-minimum method is used to rescale the values except for the values on the diagonal.

permutation : bool True or False. Default is False.

Use permutation test or not.

iter : int. Default is 1000.

The times for iteration.

Returns

corrs : array

The similarities between EEG/MEG/fNIRS data and fMRI data for searchlight.

If chl_opt=1 & sub_result=1, the shape of corrs is [n_subs, n_chls, n_x, n_y, n_z, 2]. n_subs, n_x, n_y, n_z represent the number of subjects, the number of calculation units for searchlight along the x, y, z axis and 2 represents a r-value and a p-value. If chl_opt=1 & sub_result=0, the shape of corrs is [n_chls, n_x, n_y, n_z, 2]. n_x, n_y, n_z represent the number of calculation units for searchlight along the x, y, z axis and 2 represents a r-value and a p-value. If chl_opt=0 & sub_result=1, the shape of RDMs is [n_subs, n_x, n_y, n_z, 2]. n_subs, n_x, n_y, n_z represent the number of subjects, the number of calculation units for searchlight along the x, y, z axis and 2 represents a r-value and a p-value. If chl_opt=0 & sub_result=0, the shape of corrs is [n_x, n_y, n_z, 2]. n_x, n_y, n_z represent the number of calculation units for searchlight along the x, y, z axis and 2 represents a r-value and a p-value.

Module: *corr_cal_by_rdm.py*

- a module for calculating the Similarity/Correlation Cosfficient between RDMs by different modes

'a function for calculating the similarity between RDMs based on RDMs of EEG-like data and a demo RDM'

- **rdms_corr(demo_rdm, eeg_rdms, method="spearman", rescale=False, permutation=False, iter=1000)**

Parameters

demo_rdm : array [n_cons, n_cons]
A demo RDM.

eeg_rdms : array
The EEG/MEG/fNIRS/ECoG/sEEG/electrophysiological RDM(s).
The shape can be [n_cons, n_cons] or [n1, n_cons, n_cons] or [n1, n2, n_cons, n_cons] or [n1, n2, n3, n_cons, n_cons]. ni(i=1, 2, 3) can be int(n_ts/timw_win), n_chls, n_subs.

method : string 'spearman' or 'pearson' or 'kendall' or 'similarity' or 'distance'. Default is 'spearman'.
The method to calculate the similarities.
If method='spearman', calculate the Spearman Correlations. If method='pearson', calculate the Pearson Correlations. If methd='kendall', calculate the Kendall tau Correlations. If method='similarity', calculate the Cosine Similarities. If method='distance', calculate the Euclidean Distances.

rescale : bool True or False.
Rescale the values in RDM or not.
Here, the maximum-minimum method is used to rescale the values except for the values on the diagonal.

permutation : bool True or False. Default is False.
Use permutation test or not.

iter : int. Default is 1000.
The times for iteration.

Returns

corrs : array
The similarities between EEG/MEG/fNIRS/ECoG/sEEG/ electrophysiological RDMs and a demo RDM.
If the shape of eeg_rdms is [n_cons, n_cons], the shape of corrs will be [2]. If the shape of eeg_rdms is [n1, n_cons, n_cons], the shape of corrs will be [n1, 2]. If the shape of eeg_rdms is [n1, n2, n_cons, n_cons], the shape of corrs will be [n1, n2, 2]. If the shape of eeg_rdms is [n1, n2, n3, n_cons, n_cons], the shape of corrs will be [n1, n2, n3, 2]. ni(i=1, 2, 3) can be int(n_ts/timw_win), n_chls, n_subs. 2 represents a r-value and a p-value.

Notes

The demo RDM could be a behavioral RDM, a hypothesis-based coding model RDM or a computational model RDM.

'a function for calculating the similarity between fMRI searchlight RDMs and a demo RDM'

- **fmrirdms_corr(demo_rdm, fmri_rdms, method="spearman", rescale=False, permutation=False, iter=1000)**

Parameters

demo_rdm : array [n_cons, n_cons]
A demo RDM.

fmri_rdms : array
The fMRI-Searchlight RDMs.
The shape of RDMs is [n_x, n_y, n_z, n_cons, n_cons]. n_x, n_y, n_z represent the number of calculation units for searchlight along the x, y, z axis.

method : string 'spearman' or 'pearson' or 'kendall' or 'similarity' or 'distance'. Default is 'spearman'.
The method to calculate the similarities. If method='spearman', calculate the Spearman Correlations. If method='pearson', calculate the Pearson Correlations. If methd='kendall', calculate the Kendall tau Correlations. If method='similarity', calculate the Cosine Similarities. If method='distance', calculate the Euclidean Distances.

rescale : bool True or False.
Rescale the values in RDM or not.
Here, the maximum-minimum method is used to rescale the values except for the values on the diagonal.

permutation : bool True or False. Default is False.
Use permutation test or not.

iter : int. Default is 1000.
The times for iteration.

Returns

corrs : array
The similarities between fMRI searchlight RDMs and a demo RDM
The shape of RDMs is [n_x, n_y, n_z, 2]. n_x, n_y, n_z represent the number of calculation units for searchlight along the x, y, z axis and 2 represents a r-value and a p-value.

Notes

The demo RDM could be a behavioral RDM, a hypothesis-based coding model RDM or a computational model RDM.

Part 9: To Conduct Cross-Temporal Representational Similarity Analysis

Module: *ctrdm_corr.py*

- a module for calculating the Similarity/Correlation Coefficient between two Cross-Temporal RDMs

'a function for calculating the Spearman correlation coefficient between two CTRDMs'

- **ctrdm_correlation_spearman(CTRDM1, CTRDM2)**

Parameters

CTRDM1 : array [ncons, ncons]
The CTRDM 1.
The shape of CTRDM1 must be [n_cons, n_cons]. n_cons represent the number of conidtions.

CTRDM2 : array [ncons, ncons].
The CTRDM 2.
The shape of CTRDM2 must be [n_cons, n_cons]. n_cons represent the number of conidtions.

Returns

corr : array [r, p].
The Spearman Correlation result.
The shape of corr is [2], including a r-value and a p-value.

'a function for calculating the Pearson correlation coefficient between two CTRDMs'

- **ctrdm_correlation_pearson(CTRDM1, CTRDM2)**

Parameters

CTRDM1 : array [ncons, ncons]
The CTRDM 1.
The shape of CTRDM1 must be [n_cons, n_cons]. n_cons represent the number of conidtions.

CTRDM2 : array [ncons, ncons].
The CTRDM 2.
The shape of CTRDM2 must be [n_cons, n_cons]. n_cons represent the number of conidtions.

Returns

corr : array [r, p].
The Pearson Correlation result.
The shape of corr is [2], including a r-value and a p-value.

'a function for calculating the Kendalls tau correlation coefficient between two CTRDMs'

- **rdm_correlation_kendall(CTRDM1, CTRDM2)**

Parameters

CTRDM1 : array [ncons, ncons]
The CTRDM 1.
The shape of CTRDM1 must be [n_cons, n_cons]. n_cons represent the number of conidtions.

CTRDM2 : array [ncons, ncons].
The CTRDM 2.
The shape of CTRDM2 must be [n_cons, n_cons]. n_cons represent the number of conidtions.

Returns

corr : array [r, p].
The Kendall tau Correlation result.
The shape of corr is [2], including a r-value and a p-value.

'a function for calculating the Cosine Similarity between two CTRDMs'

- **rdm_similarity(CTRDM1, CTRDM2)**

Parameters

CTRDM1 : array [ncons, ncons]
The CTRDM 1.
The shape of CTRDM1 must be [n_cons, n_cons]. n_cons represent the number of conidtions.

CTRDM2 : array [ncons, ncons].
The CTRDM 2.
The shape of CTRDM2 must be [n_cons, n_cons]. n_cons represent the number of conidtions.

Returns

similarity : float.
The Cosine Similarity result.

'a fuction for calculating the Euclidean Distance between two CTRDMs'

- **rdm_distance(CTRDM1, CTRDM2)**

Parameters

CTRDM1 : array [ncons, ncons]

The CTRDM 1.

The shape of CTRDM1 must be [n_cons, n_cons]. n_cons represent the number of conidtions.

CTRDM2 : array [ncons, ncons].

The CTRDM 2.

The shape of CTRDM2 must be [n_cons, n_cons]. n_cons represent the number of conidtions.

Returns

dist : float

The Euclidean Distance result.

Module: *ctcorr_cal.py*

- a module for calculating the cross-temporal similarities

'a function for calculating cross-temporal similarities between neural data under two conditions'

➢ **ctsim_cal(data1, data2, sub_opt=1, chl_opt=0, time_win=5, time_step=5)**

Parameters

data1 : array

EEG/MEG/fNIRS/EEG-like data from a time-window under condition1.

The shape of data must be [n_subs, n_chls, n_ts]. n_subs, n_chls, n_ts represent the number of subjects, the number of channels and the number of time-points respectively.

data2 : array

EEG/MEG/fNIRS/EEG-like data from a time-window under condition2.

The shape of data must be [n_subs, n_chls, n_ts]. n_subs, n_chls, n_ts represent the number of subjects, the number of channels and the number of time-points respectively.

sub_opt : int 0 or 1. Default is 1.

Calculate the similarities for each subject or not.

If sub_opt=0, calculating based on all data. If sub_opt=1, calculating based on each subject's data, respectively.

chl_opt : int 0 or 1. Default is 0.

Calculate the similarities for each channel or not.

If chl_opt=0, calculating based on all channels' data. If chl_opt=1, calculating based on each channel's data respectively.

time_win : int. Default is 5.

Set a time-window for calculating the similarities for

different time-points.

If time_win=5, that means each calculation process based on 5 time-points.

time_step : int. Default is 5.

The time step size for each time of calculating.

Only when time_opt=1, time_step works.

Returns

CTSimilarities : array

Cross-temporal similarities.

If sub_opt=1 and chl_opt=1, the shape of CTSimilarities will be [n_subs, n_channels, int((n_ts-time_win)/time_step)+1, int((n_ts-time_win)/time_step)+1, 2]. If sub_opt=1 and chl_opt=0, the shape of CTSimilarities will be [n_subs, int((n_ts-time_win)/time_step)+1, int((n_ts-time_win)/time_step)+1, 2]. If sub_opt=0 and chl_opt=1, the shape of CTSimilarities will be [n_channels, int((n_ts-time_win)/time_step)+1, int((n_ts-time_win)/time_step)+1, 2]. If sub_opt=0 and chl_opt=0, the shape of CTSimilarities will be [int((n_ts-time_win)/time_step)+1, int((n_ts-time_win)/time_step)+1, 2].

'a function for calculating cross-temporal similarity matrix based on temporal RDMs'

- **ctsim_rdms_cal(RDMs, method="spearman")**

Parameters

RDMs : array

The Representational Dissimilarity Matrices in time series.

The shape could be [n_ts, n_cons, n_cons] or [n_subs, n_ts, n_cons, n_cons] or [n_chls, n_ts, n_cons, n_cons] or [n_subs, n_chls, n_ts, n_cons, n_cons]. n_ts, n_conditions, n_subs, n_chls represent the number of time-points, the number of conditions, the number of subjects and the number of channels, respectively.

method : string 'spearman' or 'pearson' or 'kendall' or 'similarity' or 'distance'. Default is 'spearman'.

The method to calculate the similarities.

If method='spearman', calculate the Spearman Correlations. If method='pearson', calculate the Pearson Correlations. If methd='kendall', calculate the Kendall tau Correlations. If method='similarity', calculate the Cosine Similarities. If method='distance', calculate the Euclidean Distances.

Returns

CTSimilarities : array

Cross-temporal similarities.

If the shape of RDMs is [n_ts, n_cons, n_cons] and method='spearman' or 'pearson' or 'kendall', the shape of CTSimilarities will be [n_ts, n_ts, 2]. If the shape of RDMs is [n_subs, n_ts, n_cons, n_cons] and

method='spearman' or 'pearson' or 'kendall', the shape of CTSimilarities will be [n_subs, n_ts, n_ts, 2]. If the shape of RDMs is [n_chls, n_ts, n_cons, n_cons] and method='spearman' or 'pearson' or 'kendall', the shape of CTSimilarities will be [n_chls, n_ts, n_ts, 2]. If the shape of RDMs is [n_subs, n_channels, n_ts, n_cons, n_cons] and method='spearman' or 'pearson' or 'kendall', the shape of CTSimilarities will be [n_subs, n_channels, n_ts, n_ts, 2]. If the shape of RDMs is [n_ts, n_cons, n_cons] and method='similarity' or 'distance', the shape of CTSimilarities will be [n_ts, n_ts]. If the shape of RDMs is [n_subs, n_ts, n_cons, n_cons] and method='similarity' or 'distance', the shape of CTSimilarities will be [n_subs, n_ts, n_ts]. If the shape of RDMs is [n_chls, n_ts, n_cons, n_cons] and method='similarity' or 'distance', the shape of CTSimilarities will be [n_chls, n_ts, n_ts]. If the shape of RDMs is [n_subs, n_channels, n_ts, n_cons, n_cons] and method='similarity' or 'distance', the shape of CTSimilarities will be [n_subs, n_chls, n_ts, n_ts].

'a function for calculating cross-temporal similarities between CTRDMs and a coding model RDM'

- **ctsim_ctrdms_cal(CTRDMs, Model_RDM, method="spearman")**

Parameters

CTRDMs : array

The cross-temporal representational dissimilarity matrices.

The shape could be [n_ts, n_ts, n_cons, n_cons] or [n_subs, n_ts, n_ts, n_cons, n_cons] or [n_chls, n_ts, n_ts, n_cons, n_cons] or [n_subs, n_chls, n_ts, n_ts, n_cons, n_cons]. n_ts, n_cons, n_subs, n_chls represent the number of time-points, the number of conditions, the number of subjects and the number of channels, respectively.

Model_RDM : array [n_cons, n_cons]

The coding model RDM.

method : string 'spearman' or 'pearson' or 'kendall' or 'similarity' or 'distance'. Default is 'spearman'.

The method to calculate the similarities.

If method='spearman', calculate the Spearman Correlations. If method='pearson', calculate the Pearson Correlations. If methd='kendall', calculate the Kendall tau Correlations. If method='similarity', calculate the Cosine Similarities. If method='distance', calculate the Euclidean Distances.

'a function for calculating cross-temporal similarities between temporal model RDMs and temporal neural RDMs (dynamic-RSA)'

- **ctsim_drsa_cal(Model_RDMs, RDMs, method="spearman")**

Parameters

Model_RDMs: array

The coding model RDMs.

The shape should be [n_ts, n_cons, n_cons]. n_ts and n_cons represent the number of time-points and the number conditions, respectively.

RDMs: array

The representational dissimilariy matrices in time series.

The shape could be [n_ts, n_cons, n_cons] or [n_subs, n_ts, n_cons, n_cons] or [n_channels, n_ts, n_cons, n_cons] or [n_subs, n_channels, n_ts, n_cons, n_cons]. n_ts, n_cons, n_subs, n_channels represent the number of time-points, the number of conditions, the number of subjects and the number of channels, respectively.

method : string 'spearman' or 'pearson' or 'kendall' or 'similarity' or 'distance'. Default is 'spearman'.

The method to calculate the similarities.

If method='spearman', calculate the Spearman Correlations. If method='pearson', calculate the Pearson Correlations. If methd='kendall', calculate the Kendall tau Correlations. If method='similarity', calculate the Cosine Similarities. If method='distance', calculate the Euclidean Distances.

Returns

CTSimilarities : array

Cross-temporal similarities.

If the shape of RDMs is [n_ts, n_cons, n_cons] and method='spearman' or 'pearson' or 'kendall', the shape of CTSimilarities will be [n_ts, n_ts, 2]. If the shape of RDMs is [n_subs, n_ts, n_cons, n_cons] and method='spearman' or 'pearson' or 'kendall', the shape of CTSimilarities will be [n_subs, n_ts, n_ts, 2]. If the shape of RDMs is [n_chls, n_ts, n_cons, n_cons] and method='spearman' or 'pearson' or 'kendall', the shape of CTSimilarities will be [n_chls, n_ts, n_ts, 2]. If the shape of RDMs is [n_subs, n_chls, n_ts, n_cons, n_cons] and method='spearman' or 'pearson' or 'kendall', the shape of CTSimilarities will be [n_subs, n_chls, n_ts, n_ts, 2]. If the shape of RDMs is [n_ts, n_cons, n_cons] and method='similarity' or 'distance', the shape of CTSimilarities will be [n_ts, n_ts]. If the shape of RDMs is [n_subs, n_ts, n_cons, n_cons] and method='similarity' or 'distance', the shape of CTSimilarities will be [n_subs, n_ts, n_ts]. If the shape of RDMs is [n_chls, n_ts, n_cons, n_cons] and method='similarity' or 'distance', the shape of CTSimilarities will be [n_chls, n_ts, n_ts]. If the shape of RDMs is [n_subs, n_chls, n_ts, n_cons, n_cons] and method='similarity' or 'distance', the shape of CTSimilarities will be [n_subs, n_chls, n_ts, n_ts].

Part 10: To Conduct Classification-based EEG Decoding

Module: *decoding.py*

- a module for classification-based neural decoding

'a function for time-by-time decoding for EEG-like data (cross validation)'

- **tbyt_decoding_kfold(data, labels, n=2, navg=5, time_opt="average", time_win=5, time_step=5, nfolds=5, nrepeats=2, normalization=False, smooth=True)**

Parameters

data : array
The neural data.
The shape of data must be [n_subs, n_trials, n_chls, n_ts]. n_subs, n_trials, n_chls and n_ts represent the number of subjects, the number of trails, the number of channels and the number of time-points.

labels : array
The labels of each trial.
The shape of labels must be [n_subs, n_trials]. n_subs and n_trials represent the number of subjects and the number of trials.

n : int. Default is 2.
The number of categories for classification.

navg : int. Default is 5.
The number of trials used to average.

time_opt : string "average" or "features". Default is "average".
Average the time-points or regard the time points as features for classification
If time_opt="average", the time-points in a certain time-window will be averaged. If time_opt="features", the time-points in a certain time-window will be used as features for classification.

time_win : int. Default is 5.
Set a time-window for decoding for different time-points.
If time_win=5, that means each decoding process based on 5 time-points.

time_step : int. Default is 5.
The time step size for each time of decoding.

nfolds : int. Default is 5.
The number of folds.
k should be at least 2.

nrepeats : int. Default is 2.

The times for iteration.

normalization : boolean True or False. Default is False.
The times for iteration.

smooth : boolean True or False, or int. Default is True.
Smooth the decoding result or not.
If smooth = True, the default smoothing step is 5. If smooth = n (type of n: int), the smoothing step is n.

Returns

accuracies : array
The time-by-time decoding accuracies.
The shape of accuracies is [n_subs, int((n_ts-time_win)/time_step)+1].

'a function for time-by-time decoding for EEG-like data (hold out)'

- **tbyt_decoding_holdout(data, labels, n=2, navg=5, time_opt="average", time_win=5, time_step=5, iter=10, test_size=0.3, normalization=False, smooth=True)**

Parameters

data : array
The neural data.
The shape of data must be [n_subs, n_trials, n_chls, n_ts]. n_subs, n_trials, n_chls and n_ts represent the number of subjects, the number of trails, the number of channels and the number of time-points.

labels : array
The labels of each trial.
The shape of labels must be [n_subs, n_trials]. n_subs and n_trials represent the number of subjects and the number of trials.

n : int. Default is 2.
The number of categories for classification.

navg : int. Default is 5.
The number of trials used to average.

time_opt : string "average" or "features". Default is "average".
Average the time-points or regard the time points as features for classification
If time_opt="average", the time-points in a certain time-window will be averaged. If time_opt="features", the time-points in a certain time-window will be used as features for classification.

time_win : int. Default is 5.
Set a time-window for decoding for different time-points.
If time_win=5, that means each decoding process based on 5 time-points.

time_step : int. Default is 5.

The time step size for each time of decoding.

iter : int. Default is 10.
The times for iteration.

test_size : float. Default is 0.3.
The proportion of the test set.
test_size should be between 0.0 and 1.0.

normalization : boolean True or False. Default is False.
The times for iteration.

smooth : boolean True or False, or int. Default is True.
Smooth the decoding result or not.
If smooth = True, the default smoothing step is 5. If smooth = n (type of n: int), the smoothing step is n.

Returns

accuracies : array
The time-by-time decoding accuracies.
The shape of accuracies is [n_subs, int((n_ts-time_win)/time_step)+1].

'a function for cross-temporal decoding for EEG-like data (cross validation)'

- **ct_decoding_kfold(data, labels, n=2, navg=5, time_opt="average", time_win=5, time_step=5, nfolds=5, nrepeats=2, normalization=False, smooth=True)**

Parameters

data : array
The neural data.
The shape of data must be [n_subs, n_trials, n_chls, n_ts]. n_subs, n_trials, n_chls and n_ts represent the number of subjects, the number of trails, the number of channels and the number of time-points.

labels : array
The labels of each trial.
The shape of labels must be [n_subs, n_trials]. n_subs and n_trials represent the number of subjects and the number of trials.

n : int. Default is 2.
The number of categories for classification.

navg : int. Default is 5.
The number of trials used to average.

time_opt : string "average" or "features". Default is "average".
Average the time-points or regard the time points as features for classification
If time_opt="average", the time-points in a certain time-window will be averaged. If time_opt="features", the time-points in a certain time-window will be used as features for classification.

time_win : int. Default is 5.
Set a time-window for decoding for different time-points.
If time_win=5, that means each decoding process based on 5 time-points.

time_step : int. Default is 5.
The time step size for each time of decoding.

nfolds : int. Default is 5.
The number of folds.
k should be at least 2.

nrepeats : int. Default is 2.
The times for iteration.

normalization : boolean True or False. Default is False.
The times for iteration.

smooth : boolean True or False, or int. Default is True.
Smooth the decoding result or not.
If smooth = True, the default smoothing step is 5. If smooth = n (type of n: int), the smoothing step is n.

Returns

accuracies : array
The cross-temporal decoding accuracies.
The shape of accuracies is [n_subs, int((n_ts-time_win)/time_step)+1, int((n_ts-time_win)/time_step)+1].

'a function for cross-temporal decoding for EEG-like data (hold out)'

- **tbyt_decoding_holdout(data, labels, n=2, navg=5, time_opt="average", time_win=5, time_step=5, iter=10, test_size=0.3, normalization=False, smooth=True)**

Parameters

data : array
The neural data.
The shape of data must be [n_subs, n_trials, n_chls, n_ts]. n_subs, n_trials, n_chls and n_ts represent the number of subjects, the number of trails, the number of channels and the number of time-points.

labels : array
The labels of each trial.
The shape of labels must be [n_subs, n_trials]. n_subs and n_trials represent the number of subjects and the number of trials.

n : int. Default is 2.
The number of categories for classification.

navg : int. Default is 5.
The number of trials used to average.

time_opt : string "average" or "features". Default is "average".

Average the time-points or regard the time points as features for classification

If time_opt="average", the time-points in a certain time-window will be averaged. If time_opt="features", the time-points in a certain time-window will be used as features for classification.

time_win : int. Default is 5.
Set a time-window for decoding for different time-points.
If time_win=5, that means each decoding process based on 5 time-points.

time_step : int. Default is 5.
The time step size for each time of decoding.

iter : int. Default is 10.
The times for iteration.

test_size : float. Default is 0.3.
The proportion of the test set.
test_size should be between 0.0 and 1.0.

normalization : boolean True or False. Default is False.
The times for iteration.

smooth : boolean True or False, or int. Default is True.
Smooth the decoding result or not.
If smooth = True, the default smoothing step is 5. If smooth = n (type of n: int), the smoothing step is n.

Returns

accuracies : array
The cross-temporal decoding accuracies.
The shape of accuracies is [n_subs, int((n_ts-time_win)/time_step)+1, int((n_ts-time_win)/time_step)+1].

'a function for unidirectional transfer decoding for EEG-like data'

- **unidirectional_transfer_decoding(data1, labels1, data2, labels2, n=2, navg=5, time_opt="average", time_win=5, time_step=5, iter=10, normalization=False, smooth=True)**

Parameters

data1 : array
The neural data under condition1.
The shape of data must be [n_subs, n_trials, n_chls, n_ts]. n_subs, n_trials, n_chls and n_ts represent the number of subjects, the number of trails, the number of channels and the number of time-points.

labels1 : array
The labels of each trial under condition1.
The shape of labels must be [n_subs, n_trials]. n_subs and n_trials represent the number of subjects and the number of trials.

data2 : array
The neural data under condition2.
labels2 : array
The labels of each trial under condition2.
n : int. Default is 2.
The number of categories for classification.
navg : int. Default is 5.
The number of trials used to average.
time_opt : string "average" or "features". Default is "average".
Average the time-points or regard the time points as features for classification
If time_opt="average", the time-points in a certain time-window will be averaged. If time_opt="features", the time-points in a certain time-window will be used as features for classification.
time_win : int. Default is 5.
Set a time-window for decoding for different time-points.
If time_win=5, that means each decoding process based on 5 time-points.
time_step : int. Default is 5.
The time step size for each time of decoding.
iter : int. Default is 10.
The times for iteration.
normalization : boolean True or False. Default is False.
The times for iteration.
smooth : boolean True or False, or int. Default is True.
Smooth the decoding result or not.
If smooth = True, the default smoothing step is 5. If smooth = n (type of n: int), the smoothing step is n.

Returns

accuracies : array
The unidirectional transfer decoding accuracies.
The shape of accuracies is [n_subs, int((n_ts1-time_win)/time_step)+1, int((n_ts2-time_win)/time_step)+1].

'a function for bidirectional transfer decoding for EEG-like data'

- **bidirectional_transfer_decoding(data1, labels1, data2, labels2, n=2, navg=5, time_opt="average", time_win=5, time_step=5, iter=10, normalization=False, smooth=True)**

Parameters

data1 : array
The neural data under condition1.
The shape of data must be [n_subs, n_trials, n_chls, n_ts].

n_subs, n_trials, n_chls and n_ts represent the number of subjects, the number of trails, the number of channels and the number of time-points.

labels1 : array
The labels of each trial under condition1.
The shape of labels must be [n_subs, n_trials]. n_subs and n_trials represent the number of subjects and the number of trials.

data2 : array
The neural data under condition2.

labels2 : array
The labels of each trial under condition2.

n : int. Default is 2.
The number of categories for classification.

navg : int. Default is 5.
The number of trials used to average.

time_opt : string "average" or "features". Default is "average".
Average the time-points or regard the time points as features for classification
If time_opt="average", the time-points in a certain time-window will be averaged. If time_opt="features", the time-points in a certain time-window will be used as features for classification.

time_win : int. Default is 5.
Set a time-window for decoding for different time-points.
If time_win=5, that means each decoding process based on 5 time-points.

time_step : int. Default is 5.
The time step size for each time of decoding.

iter : int. Default is 10.
The times for iteration.

normalization : boolean True or False. Default is False.
The times for iteration.

smooth : boolean True or False, or int. Default is True.
Smooth the decoding result or not.
If smooth = True, the default smoothing step is 5. If smooth = n (type of n: int), the smoothing step is n.

Returns

Con1toCon2_accuracies : array
The 1 transfer to 2 decoding accuracies.
The shape of accuracies is [n_subs, int((n_ts1-time_win)/time_step)+1, int((n_ts2-time_win)/time_step)+1].

Con2toCon1_accuracies : array
The 2 transfer to 1 decoding accuracies.
The shape of accuracies is [n_subs, int((n_ts2-time_win)/time_step)+1, int((n_ts1-time_win)/time_step)+1].

Part 11: To Conduct Statistical Analysis

Module: *stats_cal.py*

- a module for conducting the statistical analysis

'a function for conducting the statistical analysis for results of EEG-like data'

- **stats(corrs, fisherz=True, permutation=True, iter=1000)**

Parameters

corrs : array
The correlation coefficients.
The shape of corrs must be [n_subs, n_chls, n_ts, 2]. n_subs, n_chls, n_ts represent the number of subjects, the number of channels and the number of time-points. 2 represents a r-value and a p-value.

fisherz : bool True or False. Default is True.
Conduct Fisher-Z transform.

permutation : bool True or False. Default is False.
Use permutation test or not.

iter : int. Default is 1000.
The times for iteration.

Returns

stats : array
The statistical results.
The shape of stats is [n_chls, n_ts, 2]. n_chls, n_ts represent the number of channels and the number of time-points. 2 represents a t-value and a p-value.

'a function for conducting the statistical analysis for results of fMRI data (searchlight)'

- **stats(corrs, fisherz=True, permutation=True, iter=1000)**

Parameters

corrs : array
The correlation coefficients.
The shape of corrs must be [n_subs, n_x, n_y, n_z, 2]. n_subs, n_x, n_y, n_z represent the number of subjects, the number of calculation units for searchlight along the x, y, z axis and 2 represents a r-value and a p-value.

fisherz : bool True or False. Default is True.
Conduct Fisher-Z transform.

permutation : bool True or False. Default is False.

Use permutation test or not.

iter : int. Default is 1000.
The times for iteration.

Returns

stats : array
The statistical results.
The shape of stats is [n_x, n_y, n_z, 2]. n_x, n_y, n_z represent the number of calculation units for searchlight along the x, y, z axis and 2 represents a t-value and a p-value.

Notes

n_subs must >= 6.

This function can be used for the results of searchlight fMRI NPS and searchlight fMRI RDM-correlations.

'a function for conducting the statistical analysis for results of fMRI data (searchlight) within group'

- **stats_fmri_compare_withingroup(corrs1, corrs2, fisherz=True, permutation=False, iter=1000)**

Parameters

corrs1 : array
The correlation coefficients under condition 1.
The shape of corrs must be [n_subs, n_x, n_y, n_z, 2]. n_subs, n_x, n_y, n_z represent the number of subjects, the number of calculation units for searchlight along the x, y, z axis and 2 represents a r-value and a p-value.

corrs2 : array
The correlation coefficients under condition 2.
The shape of corrs must be [n_subs, n_x, n_y, n_z, 2]. n_subs, n_x, n_y, n_z represent the number of subjects, the number of calculation units for searchlight along the x, y, z axis and 2 represents a r-value and a p-value.

fisherz : bool True or False. Default is True.
Conduct Fisher-Z transform.

permutation : bool True or False. Default is False.
Use permutation test or not.

iter : int. Default is 1000.
The times for iteration.

Returns

stats : array
The statistical results.
The shape of stats is [n_x, n_y, n_z, 2]. n_x, n_y, n_z represent the number of calculation units for searchlight along the x, y, z axis and 2 represents a t-value and a p-value.

Notes

n_subs must >= 6.

This function can be used for the results of searchlight fMRI NPS and searchlight fMRI RDM-correlations.

'a function for conducting the statistical analysis for results of fMRI data (searchlight) between two groups'

- **stats_fmri_compare_betweengroups(corrs1, corrs2, fisherz=True, permutation=False, iter=1000)**

Parameters

corrs1 : array
The correlation coefficients for group 1.
The shape of corrs must be [n_subs, n_x, n_y, n_z, 2]. n_subs, n_x, n_y, n_z represent the number of subjects, the number of calculation units for searchlight along the x, y, z axis and 2 represents a r-value and a p-value.

corrs2 : array
The correlation coefficients for group 2.
The shape of corrs must be [n_subs, n_x, n_y, n_z, 2]. n_subs, n_x, n_y, n_z represent the number of subjects, the number of calculation units for searchlight along the x, y, z axis and 2 represents a r-value and a p-value.

fisherz : bool True or False. Default is True.
Conduct Fisher-Z transform.

permutation : bool True or False. Default is False.
Use permutation test or not.

iter : int. Default is 1000.
The times for iteration.

Returns

stats : array
The statistical results.
The shape of stats is [n_x, n_y, n_z, 2]. n_x, n_y, n_z represent the number of calculation units for searchlight along the x, y, z axis and 2 represents a t-value and a p-value.

Notes

n_subs must >= 6.

This function can be used for the results of searchlight fMRI NPS and searchlight fMRI RDM-correlations.

'a function for conducting the statistical analysis for results of fMRI data (ISC searchlight)'

- **stats_iscfmri(corrs, fisherz=True, permutation=False, iter=1000)**

Parameters

corrs : array

The correlation coefficients.

The shape of corrs must be [n_ts, n_subs!/(2!(n_subs-2)!), n_x, n_y, n_z, 2]. n_ts, n_subs, n_x, n_y, n_z represent the number of subjects, the number of calculation units for searchlight along the x, y, z axis and 2 represents a r-value and a p-value.*

fisherz : bool True or False. Default is True.
Conduct Fisher-Z transform.

permutation : bool True or False. Default is False.
Use permutation test or not.

iter : int. Default is 1000.
The times for iteration.

Returns

stats : array
The statistical results.

The shape of stats is [n_ts, n_x, n_y, n_z, 2]. n_ts, n_x, n_y, n_z represent the number of time-points, the number of calculation units for searchlight along the x, y, z axis and 2 represents a t-value and a p-value.

Notes

n_subs must >= 4 (n_subs!/(2!*(n_subs-2)!) >= 6).

'a function for conducting the statistical analysis for results of EEG-like data (for STPS)'

- **stats_stps(corrs1, corrs2, fisherz=True, permutation=False, iter=1000)**

Parameters

corrs1 : array
The correlation coefficients under condition1.

The shape of corrs1 must be [n_subs, n_chls, n_ts]. n_subs, n_chls, n_ts represent the number of subjects, the number of channels and the number of time-points.

corrs2 : array
The correlation coefficients under condition2.

The shape of corrs2 must be [n_subs, n_chls, n_ts]. n_subs, n_chls, n_ts represent the number of subjects, the number of channels and the number of time-points.

fisherz : bool True or False. Default is True.
Conduct Fisher-Z transform.

permutation : bool True or False. Default is False.
Use permutation test or not.

iter : int. Default is 1000.
The times for iteration.

Returns

stats : array
The statistical results.

The shape of stats is [n_chls, n_ts, 2]. n_chls, n_ts represent the number of channels and the number of time-points. 2 represents a t-value and a p-value.

Notes

n_subs must >= 6.

'a function for conducting the statistical analysis for results of fMRI data (STPS searchlight)'

- **stats_stpsfmri(corrs1, corrs2, fisherz=True, permutation=False, iter=1000)**

Parameters

corrs1 : array
The correlation coefficients under condition1.
The shape of corrs1 must be [n_subs, n_x, n_y, n_z]. n_subs, n_x, n_y, n_z represent the number of subjects, the number of calculation units for searchlight along the x, y, z axis.

corrs2 : array
The correlation coefficients under condition2.
The shape of corrs1 must be [n_subs, n_x, n_y, n_z]. n_subs, n_x, n_y, n_z represent the number of subjects, the number of calculation units for searchlight along the x, y, z axis.

fisherz : bool True or False. Default is True.
Conduct Fisher-Z transform.

permutation : bool True or False. Default is False.
Use permutation test or not.

iter : int. Default is 1000.
The times for iteration.

Returns

stats : array
The statistical results.
The shape of stats is [n_x, n_y, n_z, 2]. n_x, n_y, n_z represent the number of calculation units for searchlight along the x, y, z axis and 2 represents a t-value and a p-value.

Notes

n_subs must >= 6.

Part 12: To Save Results as a NIfTI file (for fMRI)

Module: *nii_save.py*

- a module for saving the RSA results in a .nii file for fMRI

'a function for saving the searchlight correlation coefficients as a NIfTI file for fMRI'

- **corr_save_nii(corrs, affine, filename=None, corr_mask=get_HOcort(), size=[60, 60, 60], ksize=[3, 3, 3], strides=[1, 1, 1], p=1, r=0, correct_method=None, clusterp=0.05, smooth=True, plotrlt=True, img_background=None)**

Parameters

corrs : array
The similarities between behavioral data and fMRI data for searchlight.
The shape of RDMs is [n_x, n_y, n_z, 2]. n_x, n_y, n_z represent the number of calculation units for searchlight along the x, y, z axis and 2 represents a r-value and a p-value.

affine : array or list
The position information of the fMRI-image array data in a reference space.

filename : string. Default is None - 'rsa_result.nii'.
The file path+filename for the result .nii file.
If the filename does not end in ".nii", it will be filled in automatically.

corr_mask : string. Default is get_HOcort().
The filename of a mask data for correcting the RSA result.
It can just be one of your fMRI data files in your experiment for a mask file for ROI. If the corr_mask is a filename of a ROI mask file, only the RSA results in ROI will be visible.

size : array or list [nx, ny, nz]. Default is [60, 60, 60].
The size of the fMRI-img in your experiments.

ksize : array or list [kx, ky, kz]. Default is [3, 3, 3].
The size of the calculation unit for searchlight.
kx, ky, kz represent the number of voxels along the x, y, z axis.

strides : array or list [sx, sy, sz]. Default is [1, 1, 1].
The strides for calculating along the x, y, z axis.

p : float. Default is 1.

The threshold of p-values.

Only the results those p-values are lower than this value will be visible.

r : float. Default is 0.

The threshold of r-values.

Only the results those r-values are higher than this value will be visible.

correct_method : None or string 'FWE' or 'FDR' or 'Cluster-FWE' or 'Cluster-FDR'. Default is None.

The method for correcting the RSA results.

If correct_method='FWE', here the FWE-correction will be used. If correct_methd='FDR', here the FDR-correction will be used. If correct_method='Cluster-FWE', here the Cluster-wise FWE-correction will be used. If correct_methd='Cluster-FDR', here the Cluster-wise FDR-correction will be used. If correct_method=None, no correction. If correct_method=None, no correction. Only when p<1, correct_method works.

clusterp : float. Default is 0.05.

The threshold of p-value for cluster-wise correction.

Only when correct_method='Cluster-FDR' or 'Cluster-FWE', clusterp works.

smooth : bool True or False. Default is True.

Smooth the RSA result or not.

plotrlt : bool True or False.

Plot the RSA result automatically or not.

img_background : None or string. Default if None.

The filename of a background image that the RSA results will be plotted on the top of it.

If img_background=None, the background will be ch2.nii.gz. Only when plotrlt=True, img_background works.

Returns

img : array

The array of the correlation coefficients map.

The shape is [nx, ny, nz]. nx, ny, nz represent the size of the fMRI-img.

Notes

A result .nii file of searchlight correlation coefficients will be generated at the corresponding address of filename.

'a function for saving the searchlight statistical results as a NIfTI file for fMRI'

- **stats_save_nii(corrs, affine, filename=None, corr_mask=get_HOcort(), size=[60, 60, 60], ksize=[3, 3, 3], strides=[1, 1, 1], p=0.05,**

correct_method=None, clusterp=0.05, smooth=False, plotrlt=True, img_background=None)

Parameters

stats : array

The statistical results between behavioral data and fMRI data for searchlight.

The shape of RDMs is [n_x, n_y, n_z, 2]. n_x, n_y, n_z represent the number of calculation units for searchlight along the x, y, z axis and 2 represents a t-value and a p-value. If the filename does not end in ".nii", it will be filled in automatically.

affine : array or list

The position information of the fMRI-image array data in a reference space.

filename : string. Default is None - 'rsa_result.nii'.

The file path+filename for the result .nii file.

corr_mask : string

The filename of a mask data for correcting the RSA result.

It can just be one of your fMRI data files in your experiment for a mask file for ROI. If the corr_mask is a filename of a ROI mask file, only the RSA results in ROI will be visible.

size : array or list [nx, ny, nz]. Default is [60, 60, 60].

The size of the fMRI-img in your experiments.

ksize : array or list [kx, ky, kz]. Default is [3, 3, 3].

The size of the calculation unit for searchlight.

kx, ky, kz represent the number of voxels along the x, y, z axis.

strides : array or list [sx, sy, sz]. Default is [1, 1, 1].

The strides for calculating along the x, y, z axis.

p : float. Default is 0.05.

The threshold of p-values.

Only the results those p-values are lower than this value will be visible.

correct_method : None or string 'FWE' or 'FDR' or 'Cluster-FWE' or 'Cluster-FDR'. Default is None.

The method for correcting the RSA results.

If correct_method='FWE', here the FWE-correction will be used. If correct_methd='FDR', here the FDR-correction will be used. If correct_method='Cluster-FWE', here the Cluster-wise FWE-correction will be used. If correct_methd='Cluster-FDR', here the Cluster-wise FDR-correction will be used. If correct_method=None, no correction. Only when p<1, correct_method works.

clusterp : float. Default is 0.05.

The threshold of p-value for cluster-wise correction.

Only when correct_method='Cluster-FDR' or 'Cluster-FWE', clusterp works.

smooth : bool True or False. Default is False.
Smooth the RSA result or not.
plotrlt : bool True or False. Default is True.
Plot the RSA result automatically or not.
img_background : None or string. Default if None.
The filename of a background image that the RSA results will be plotted on the top of it. If img_background=None, the background will be ch2.nii.gz. Only when plotrlt=True, img_background works.

Returns

img : array
The array of the statistical results t-values map.
The shape is [nx, ny, nz]. nx, ny, nz represent the size of the fMRI-img.

Notes

A result .nii file of searchlight correlation coefficients will be generated at the corresponding address of filename.

Part 13: To Plot the Results

Module: *rsa_plot.py*

- a module for plotting the results

'a function for plotting the RDM'

- **plot_rdm(rdm, percentile=False, rescale=False, lim=[0, 1], conditions=None, con_fontsize=12, cmap=None)**

Parameters

rdm : array or list [n_cons, n_cons]
A representational dissimilarity matrix.

percentile : bool True or False. Default is False.
Rescale the values in RDM or not by displaying the percentile.

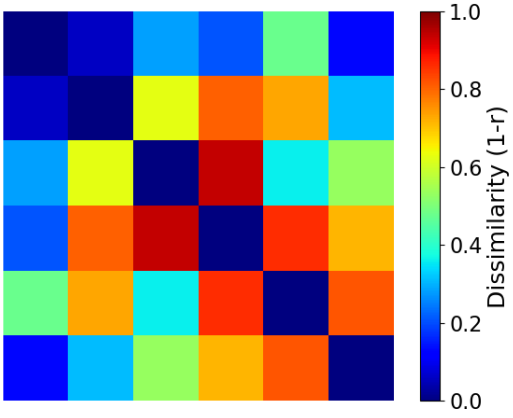
rescale : bool True or False. Default is False.
Rescale the values in RDM or not.
Here, the maximum-minimum method is used to rescale the values except for the values on the diagnal.

lim : array or list [min, max]. Default is [0, 1].
The corrs view lims.

conditions : string-array or string-list. Default is None.
The labels of the conditions for plotting.
conditions should contain n_cons strings, If conditions=None, the labels of conditions will be invisible.

con_fontsize : int or float. Default is 12.
The fontsize of the labels of the conditions for plotting.

cmap : matplotlib colormap. Default is None.
The colormap for RDM.
If cmap=None, the ccolormap will be 'jet'.



'a function for plotting the RDM with values'

- **plot_rdm(rdm, percentile=False, rescale=False, lim=[0, 1], conditions=None, con_fontsize=12, cmap=None)**

Parameters

rdm : array or list [n_cons, n_cons]
A representational dissimilarity matrix.

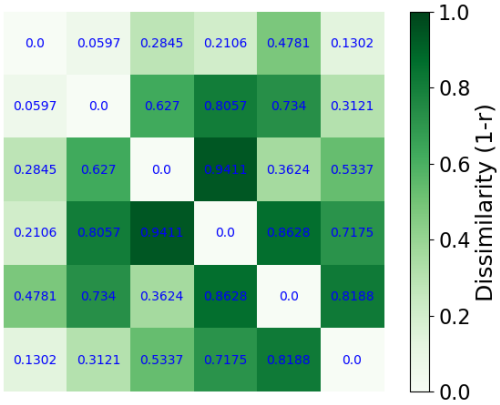
lim : array or list [min, max]. Default is [0, 1].
The corrs view lims.

value_fontsize : int or float. Default is 10.
The fontsize of the values on the RDM.

conditions : string-array or string-list or None. Default is None.
The labels of the conditions for plotting.
conditions should contain n_cons strings, If conditions=None, the labels of conditions will be invisible.

con_fontsize : int or float. Default is 12.
The fontsize of the labels of the conditions for plotting.

cmap : matplotlib colormap or None. Default is None.
The colormap for RDM.
If cmap=None, the ccolormap will be 'Greens'.



'a function for plotting the correlation coefficients by time sequence'

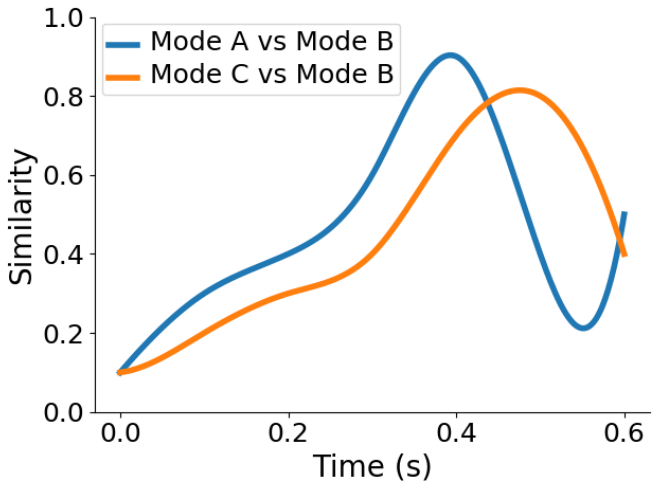
- **plot_corrs_by_time(corrs, labels=None, time_unit=[0, 0.1])**

Parameters

corrs : array
The correlation coefficients time-by-time.
The shape of corrs must be [n, ts, 2] or [n, ts]. n represents the number of curves of the correlation coefficient by time sequence. ts represents the time-points. If shape of corrs is [n, ts 2], each time-point of each correlation coefficient curve contains a r-value and a p-value. If shape is [n, ts], only r-values.

label : string-array or string-list or None. Default is None.
The label for each corrs curve.
If label=None, no legend in the figure.

time_unit : array or list [start_t, t_step]. Default is [0, 0.1]
The time information of corrs for plotting start_t represents the start time and t_step represents the time between two adjacent time-points. Default time_unit=[0, 0.1], which means the start time of corrs is 0 sec and the time step is 0.1 sec.



'a function for plotting the time-by-time Similarities with statistical results'

- **plot_tbytsim_withstats(similarities, start_time=0, end_time=1, time_interval=0.01, smooth=True, p=0.05, cbpt=True, stats_time=[0, 1], color='r', xlim=[0, 1], ylim=[-0.1, 0.8], xlabel='Time (s)', ylabel='Representational Similarity', figsize=[6.4, 3.6], x0=0, ticksize=12, fontsize=16, markersize=2, avgshow=False)**

Parameters

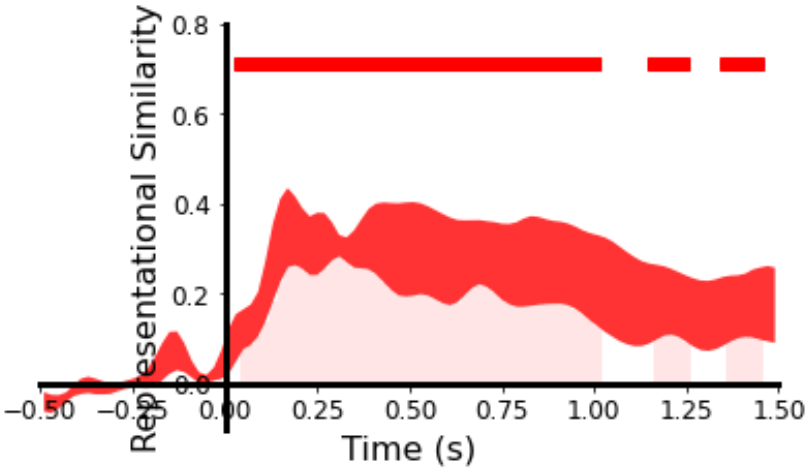
similarities : array
The Similarities.
The size of similarities should be [n_subs, n_ts] or [n_subs, n_ts, 2]. n_subs, n_ts represent the number of subjects and number of time-points. 2 represents the similarity and a p-value.

start_time : int or float. Default is 0.
The start time.

end_time : int or float. Default is 1.
The end time.

time_interval : float. Default is 0.01.
The time interval between two time samples.

smooth :	bool True or False. Default is True. Smooth the results or not.
chance :	float. Default is 0.5. The chance level.
p :	float. Default is 0.05. The threshold of p-values.
cbpt :	bool True or False. Default is True. Conduct cluster-based permutation test or not.
stats_time :	array or list [stats_time1, stats_time2]. Default os [0, 1]. Time period for statistical analysis.
color :	matplotlib color or None. Default is 'r'. The color for the curve.
xlim :	array or list [xmin, xmax]. Default is [0, 1]. The x-axis (time) view lims.
ylim :	array or list [ymin, ymax]. Default is [0.4, 0.8]. The y-axis (decoding accuracy) view lims.
xlabel:	string. Default is 'Time (s)'. The label of x-axis.
ylabel:	string. Default is 'Representational Similarity. The label of y-axis.
figsize :	array or list, [size_X, size_Y]. Default is [6.4, 3.6]. The size of the figure.
xo :	float. Default is 0. The Y-axis is at x=xo.
ticksize :	int or float. Default is 12. The size of the ticks.
fontsize :	int or float. Default is 16. The fontsize of the labels.
markersize :	int or float. Default is 2. The size of significant markers.
avgshow :	boolen True or False. Default is False. Show the averaging decoding accuracies or not.



'a function for plotting the plotting the time-by-time decoding accuracies'

- **plot_tbyt_decoding_acc(acc, start_time=0, end_time=1, time_interval=0.01, chance=0.5, p=0.05, cbpt=True, stats_time=[0, 1], color='r', xlim=[0, 1], ylim=[0.4, 0.8], xlabel='Time (s)', ylabel='Decoding Accuracy', figsize=[6.4, 3.6], xo=0, ticksize=12, fontsize=16, markersize=2, avgshow=False)**

Parameters

acc : array
The decoding accuracies.
The size of acc should be [n_subs, n_ts]. n_subs, n_ts represent the number of subjects and number of time-points.

start_time : int or float. Default is 0.
The start time.

end_time : int or float. Default is 1.
The end time.

time_interval : float. Default is 0.01.
The time interval between two time samples.

chance : float. Default is 0.5.
The chance level.

p : float. Default is 0.05.
The threshold of p-values.

cbpt : bool True or False. Default is True.
Conduct cluster-based permutation test or not.

stats_time : array or list [stats_time1, stats_time2]. Default os [0, 1].
Time period for statistical analysis.

color : matplotlib color or None. Default is 'r'.
The color for the curve.

xlim : array or list [xmin, xmax]. Default is [0, 1].
The x-axis (time) view lims.

ylim : array or list [ymin, ymax]. Default is [0.4, 0.8].
The y-axis (decoding accuracy) view lims.

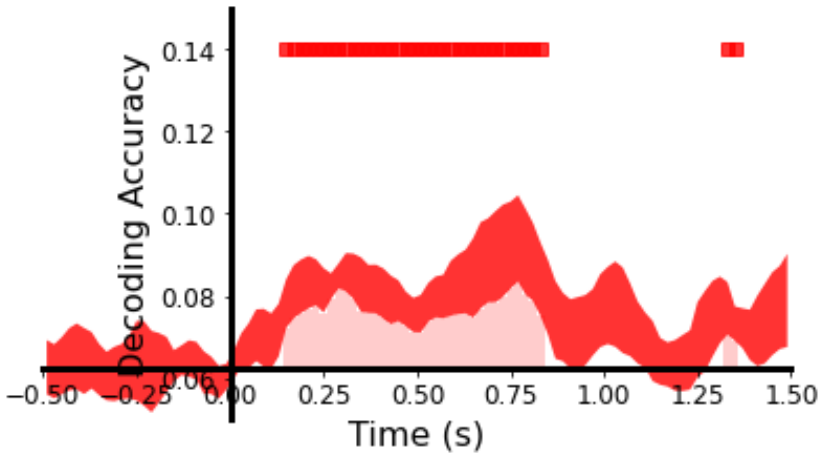
xlabel: string. Default is 'Time (s)'.
The label of x-axis.

ylabel: string. Default is 'Decoding Accuracy.
The label of y-axis.

figsize : array or list, [size_X, size_Y]. Default is [6.4, 3.6].
The size of the figure.

xo : float. Default is 0.
The Y-axis is at x=xo.

ticksize : int or float. Default is 12.
The size of the ticks.
fontsize : int or float. Default is 16.
The fontsize of the labels.
markersize : int or float. Default is 2.
The size of significant markers.
avgshow : boolen True or False. Default is False.
Show the averaging decoding accuracies or not.



'a function for plotting the differences of time-by-time decoding accuracies between two conditions'

- **plot_tbyt_diff_decoding_acc(acc1, acc2, start_time=0, end_time=1, time_interval=0.01, chance=0.5, p=0.05, cbpt=True, stats_time=[0, 1], color1='r', color2='b', xlim=[0, 1], ylim=[0.4, 0.8], xlabel='Time (s)', ylabel='Decoding Accuracy', figsize=[6.4, 3.6], x0=0, ticksize=12, fontsize=16, markersize=2, avgshow=False)**

Parameters

acc1 : array
The decoding accuracies under condition1.
The size of acc1 should be [n_subs, n_ts]. n_subs, n_ts represent the number of subjects and number of time-points.
acc2 : array
The decoding accuracies under condition2.
The size of acc2 should be [n_subs, n_ts]. n_subs, n_ts represent the number of subjects and number of time-points.
start_time : int or float. Default is 0.
The start time.
end_time : int or float. Default is 1.

	The end time.
time_interval :	float. Default is 0.01.
	The time interval between two time samples.
chance :	float. Default is 0.5.
	The chance level.
p :	float. Default is 0.05.
	The threshold of p-values.
cbpt :	bool True or False. Default is True.
	Conduct cluster-based permutation test or not.
stats_time :	array or list [stats_time1, stats_time2]. Default os [0, 1].
	Time period for statistical analysis.
color1 :	matplotlib color or None. Default is 'r'.
	The color for the curve under condition1.
color2 :	matplotlib color or None. Default is 'r'.
	The color for the curve under condition2.
xlim :	array or list [xmin, xmax]. Default is [0, 1].
	The x-axis (time) view lims.
ylim :	array or list [ymin, ymax]. Default is [0.4, 0.8].
	The y-axis (decoding accuracy) view lims.
xlabel:	string. Default is 'Time (s)'.
	The label of x-axis.
ylabel:	string. Default is 'Decoding Accuracy.
	The label of y-axis.
figsize :	array or list, [size_X, size_Y]. Default is [6.4, 3.6].
	The size of the figure.
x0 :	float. Default is 0.
ticksize :	int or float. Default is 12.
	The size of the ticks.
	The Y-axis is at x=x0.
fontsize :	int or float. Default is 16.
	The fontsize of the labels.
markersize :	int or float. Default is 2.
	The size of significant markers.
avgshow :	boolen True or False. Default is False.
	Show the averaging decoding accuracies or not.

'a function for plotting the plotting the cross-temporal decoding accuracies'

- **plot_ct_decoding_acc(acc, start_timex=0, end_timex=1, start_timey=0, end_timey=1, time_intervalx=0.01, time_intervaly=0.01, chance=0.5, p=0.05, cbpt=True, stats_timex=[0, 1], stats_timey=[0, 1], xlim=[0, 1], ylim=[0, 1], clim=[0.4, 0.8], xlabel='Training Time (s)', ylabel='Test Time**

(s)', clabel='Decoding Accuracy', figsize=[6.4, 4.8], cmap="viridis", ticksize=12, fontsize=16)

Parameters

acc : array
The decoding accuracies.
The size of acc should be [n_subs, n_ts]. n_subs, n_ts represent the number of subjects and number of time-points.

start_timex : int or float. Default is 0.
The training start time.

end_timex : int or float. Default is 1.
The training end time.

start_timey : int or float. Default is 0.
The test start time.

end_timey : int or float. Default is 1.
The test end time.

time_intervalx : float. Default is 0.01.
The training time interval between two time samples.

time_intervaly : float. Default is 0.01.
The test time interval between two time samples.

chance : float. Default is 0.5.
The chance level.

p : float. Default is 0.05.
The threshold of p-values.

cbpt : bool True or False. Default is True.
Conduct cluster-based permutation test or not.

stats_timex : array or list [stats_timex1, stats_timex2]. Default os [0, 1].
Trainning time period for statistical analysis.

stats_timey : array or list [stats_timey1, stats_timey2]. Default os [0, 1].
Test time period for statistical analysis.

xlim : array or list [xmin, xmax]. Default is [0, 1].
The x-axis (training time) view lims.

ylim : array or list [ymin, ymax]. Default is [0, 1].
The y-axis (test time) view lims.

clim : array or list [cmin, cmax]. Default is [0.4, 0.8].
The color-bar (decoding accuracy) view lims.

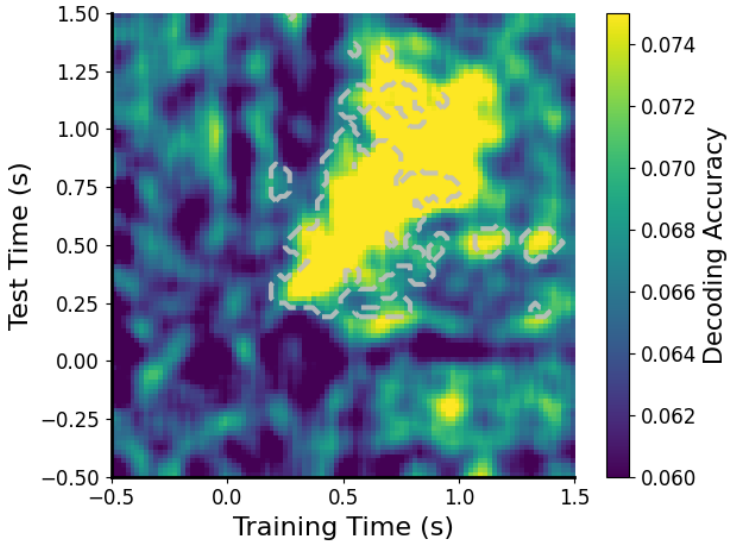
xlabel: string. Default is 'Training Time (s)'.
The label of x-axis.

ylabel: string. Default is 'Test Time (s)'.
The label of y-axis.

clabel: string. Default is 'Decoding Accuracy'.
The label of color-bar.

figsize : array or list, [size_X, size_Y]. Default is [6.4, 3.6].
The size of the figure.

cmap : matplotlib colormap or None. Default is None.
The colormap for the figure.
ticksize : int or float. Default is 12.
The size of the ticks.
fontsize : int or float. Default is 16.
The fontsize of the labels.



'a function for plotting the plotting the cross-temporal decoding accuracies between two conditions'

- **plot_ct_diff_decoding_acc(acc1, acc2, start_timex=0, end_timex=1, start_timey=0, end_timey=1, time_intervalx=0.01, time_intervaly=0.01, chance=0.5, p=0.05, cbpt=True, stats_timex=[0, 1], stats_timey=[0, 1], xlim=[0, 1], ylim=[0, 1], clim=[0.4, 0.8], xlabel='Training Time (s)', ylabel='Test Time (s)', clabel='Differences of Decoding Accuracies', figsize=[6.4, 4.8], cmap="viridis", ticksize=12, fontsize=16)**

Parameters

acc1 : array
The decoding accuracies under condition1.
The size of acc1 should be [n_subs, n_ts]. n_subs, n_ts represent the number of subjects and number of time-points.
acc2 : array
The decoding accuracies under condition2.
The size of acc2 should be [n_subs, n_ts]. n_subs, n_ts represent the number of subjects and number of time-points.
start_timex : int or float. Default is 0.
The training start time.

end_timex : int or float. Default is 1.
The training end time.
start_timey : int or float. Default is 0.
The test start time.
end_timey : int or float. Default is 1.
The test end time.
time_intervalx : float. Default is 0.01.
The training time interval between two time samples.
time_intervaly : float. Default is 0.01.
The test time interval between two time samples.
chance : float. Default is 0.5.
The chance level.
p : float. Default is 0.05.
The threshold of p-values.
cbpt : bool True or False. Default is True.
Conduct cluster-based permutation test or not.
stats_timex : array or list [stats_timex1, stats_timex2]. Default os [0, 1].
Trainning time period for statistical analysis.
stats_timey : array or list [stats_timey1, stats_timey2]. Default os [0, 1].
Test time period for statistical analysis.
xlim : array or list [xmin, xmax]. Default is [0, 1].
The x-axis (training time) view lims.
ylim : array or list [ymin, ymax]. Default is [0, 1].
The y-axis (test time) view lims.
clim : array or list [cmin, cmax]. Default is [0.4, 0.8].
The color-bar (decoding accuracy) view lims.
xlabel: string. Default is 'Training Time (s)'.
The label of x-axis.
ylabel: string. Default is 'Test Time (s)'.
The label of y-axis.
clabel: string. Default is 'Differences of Decoding Accuracies'.
The label of color-bar.
figsize : array or list, [size_X, size_Y]. Default is [6.4, 3.6].
The size of the figure.
cmap : matplotlib colormap or None. Default is None.
The colormap for the figure.
ticksize : int or float. Default is 12.
The size of the ticks.
fontsize : int or float. Default is 16.
The fontsize of the labels.

'a function for plotting the hotmap of correlations coefficients for channels/regions by time sequence'

➢ **plot_corrs_hotmap(corrs, chllabels=None, time_unit=[0, 0.1], lim=[0, 1], smooth=False, figsize=None, cmap=None)**

Parameters

corrs : array
The correlation coefficients time-by-time.
The shape of corrs must be [n_chls, ts, 2] or [n_chls, ts]. n_chls represents the number of channels or regions. ts represents the number of time-points. If shape of corrs is [n_chls, ts 2], each time-point of each channel/region contains a r-value and a p-value. If shape is [n_chls, ts], only r-values.

chllabel : string-array or string-list or None. Default is None.
The label for channels/regions.
If label=None, the labels will be '1st', '2nd', '3th', '4th', ... automatically.

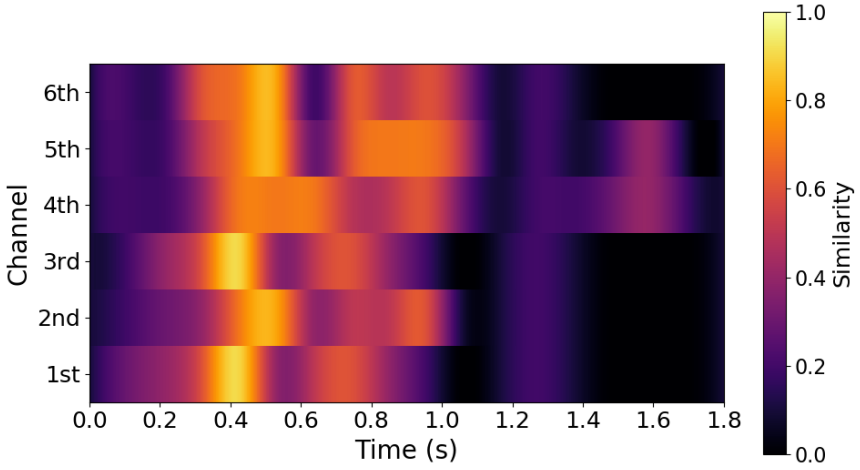
time_unit : array or list [start_t, t_step]. Default is [0, 0.1]
The time information of corrs for plotting
start_t represents the start time and t_step represents the time between two adjacent time-points. Default time_unit=[0, 0.1], which means the start time of corrs is 0 sec and the time step is 0.1 sec.

lim : array or list [min, max]. Default is [0, 1].
The corrs view lims.

smooth : bool True or False. Default is False.
Smooth the results or not.

figsize : array or list, [size_X, size_Y]
The size of the figure.
If figsize=None, the size of the figure will be ajusted automatically.

cmap : matplotlib colormap or None. Default is None.
The colormap for the figure.
If cmap=None, the ccolormap will be 'inferno'.



'a function for plotting the hotmap of correlations coefficients for channels/regions by time sequence with the significant outline'

- **plot_corrs_hotmap_withstats(corrs, chllabels=None, time_unit=[0, 0.1], lim=[0, 1], p =0.05, cbpt=False, clusterp=0.05, stats_time=[0, 1], smooth=False, xlabel='Time (s)', ylabel='Channel', clabel='Similarity', ticksize=18, figsize=None, cmap=None)**

Parameters

corrs : array
The correlation coefficients time-by-time.
The shape of corrs must be [n_subs, n_chls, ts, 2] or [n_subs, n_chls, ts]. n_subs represents the number of channels or regions. ts represents the number of time-points. If shape of corrs is [n_chls, ts 2], each time-point of each channel/region contains a r-value and a p-value. If shape is [n_chls, ts], only r-values.

chllabels : string-array or string-list or None. Default is None.
The label for channels/regions.
If label=None, the labels will be '1st', '2nd', '3th', '4th', ... automatically.

time_unit : array or list [start_t, t_step]. Default is [0, 0.1]
The time information of corrs for plotting
start_t represents the start time and t_step represents the time between two adjacent time-points. Default time_unit=[0, 0.1], which means the start time of corrs is 0 sec and the time step is 0.1 sec.

lim : array or list [min, max]. Default is [0, 1].
The corrs view lims.

p: float. Default is 0.05.
The p threshold for outline.

cbpt : bool True or False. Default is True.
Conduct cluster-based permutation test or not.

clusterp : float. Default is 0.05.
The threshold of cluster-defining p-values.

stats_time : array or list [stats_time1, stats_time2]. Default os [0, 1].
The time period for statistical analysis.

smooth : bool True or False. Default is False.
Smooth the results or not.

xlabel: string. Default is 'Time (s)'.
The label of x-axis.

ylabel: string. Default is 'Channel'.
The label of y-axis.

clabel: string. Default is 'Similarity'.

The label of color-bar.

ticksize : int or float. Default is 18.
The size of the ticks.

figsize : array or list, [size_X, size_Y]
The size of the figure.
If figsize=None, the size of the figure will be ajusted automatically.

cmap : matplotlib colormap or None. Default is None.
The colormap for the figure.
If cmap=None, the ccolormap will be 'inferno'.

'a function for plotting the hotmap of neural pattern similarities for channels/regions by time sequence'

- **plot_nps_hotmap(similarities, chllabels=None, time_unit=[0, 0.1], lim=[0, 1], abs=False, smooth=False, figsize=None, cmap=None)**

Parameters

similarities : array
The neural pattern similarities time-by-time.
The shape of similarities must be [n_chls, ts]. n_chls represents the number of channels or regions. ts represents the number of time-points.

chllabels : string-array or string-list or None. Default is None.
The label for channels/regions.
If label=None, the labels will be '1st', '2nd', '3th', '4th', ... automatically.

time_unit : array or list [start_t, t_step]. Default is [0, 0.1]
The time information of corrs for plotting
start_t represents the start time and t_step represents the time between two adjacent time-points. Default time_unit=[0, 0.1], which means the start time of corrs is 0 sec and the time step is 0.1 sec.

lim : array or list [min, max]. Default is [0, 1].
The corrs view lims.

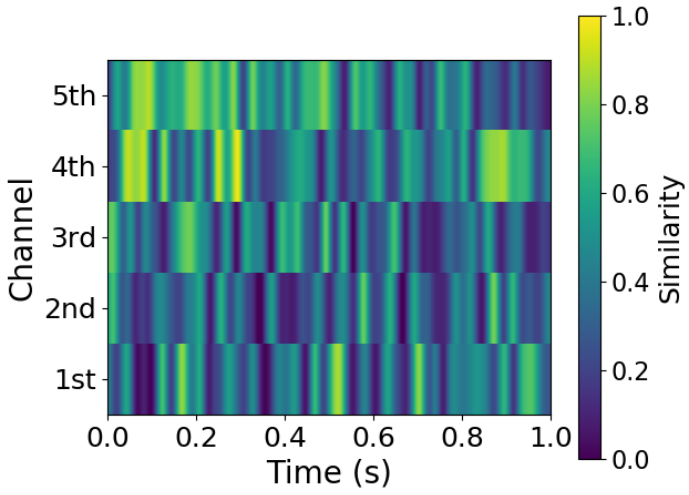
abs : boolean True or False. Default is False.
Change the similarities into absolute values or not.

smooth : boolean True or False. Default is False.
Smooth the results or not.

figsize : array or list, [size_X, size_Y]
The size of the figure.
If figsize=None, the size of the figure will be ajusted automatically.

cmap : matplotlib colormap or None. Default is None.
The colormap for the figure.

If cmap=None, the ccolormap will be 'viridis'.



'a function for plotting the hotmap of statistical results for channels/regions by time sequence'

- **plot_t_hotmap_withstats(results, chllabels=None, time_unit=[0, 0.1], lim=[-7, 7], p=0.05, cbpt=False, clusterp=0.05, stats_time=[0, 1], smooth=False, xlabel='Time (s)', ylabel='Channel', clabel='t', ticksize=18, figsize=None, cmap=None)**

Parameters

results : array
The results.
The shape of results must be [n_subs, n_chls, ts, 2] or [n_subs, n_chls, ts]. n_subs represents the number of subjects. n_chls represents the number of channels or regions. ts represents the number of time-points. If shape of results is [n_chls, ts, 2], each time-point of each channel/region and each subject contains a r-value and a p-value. If shape is [n_chls, ts], only r-values.

chllabels : string-array or string-list or None. Default is None.
The label for channels/regions.
If label=None, the labels will be '1st', '2nd', '3th', '4th', ... automatically.

time_unit : array or list [start_t, t_step]. Default is [0, 0.1]
The time information of corrs for plotting
start_t represents the start time and t_step represents the time between two adjacent time-points. Default time_unit=[0, 0.1], which means the start time of corrs is 0 sec and the time step is 0.1 sec.

lim : array or list [min, max]. Default is [-7, -7].
The corrs view lims.

p: float. Default is 0.05.

The p threshold for outline.

cbpt : bool True or False. Default is True.
Conduct cluster-based permutation test or not.

clusterp : float. Default is 0.05.
The threshold of cluster-defining p-values.

stats_time : array or list [stats_time1, stats_time2]. Default os [0, 1].
The time period for statistical analysis.

smooth : boolean True or False. Default is False.
Smooth the results or not.

xlabel: string. Default is 'Time (s)'.
The label of x-axis.

ylabel: string. Default is 'Channel'.
The label of y-axis.

clabel: string. Default is 't'.
The label of color-bar.

ticksize : int or float. Default is 18.
The size of the ticks.

figsize : array or list, [size_X, size_Y]
The size of the figure.
If figsize=None, the size of the figure will be ajusted automatically.

cmap : matplotlib colormap or None. Default is None.
The colormap for the figure.
If cmap=None, the ccolormap will be 'bwr'.

'a function for plotting the RSA-result regions by 3 cuts (frontal, axial & lateral)'

- **plot_brainrsa_regions(img, threshold=None,**

  background=get_bg_ch2(), type='r')

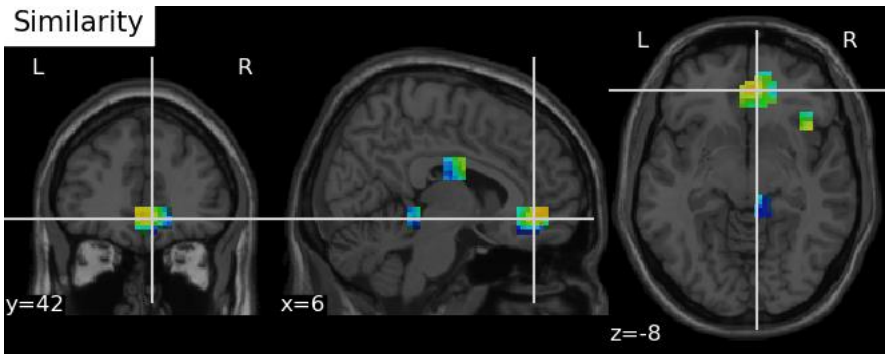
Parameters

img : string
The file path of the .nii file of the RSA results.

threshold : None or int. Default is None.
The threshold of the number of voxels used in correction.
If threshold=n, only the similarity clusters consisting more than threshold voxels will be visible. If it is None, the threshold-correction will not work.

background : Niimg-like object or string. Default is stuff.get_bg_ch2()
The background image that the RSA results will be plotted on top of.

type : string 'r' or 't'
The type of result (r-values or t-values).



'a function for plotting the RSA-result by different cuts'

- **plot_brainrsa_montage(img, threshold=None, slice=[6, 6, 6], background=get_bg_ch2bet(), type='r')**

Parameters

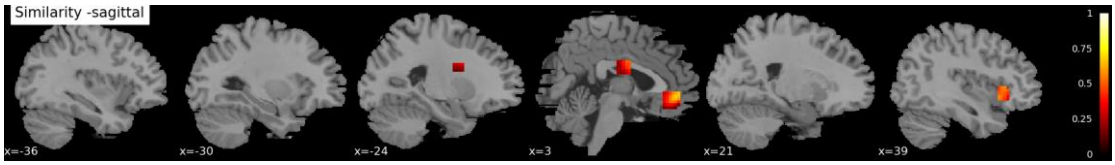
img : string
The file path of the .nii file of the RSA results.

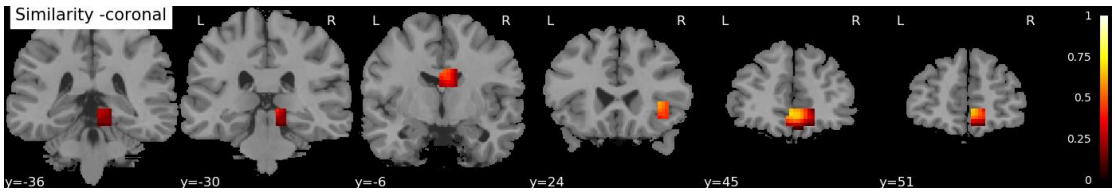
threshold : None or int. Default is None.
The threshold of the number of voxels used in correction.
If threshold=n, only the similarity clusters consisting more than threshold voxels will be visible. If it is None, the threshold-correction will not work.

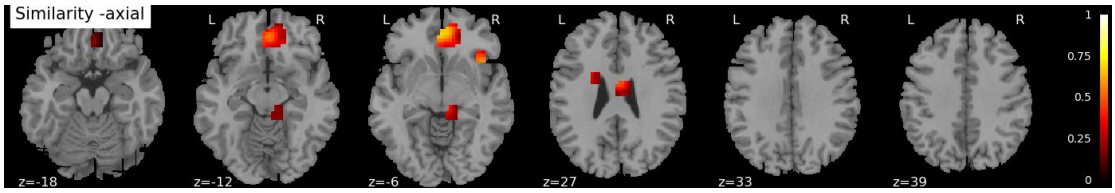
slice : array
The point where the cut is performed.
If slice=[slice_x, slice_y, slice_z], slice_x, slice_y, slice_z represent the coordinates of each cut in the x, y, z direction. If slice=[[slice_x1, slice_x2], [slice_y1, slice_y2], [slice_z1, slice_z2]], slice_x1 & slice_x2 represent the coordinates of each cut in the x direction, slice_y1 & slice_y2 represent the coordinates of each cut in the y direction, slice_z1 & slice_z2 represent the coordinates of each cut in the z direction.

background : Niimg-like object or string. Default is stuff.get_bg_ch2bet()
The background image that the RSA results will be plotted on top of.

type : string 'r' or 't'
The type of result (r-values or t-values).







'a function for plotting the 2-D projection of the RSA-result'

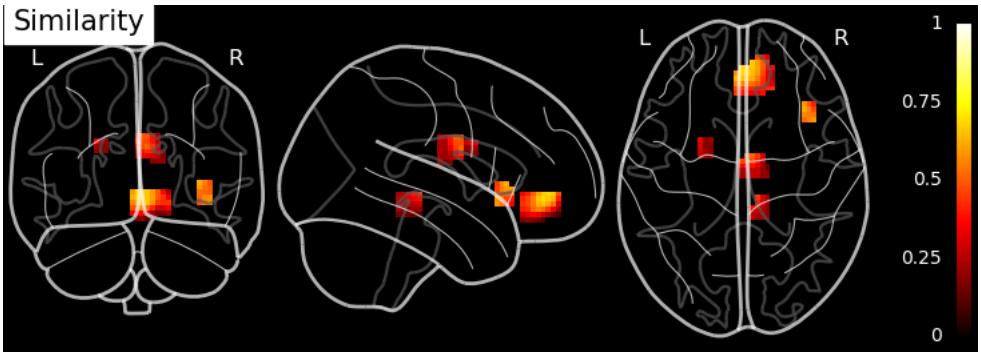
- **plot_brainrsa_glass(img, threshold=None, slice=[6, 6, 6], background=get_bg_ch2bet(), type='r')**

Parameters

img : string
The file path of the .nii file of the RSA results.

threshold : None or int. Default is None.
The threshold of the number of voxels used in correction.
If threshold=n, only the similarity clusters consisting more than threshold voxels will be visible. If it is None, the threshold-correction will not work.

type : string 'r' or 't'
The type of result (r-values or t-values).



'a function for plotting the RSA-result into a brain surface'

- **plot_brainrsa_surface(img, threshold=None, type='r')**

Parameters

img : string

The file path of the .nii file of the RSA results.

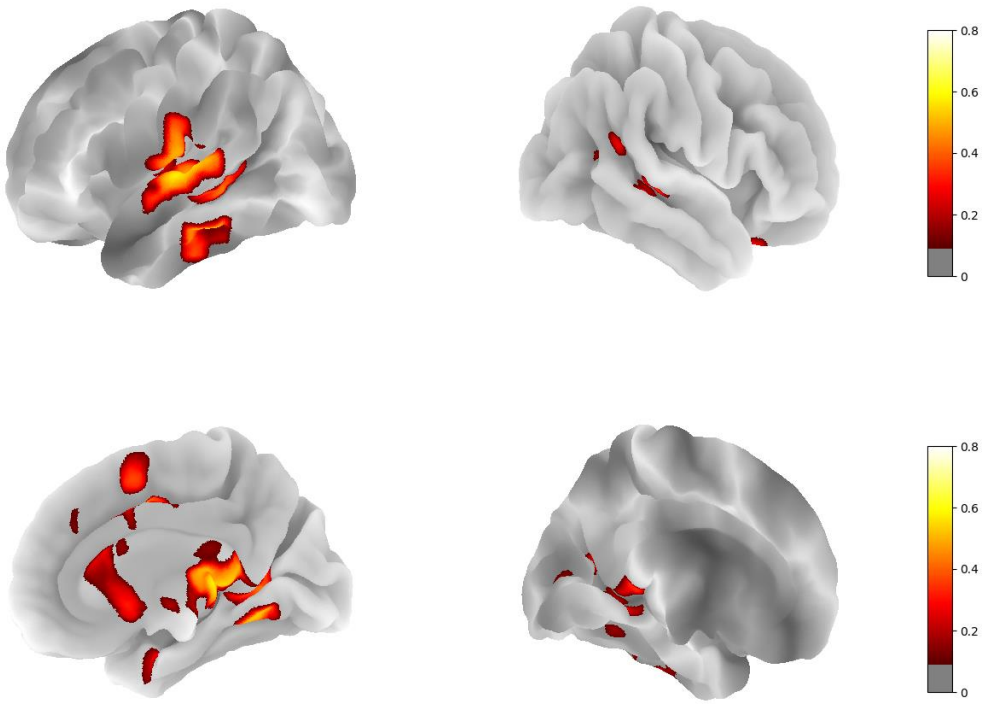
threshold : None or int. Default is None.

The threshold of the number of voxels used in correction.

If threshold=n, only the similarity clusters consisting more than threshold voxels will be visible. If it is None, the threshold-correction will not work.

type : string 'r' or 't'

The type of result (r-values or t-values).



'a function for plotting the RSA-result by a set of images'

- **plot_brainrsa_rlts(img, threshold=None, slice=[6, 6, 6], background=None, type='r')**

Parameters

img : string

The file path of the .nii file of the RSA results.

threshold : None or int. Default is None.

The threshold of the number of voxels used in correction.

If threshold=n, only the similarity clusters consisting more than threshold voxels will be visible. If it is None, the threshold-correction will not work.

background : Niimg-like object or string. Default is None.

The background image that the RSA results will be plotted on top of.

type : string 'r' or 't'
The type of result (r-values or t-values).

Part 14: Other Functions

Module: *stuff.py*

- a module for some simple but important processes

'a function for zeroing the value close to zero'

- **limtozero(x)**

Parameters

x : float

Returns

o

'a function for fMRI RSA results correction by threshold'

- **correct_by_threshold(img, threshold)**

Parameters

img : array
A 3-D array of the fMRI RSA results.
The shape of img should be [nx, ny, nz]. nx, ny, nz represent the shape of the fMRI-img.

threshold : int
The number of voxels used in correction.
If threshold=n, only the similarity clusters consisting more than n voxels will be visualized.

Returns

img : array
A 3-D array of the fMRI RSA results after correction.
The shape of img should be [nx, ny, nz]. nx, ny, nz represent the shape of the fMRI-img.

'a function for getting the affine of the fMRI-img'

- **get_affine(file_name)**

Parameters

file_name : string
The filename of a sample fMRI-img in your experiment.

Returns

affine : array
The position information of the fMRI-image array data in a

reference space.

'a function for FWE-correction for fMRI RSA results'

- **fwe_correct(p, p_threshold)**

Parameters

p : array
The p-value map (3-D).

p_threshold: string
The p threshold.

Returns

fwep : array.
The FWE corrected p-value map.

'a function for FDR-correction for fMRI RSA results'

- **fdr_correct(p, p_threshold)**

Parameters

p : array
The p-value map (3-D).

p_threshold: string
The p threshold.

Returns

fdrp : array.
The FDR corrected p-value map.

'a function for Cluster-wise FWE-correction for fMRI RSA results'

- **custer_fwe_correct(p, p_threshold)**

Parameters

p : array
The p-value map (3-D).

p_threshold1: string
The voxel-wise p threshold.

p_threshold2: string
The cluster-wise p threshold.

Returns

clusterfwep : array.
The Cluster-wise FWE corrected p-value map.

'a function for Cluster-wise FDR-correction for fMRI RSA results'

- **custer_fdr_correct(p, p_threshold)**

Parameters

p : array
The p-value map (3-D).

p_threshold1: string
The voxel-wise p threshold.

p_threshold2: string
The cluster-wise p threshold.

Returns

clusterfwep : array.
The Cluster-wise FDR corrected p-value map.

'a function for getting ch2.nii.gz'

- **get_bg_ch2()**

Returns

path : string
The absolute file path of 'ch2.nii.gz'.

'a function for getting ch2bet.nii.gz'

- **get_bg_ch2bet()**

Returns

path : string.
The absolute file path of 'ch2bet.nii.gz'.

'a function for getting HarvardOxford-cort-maxprob-thro-1mm.nii.gz'

- **get_HOcort()**

Returns

path : string.
The absolute file path of 'HarvardOxford-cort-maxprob-thro-1mm.nii.gz'.

'a function for filtering the data by a ROI mask'

- **datamask(fmri_data, mask_data)**

Parameters

fmri_data : array
The fMRI data.
The shape of fmri_data is [nx, ny, nz]. nx, ny, nz represent the

size of the fMRI data.

mask_data : array
The mask data.
The shape of mask_data is [nx, ny, nz]. nx, ny, nz represent the size of the fMRI data.

Returns

newfmri_data : array
The new fMRI data.
The shape of newfmri_data is [nx, ny, nz]. nx, ny, nz represent the size of the fMRI data.

'a function for projecting the position of a point in matrix coordinate system to the position in MNI coordinate system'

- **position_to_mni(point, affine)**

Parameters

point : list or array
The position in matrix coordinate system.

affine : array or list
The position information of the fMRI-image array data in a reference space.

Returns

newpoint : array
The position in MNI coordinate system.

'a function for projecting the position in MNI coordinate system to the position of a point in matrix coordinate system'

- **mniposition_to(mnipoint, affine)**

Parameters

point : list or array
The position in MNI coordinate system.

affine : array or list
The position information of the fMRI-image array data in a reference space.

Returns

newpoint : array
The position in matrix coordinate system.

'a function for converting data of the mask template to your data template'

- **mask_to(mask, size, affine, filename=None)**

Parameters

mask : string
The file path+filename for the mask of certain template.
size : array or list [nx, ny, nz]
The size of the fMRI-img in your experiments.
affine : array or list
The position information of the fMRI-image array data in a reference space.
filename : string. Default is None - 'newmask.nii'.
The file path+filename for the mask for your data template .nii file.

Notes

A result .nii file of new mask will be generated at the corresponding address of filename.

'a function for permutation test'

- **permutation_test(v1, v2, iter=1000)**

Parameters

v1 : array
Vector 1.
v2 : array
Vector 2.
iter : int. Default is 1000.
The times for iteration.

Returns

p : float
The permutation test result, p-value.

'a function for permutation test for correlation coefficients'

- **permutation_corr(v1, v2, method="spearman", iter=1000)**

Parameters

v1 : array
Vector 1.
v2 : array
Vector 2.
method : string 'spearman' or 'pearson' or 'kendall' or 'similarity' or 'distance'. Default is 'spearman'.
The method to calculate the similarities.
If method='spearman', calculate the Spearman Correlations. If method='pearson', calculate the Pearson Correlations. If methd='kendall', calculate the Kendall tau Correlations. If method='similarity', calculate the Cosine Similarities. If method='distance', calculate the Euclidean Distances.

iter : int. Default is 1000.
The times for iteration.

Returns

p : float
The permutation test result, p-value.

'a function for getting the 1-D & 1-sided cluster-index information'

- **get_cluster_index_1d_1sided(m)**

Parameters

m : array
A significant vector.
The values in m should be 0 or 1, which represent not significant point or significant point, respectively.

Returns

index_v : array
The cluster-index vector.
index_n : int
The number of clusters.

'a function for getting the 1-D & 1-sided cluster-index information'

- **get_cluster_index_1d_2sided(m)**

Parameters

m : array
A significant vector.
The values in m should be 0 or 1 or -1, which represent not significant point or significantly higher point or significantly less point, respectively.

Returns

index_v1 : array
The "greater" cluster-index vector.
index_n1 : int
The number of "greater" clusters.
index_v2 : array
The "less" cluster-index vector.
index_n2 : int
The number of "less" clusters.

'a function for getting the 2-D & 1-sided cluster-index information'

- **get_cluster_index_2d_1sided(m)**

Parameters

m : array
A significant vector.
The values in m should be 0 or 1, which represent not significant point or significant point, respectively.

Returns

index_v : array
The cluster-index vector.
index_n : int
The number of clusters.

'a function for getting the 2-D & 2-sided cluster-index information'

- **get_cluster_index_2d_2sided(m)**

Parameters

m : array
A significant vector.
The values in m should be 0 or 1 or -1, which represent not significant point or significantly higher point or significantly less point, respectively.

Returns

index_v1 : array
The "greater" cluster-index vector.
index_n1 : int
The number of "greater" clusters.
index_v2 : array
The "less" cluster-index vector.
index_n2 : int
The number of "less" clusters.

'a function for 1-sample & 1-sided cluster-based permutation test for 1-D results'

- **clusterbased_permutation_1d_1samp_1sided(results, level=0, p_threshold=0.05, clusterp_threshold=0.05, n_threshold=2, iter=1000)**

Parameters

results : array
A result matrix.
The shape of results should be [n_subs, x]. n_subs represents the number of subjects.
level : float. Default is 0.
An expected value in null hypothesis. (Here, results > level)
p_threshold : float. Default is 0.05.
The threshold of p-values.
clusterp_threshold : float. Default is 0.05.

The threshold of cluster-defining p-values.

n_threshold : int. Default is 2.

The threshold of number of values in one cluster (number of values per cluster > n_threshold).

iter : int. Default is 1000.

The times for iteration.

Returns

ps : float

The permutation test resultz, p-values.

The shape of ps is [x]. The values in ps should be 0 or 1, which represent not significant point or significant point after cluster-based permutation test, respectively.

'a function for 1-sample & 2-sided cluster-based permutation test for 1-D results'

- **clusterbased_permutation_1d_1samp_2sided(results, level=0, p_threshold=0.05, clusterp_threshold=0.05, n_threshold=2, iter=1000)**

Parameters

results : array

A result matrix.

The shape of results should be [n_subs, x]. n_subs represents the number of subjects.

level : float. Default is 0.

An expected value in null hypothesis. (Here, results > level)

p_threshold : float. Default is 0.05.

The threshold of p-values.

clusterp_threshold : float. Default is 0.05.

The threshold of cluster-defining p-values.

n_threshold : int. Default is 2.

The threshold of number of values in one cluster (number of values per cluster > n_threshold).

iter : int. Default is 1000.

The times for iteration.

Returns

ps : float

The permutation test resultz, p-values.

The shape of ps is [x]. The values in ps should be 0 or 1 or -1, which represent not significant point or significant greater point or significant less point after cluster-based permutation test, respectively.

'a function for 1-sided cluster-based permutation test for 1-D results'

- **clusterbased_permutation_1d_1sided(result1, result2,**

  p_threshold=0.05, clusterp_threshold=0.05, n_threshold=2, iter=1000)

Parameters

results1 : array
A result matrix under condition1.
The shape of results1 should be [n_subs, x]. n_subs represents the number of subjects.

results2 : array
A result matrix under condition2.
The shape of results2 should be [n_subs, x]. n_subs represents the number of subjects. (Here, results1 > results2)

p_threshold : float. Default is 0.05.
The threshold of p-values.

clusterp_threshold : float. Default is 0.05.
The threshold of cluster-defining p-values.

n_threshold : int. Default is 2.
The threshold of number of values in one cluster (number of values per cluster > n_threshold).

iter : int. Default is 1000.
The times for iteration.

Returns

ps : float
The permutation test resultz, p-values.
The shape of ps is [x]. The values in ps should be 0 or 1, which represent not significant point or significant point after cluster-based permutation test, respectively.

'a function for 2-sided cluster based permutation test for 1-D results'

- **clusterbased_permutation_1d_1samp_2sided(results1, results2,**

  p_threshold=0.05, clusterp_threshold=0.05, n_threshold=2, iter=1000)

Parameters

results1 : array
A result matrix under condition1.
The shape of results1 should be [n_subs, x]. n_subs represents the number of subjects.

results2 : array
A result matrix under condition2.
The shape of results2 should be [n_subs, x]. n_subs represents the number of subjects. (Here, results1 > results2)

p_threshold : float. Default is 0.05.
The threshold of p-values.

clusterp_threshold : float. Default is 0.05.
The threshold of cluster-defining p-values.
n_threshold : int. Default is 2.
The threshold of number of values in one cluster (number of values per cluster > n_threshold).
iter : int. Default is 1000.
The times for iteration.

Returns

ps : float
The permutation test resultz, p-values.
The shape of ps is [x]. The values in ps should be 0 or 1 or -1, which represent not significant point or significant greater point or significant less point after cluster-based permutation test, respectively.

'a function for 1-sample & 1-sided cluster based permutation test for 2-D results'

- **clusterbased_permutation_2d_1samp_1sided(results, level=0, p_threshold=0.05, clusterp_threshold=0.05, n_threshold=4, iter=1000)**

Parameters

results : array
A result matrix.
The shape of results should be [n_subs, x1, x2]. n_subs represents the number of subjects.
level : float. Default is 0.
An expected value in null hypothesis. (Here, results > level)
p_threshold : float. Default is 0.05.
The threshold of p-values.
clusterp_threshold : float. Default is 0.05.
The threshold of cluster-defining p-values.
n_threshold : int. Default is 4.
The threshold of number of values in one cluster (number of values per cluster > n_threshold).
iter : int. Default is 1000.
The times for iteration.

Returns

ps : float
The permutation test resultz, p-values.
The shape of ps is [x1, x2]. The values in ps should be 0 or 1, which represent not significant point or significant point after cluster-based permutation test, respectively.

'a function for 1-sample & 2-sided cluster based permutation test for 2-D results'

- **clusterbased_permutation_2d_1samp_2sided(results, level=0, p_threshold=0.05, clusterp_threshold=0.05, n_threshold=4, iter=1000)**

Parameters

results : array
A result matrix.
The shape of results should be [n_subs, x1, x2]. n_subs represents the number of subjects.

level : float. Default is 0.
An expected value in null hypothesis. (Here, results > level)

p_threshold : float. Default is 0.05.
The threshold of p-values.

clusterp_threshold : float. Default is 0.05.
The threshold of cluster-defining p-values.

n_threshold : int. Default is 4.
The threshold of number of values in one cluster (number of values per cluster > n_threshold).

iter : int. Default is 1000.
The times for iteration.

Returns

ps : float
The permutation test resultz, p-values.
The shape of ps is [x1, x2]. The values in ps should be 0 or 1 or -1, which represent not significant point or significant greater point or significant less point after cluster-based permutation test, respectively.

'a function for 1-sided cluster based permutation test for 2-D results'

- **clusterbased_permutation_2d_1sided(results1, result2, level=0, p_threshold=0.05, clusterp_threshold=0.05, n_threshold=4, iter=1000)**

Parameters

results1 : array
A result matrix under condition1.
The shape of results1 should be [n_subs, x1, x2]. n_subs represents the number of subjects.

results2 : array
A result matrix under condition2.
The shape of result2s should be [n_subs, x1, x2]. n_subs represents the number of subjects. (results1 > results2)

p_threshold : float. Default is 0.05.
The threshold of p-values.

clusterp_threshold : float. Default is 0.05.
The threshold of cluster-defining p-values.

n_threshold : int. Default is 4.
The threshold of number of values in one cluster (number of values per cluster > n_threshold).
iter : int. Default is 1000.
The times for iteration.

Returns

ps : float
The permutation test resultz, p-values.
The shape of ps is [x1, x2]. The values in ps should be 0 or 1, which represent not significant point or significant point after cluster-based permutation test, respectively.

'a function for 2-sided cluster based permutation test for 2-D results'

- **clusterbased_permutation_2d_2sided(results1, result2, level=0, p_threshold=0.05, clusterp_threshold=0.05, n_threshold=4, iter=1000)**

Parameters

results1 : array
A result matrix under condition1.
The shape of results1 should be [n_subs, x1, x2]. n_subs represents the number of subjects.
results2 : array
A result matrix under condition2.
The shape of result2s should be [n_subs, x1, x2]. n_subs represents the number of subjects. (results1 > results2)
p_threshold : float. Default is 0.05.
The threshold of p-values.
clusterp_threshold : float. Default is 0.05.
The threshold of cluster-defining p-values.
n_threshold : int. Default is 4.
The threshold of number of values in one cluster (number of values per cluster > n_threshold).
iter : int. Default is 1000.
The times for iteration.

Returns

ps : float
The permutation test resultz, p-values.
The shape of ps is [x1, x2]. The values in ps should be 0 or 1 or -1, which represent not significant point or significant greater point or significantly less point after cluster-based permutation test, respectively.

'a function for smoothing the 1-D results'

- **smooth_1d(x, n=5)**

Parameters

x : array
The results.
The shape of x should be [n_subs, n_ts]. n_subs, n_ts represent the number of subjects and the number of time-points.

n : int. Default is 5.
The smoothing step is n.

Returns

x_smooth : array
The results after smoothing.
The shape of x_smooth should be [n_subs, n_ts]. n_subs, n_ts represent the number of subjects and the number of time-points.

'a function for smoothing the 2-D results'

- **smooth_2d(x, n=5)**

Parameters

x : array
The results.
The shape of x should be [n_subs, n_ts1, n_ts2]. n_subs represents the number of subjects. n_ts1, n_ts2 represent the number of time-points.

n : int. Default is 5.
The smoothing step is n.

Returns

x_smooth : array
The results after smoothing.
The shape of x should be [n_subs, n_ts1, n_ts2]. n_subs represents the number of subjects. n_ts1, n_ts2 represent the number of time-points.

Part 15: Demo

Currently, we provide three demos for showing how to use NeuroRA.

Demo1 is a demo based on the publicly available visual-92-categories-task MEG datasets. (Reference: *Cichy, R. M., Pantazis, D., & Oliva, A. "Resolving human object recognition in space and time." Nature neuroscience (2014): 17(3), 455-462.*)

Demo2 is a demo based on the publicly available Haxby fMRI datasets. (Reference: *Haxby, J. V. (2001). Distributed and Overlapping Representations of Faces and Objects in Ventral Temporal Cortex. Science, 293(5539), 2425–2430.*)

Demo3 is a demo for comparing classification-based EEG decoding and EEG RSA based on the data of Bae & Luck's work in 2018. (Reference: *Bae, G.Y., Luck, S.J. (2018). Dissociable decoding of spatial attention and working memory from eeg oscillations and sustained potentials. The Journal of Neuroscience, 38(2), 409-422.*)

Users can see more details here:
https://github.com/zitonglu1996/NeuroRA/tree/master/demo.
Both .py files and .ipynb files are provided.

Users can see more details below:
GitHub Website: https://github.com/ZitongLu1996/NeuroRA
NeuroRA Website: https://ZitongLu1996.github.io/NeuroRA/
Online Documentation: https://neurora.github.io/documentation
PyPi Website: https://pypi.org/project/neurora/
Paper: https://doi.org/10.3389/fninf.2020.563669