Tutorial of NeuroRA Version 1.1

Updated by 2020-04-22

This Tutorial of NeuroRA provides information on how to use the NeuroRA including its easy-to-use functions.

Before you read it, you only need to spend a little time learning the basic Python syntax and this toolkit is easy to understand. In addition, it would be better if you are familiar with Python, especially the matrix operations based on NumPy.

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**

Institute of Cognitive Neuroscience

East China Normal University

This tutorial consists of these parts:

- Introduction & Installation
- Data Conversion
- Calculate the neural pattern similarity (NPS)
- Calculate the Spatiotemporal pattern similarity (STPS)
- Calculate the Representational Dissimilarity Matrix (RDM)
- Representational Similarity Analysis (RSA)
- Save as a NIfTI file (for fMRI)
- Visualization for results
- Others
- Demo based on NeuroRA

Content

Part 1: Introduction

NueorRA 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, ECoG, electrophysiological and fMRI data.

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.

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.

MNE-Python: a Python software for exploring, visualizing, and analyzing human neurophysiological data.

Paper

Lu, Z., & Ku, Y. *NeuroRA: A Python toolbox of representational analysis from multi-modal neural data.* (bioRxiv: https://doi.org/10.1101/2020.03.25.008086)

Part 2: Data Conversion

Type of Neural Bata	Data Conversion Scheme
fMRI	Use *Nibabel* (https://nipy.org/nibabel/) to load fMRI data. *import nibabel as nib* *fmrifilename = "demo.nii" # the fmri data file name with full address* *data = nib.load(fmrifilename).get_fdata() # load fMRI data as ndarray* *……*
EEG/MEG	Use *MATLAB EEGLab* (http://sccn.ucsd.edu/eeglab/) to do preprocessing and obtain .mat files, and use *Scipy* (https://www.scipy.org) to load EEG data (.mat). *import scipy.io as sio* *filename = "demo.mat" # the EEG/MEG data file name with full address* *data = sio.loadmat(filename)["data"] # load EEG/MEG data as ndarray* *……* Or use *MNE* (https://mne-tools.github.io) to do preprocessing and return *ndarray*-type data.
fNIRS	For raw data from device, use *Numpy* (http://www.numpy.org) to load fNIRS data (.txt or .csv). *import numpy as np* *txtfilename = "demo.txt" # the fNIRS data file name with full address* *csvfilename = "demo.csv"* *data = np.loadtxt(txtfilename) # load fNIRS data as ndarray* *data = np.loadtxt(csvfilename, delimiter, usecols, unpack)* *……*
ECoG/sEEG	Use *Brainstorm* (https://neuroimage.usc.edu/brainstorm/) to do preprocessing and obtain .mat files, and use Scipy to load ECoG data (.mat).
Electrophysiology	Use *pyABF* (https://github.com/swharden/pyABF) to load electrophysiology data (.abf). *import pyabf* *abf = pyabf.ABF("demo.abf") # the electrophysiology data file name with full address* *abf.setSweep(sweepNumber, channel) # access sweep data* *data = abf.sweepY # get sweep data with sweepY* *……*
Two functions, *NumPy.reshape()* & *NumPy.transpose()*, are recommended for further data transformation	

Part 3: Calculate the Neural Pattern Similarity

Module: ***nps_cal.py***

"A module for calculating the neural pattern similarity based on neural data"

- nps(data, time_win=5, time_step=5, sub_opt=0)

  A function for calculating the neural pattern similarity for EEG-like data.

  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 0.*

  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.

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

A function for calculating the neural pattern similarity of fMRI data (searchlight).

Parameters:

fmri_data: *array [nx, ny, nz].*

The fmri data.

The shape of fmri_data must be [n_cons, n_chls, nx, ny, nz]. n_cons, n_chls, nx, ny, nz represent the number of conidtions, the number of channels & the size of fMRI-img, respectively.

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

The size of the fMRI-img.

nx, ny, nz 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.

Returns:

NPS: *array.*

The fMRI NPS for searchlight.

The shape of NPS 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

◆ nps_fmri_roi(fmri_data, mask_data)

A function for calculating the neural pattern similarity of fMRI data (for ROI).

Parameters:

fmri_data: *array [nx, ny, nz].*

The fmri data.

The shape of fmri_data must be [n_cons, n_chls, nx, ny, nz]. n_cons, n_chls, nx, ny, nz represent the number of conidtions, 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:

NPS: *array.*

The fMRI NPS for ROI.

The shape of NPS is [2]. 2 representation a r-value and a p-value.

Part 4: Calculate the Spatiotemporal Pattern Similarity

Module: ***stps_cal.py***

"A module for calculating the spatiotemporal pattern similarity based on neural data"

- nps(data1, data2, time_win=20, time_step=1, sub_opt=0)

  A function for calculating the spatiotemporal pattern similarities (STPS).

  Parameters:

  data1: *array.*

  The data under condition 1.

  The shape of data1 must be [n_subs1, n_chls, n_trials1, n_ts]. n_subs1, n_chls, n_trials1, n_ts represent the number of subjects, the number of channels or regions, the number of trials and the number of time-points of the data under condition 1.

  data2: *array.*

  The data under condition 2.

  The shape of data2 must be [n_subs2, n_chls, n_trials2, n_ts]. n_subs1, n_chls, n_trials1, n_ts represent the number of subjects, the number of channels or regions, the number of trials and the number of time-points of the data under condition 2.

  time_win: *int. Default is 20.*

  Set a time-window for calculating the NPS 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_chls, int((n_ts-time_win)/time_step) +1, 2]. 2 representation a t-value and a p-value.

Part 5: Calculate the RDM

Module: ***rdm_cal.py***

"A module for calculating the representational dissimilarity matrix based on multimode neural data"

- bhvRDM(bhv_data, sub_opt=0)

  A function for calculating the RDM based on behavioral data.

  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 0.*

  Calculate the RDM for each subject or not.

  If sub_opt=0, return only one RDM based on all data. If sub_opt=1, return n_subs RDMs based on each subject's data.

  Returns:

  RDM(s): *array*

  The behavioral RDM.

  If sub_opt=0, return only one RDM. The shape is [n_cons, n_cons]. If sub_opt=1, return n_subs RDMs. The shape is [n_subs, n_cons, n_cons].

- eegRDM(EEG_data, sub_opt=0, chl_opt=0, time_opt=0, time_win=5, time_step=5)

  A function for calculating the RDM(s) based on EEG/MEG/fNIRS data.

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 0.*

Calculate the RDM for each subject or not.

If sub_opt=0, return only one RDM based on all data. If sub_opt=1, return n_subs RDMs based on each subject's data

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 NPS 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.

Returns:

RDM(s): *array.*

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].

◆ ecogRDM(ele_data, opt="all" , time_win=5, time_step=5)

A function for calculating the RDM(s) based on ECoG/sEEG/electrophysiological data.

Parameters:

ele_data: *array.*

The ECoG/sEEG/electrophysiology data.

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

opt: *string 'channel', 'time' or 'all'. Default is 'all'.*

Calculate the RDM for each channel or for each time-point or for the whole data.

If opt='channel', return n_chls RDMs based on each channel's data. If opt='time', return int(ts/time_win) RDMs based on each time-point's data respectively. If opt='allin', return only one RDM based on all data.

time_win: *int. Default is 5.*

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

Only when opt='time', 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 opt='time', time_step works.

Returns:

RDM(s): *array.*

The ECoG/sEEG/electrophysiology RDM.

If opt='channel', return n_chls RDM. The shape is [n_chls, n_cons, n_cons]. If opt='time', 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 opt='all', return one RDM. The shape is [n_cons, n_cons].

◆ fmriRDM(fmri_data, ksize=[3, 3, 3], strides=[1, 1, 1])

A function for calculating the RDM based on fMRI data (searchlight).

Parameters:

fmri_data: *array.*

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

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

The size of the fMRI-img. nx, ny, nz 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.

Returns:

RDMs: *array.*

The fMRI-Searchlight RDM.

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

- fmriRDM_roi(fmri_data, mask_data)

A function for calculating the RDM based on fMRI data of a ROI.

Parameters:

fmri_data: *array [nx, ny, nz].*

The fmri data.

The shape of fmri_data must be [n_cons, n_chls, nx, ny, nz]. n_cons, n_chls, nx, ny, nz represent the number of conidtions, 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:

rdm: *array.*

The fMRI-ROI RDM.

The shape of RDM is [n_cons, n_cons].

Part 6: Representational Similarity Analysis

Module: ***rdm_corr.py***

"A module for calculating the Similarity/Correlation Coefficient between two RDMs"

- rdm_correlation_spearman(RDM1, RDM2, rescale=False)

A function for calculating the Spearman correlation coefficient between two RDMs.

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:

corr: *array [r, p].*

The Spearman Correlation result.

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

- rdm_correlation_pearson(RDM1, RDM2, rescale=False)

A function for calculating the Pearson correlation coefficient between two

RDMs.

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:

corr: *array [r, p].*

The Pearson Correlation result.

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

◆ rdm_correlation_kendall(RDM1, RDM2, rescale=False)

A function for calculating the Kendalls tau correlation coefficient between two RDMs.

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:

corr: *array [r, p].*

The Kendalls tau Correlation result.

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

◆ rdm_similarity(RDM1, RDM2, rescale=False)

A function for calculating the Cosine Similarity between two RDMs.

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: *array [r, p].*

The Cosine Similarity result.

The shape of corr is [2], corr[0] is the Cosine Similarity result and corr[1] is 0.

- rdm_distance(RDM1, RDM2, rescale=False)

A function for calculating the Euclidean Distances between two RDMs.

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: *array [r, p].*

The Euclidean Distance result.

The shape of corr is [2], corr[0] is the Euclidean Distance result and corr[1] is 0.

- rdm_permutation (RDM1, RDM2, iter=1000, rescale=False)

A function for permutation test between two RDMs.

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.

iter: *int*

The number of iterations.

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:

p: *float*

The permutation test result, p-value.

Module: ***corr_cal.py***

"A module for calculating the Similarity/Correlation Coefficient between two different modes data"

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

A function for calculating the Similarity/Correlation Coefficient between behavioral data and EEG/MEG/fNIRS data.

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.

Returns:

corr(s): *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 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.

- bhvANDecog_corr(bhv_data, ele_data, time_win=5, time_win=5, ecog_opt="allin", method="spearman" , rescale=False)

A function for calculating the Similarity/Correlation Coefficient between behavioral data and ECoG/electricophysiological data.

Parameters:

bhv_data: *array.*

The behavioral data.

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

ele_data: *array.*

The ECoG/sEEG/electrophysiological data.

The shape of EEGdata must be [n_cons, n_trials, n_chls, n_ts]. n_cons, n_trials, n_chls & n_ts represent the number of conidtions, 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 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.

ecog_opt: *string 'channel', 'time' or 'all'. Default is 'all'.*

Calculate the RDM for each channel or for each time-point or for the whole data.

If ecog_opt='channel', return n_chls RDMs based on each channel's data. If ecog_opt='time', return int((n_ts-time_win)/time_step)+1 RDMs based on each time-point's data respectively. If ecog_opt='allin', return only one RDM based on all data.

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.

Returns:

corr(s): *array.*

The similarities between behavioral data and ECoG/sEEG/electrophysiology.

If opt='channel', 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 opt='time', return int(n_ts/time_win) 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 opt='all', 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.

◆ bhvANDfmri_corr(bhv_data, fmri_data, ksize=[3, 3, 3], strides=[1, 1, 1], method="spearman" , rescale=False)

A function for calculating the Similarity/Correlation Coefficient between behavioral data and fMRI data (searchlight).

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, n_chls, nx, ny, nz]. n_cons, n_chls, nx, ny, nz represent the number of conidtions, the number of channels & the size of fMRI-img, respectively.

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

The size of the fMRI-img. nx, ny, nz 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.

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.

Returns:

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.

- eegANDfmri_corr(eeg_data, fmri_data, chl_opt=0, ksize=[3, 3, 3], strides=[1, 1, 1], method="spearman" , rescale=False)

A function for calculating the Similarity/Correlation Coefficient between EEG/MEG/fNIRS data and fMRI data (searchlight).

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, n_chls, nx, ny, nz]. n_cons, n_chls, nx, ny, nz represent the number of conidtions, the number of channels & 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 fMRI-img. nx, ny, nz 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.

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.

Returns:

corrs: *array.*

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

If chl_opt=1, the shape of RDMs 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, 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.

Module: ***corr_cal_by_rdm.py***

"A module for calculating the Similarity/Correlation Coefficient between RDMs by different modes"

◆ eegrdms_corr(demo_rdm, eeg_rdms, method="spearman" , rescale=False)

A function for calculating the Similarity/Correlation Coefficient between RDMs based on EEG/MEG/fNIRS/ECoG/sEEG/electrophysiological data and a demo RDM.

Parameters:

demo_rdm: *array [n_cons, n_cons].*

A demo RDM.

eeg_rdm: *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.

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.

◆ fmrirdms_corr(demo_rdm, fMRI_rdms, method="spearman" , rescale=False)

A function for calculating the Similarity/Correlation Coefficient between fMRI RDMs (searchlight) and a demo RDM.

Parameters:

demo_rdm: *array [n_cons, n_cons].*

A demo RDM.

fmri_rdm: *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.

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.

Part 7: Save as a NIfTI file (for fMRI)

Module: ***corr_to_nii.py***

"A module for saving the correlation coefficients in a .nii file"

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

  A function for saving the searchlight RSA results as a NIfTI file for fMRI.

  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.

  filename: *string.*

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

  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.

  corr_mask: *string.*

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

  It can just be one of your fMRI data file 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 fMRI-img. nx, ny, nz 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'. 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=None, no correction. Only when p<1, correct_method works.

smooth: *bool True or False*

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 RSA result.

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

Part 8: Visualization for Results

Module: ***rsa_plot.py***

"A module for plotting the NeuroRA results"

- plot_rdm(rdm, rescale=False, conditions=None, con_fontsize=12, cmap=None)

  A function for plotting the RDM.

  Parameters:

  rdm: *array or list [n_cons, n_cons].*

  A representational dissimilarity matrix.

  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.

  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.

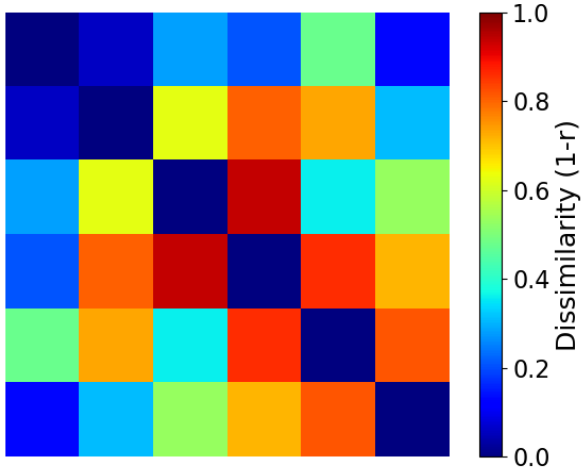
  con_fontsize: *int or float. Default is 12.*

  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'.



- plot_rdm_withvalue(rdm, value_fontsize=10, conditions=None, con_fontsize=12, cmap=None)

A function for plotting the RDM with visible values.

Parameters:

rdm: *array or list [n_cons, n_cons]*

A representational dissimilarity matrix.

value_fontsize: *int or float. Default is 10.*

The value_fontsize of the values on the RDM.

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.

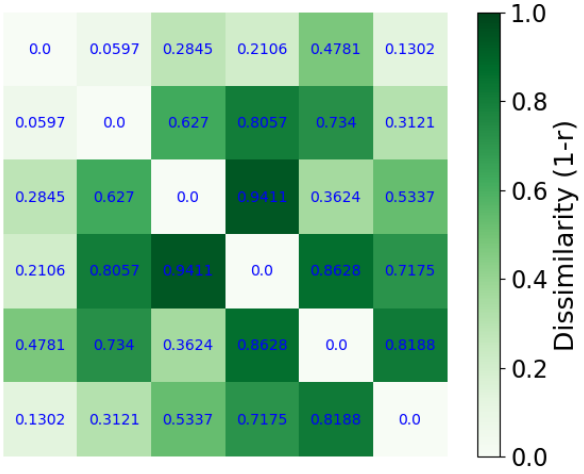
con_fontsize: *int or float. Default is 12.*

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 'Greens'.



◆ plot_corrs_by_time(corrs, labels=None, time_unit=[0, 1])

A function for plotting the correlation coefficients by time sequence.

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.

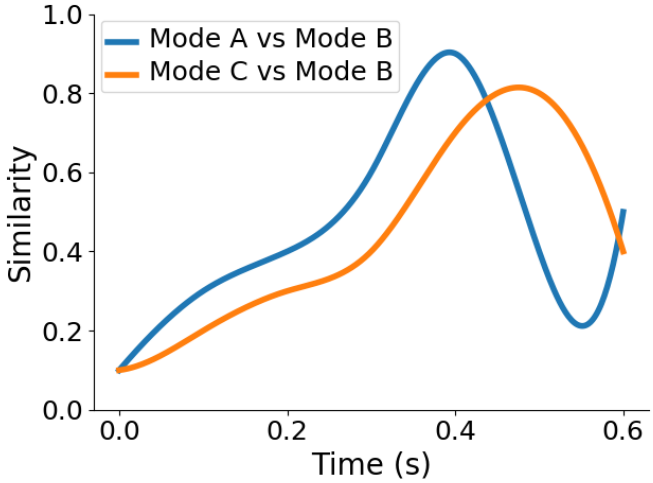
labels: *string-array or string-list or None. Default is None.*

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.



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

A function for plotting the hotmap of the correlation coefficients for channels/regions by time sequence.

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.

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].*

The corrs view lims.

smooth: *bool True or False.*

Smooth the results or not.

figsize: *array or list, [size_X, size_Y]*

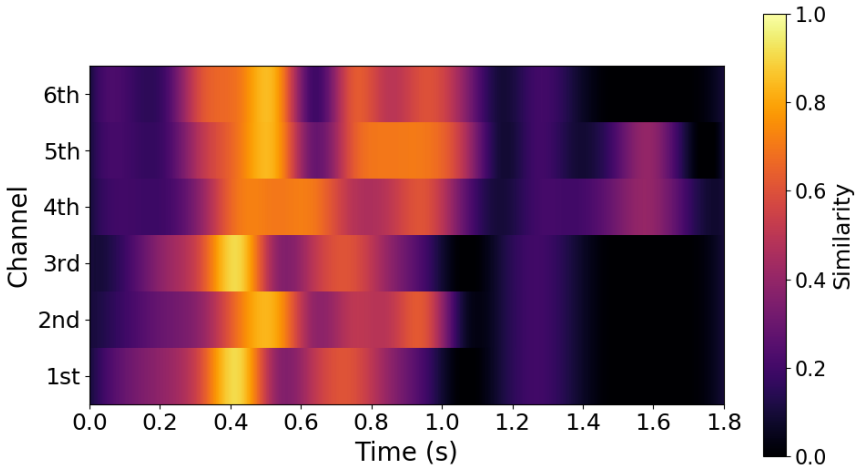
The size of the figure.

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'.



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

A function for plotting the hotmap of neural pattern similarities for channels/regions by time sequence.

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].*

The corrs view lims.

abs: *boolean True or False.*

Change the similarities into absolute values or not.

smooth: *bool True or False.*

Smooth the results or not.

figsize: *array or list, [size_X, size_Y]*

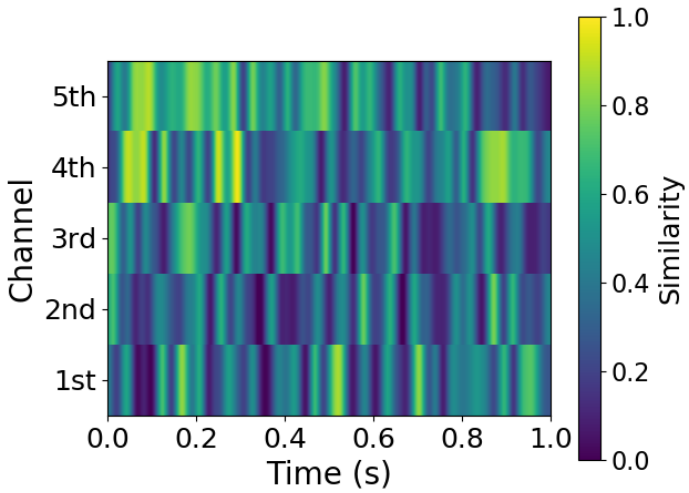
The size of the figure.

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'.



◆ plot_brainrsa_region(img, threshold=None, background=get_bg_ch2())

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

Parameters:

img: *string.*

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

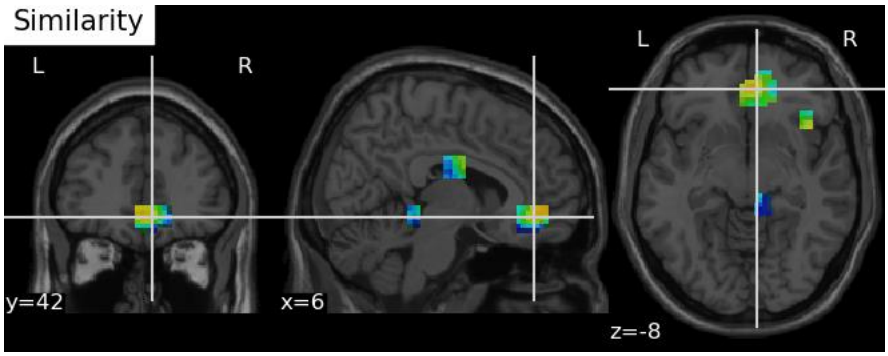
threshold: *None or int. Default is None.*

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.



- plot_brainrsa_montage (img, threshold=None, slice=[6, 6, 6], background=get_bg_ch2bet())

A function for plotting the RSA-result by different cuts.

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.

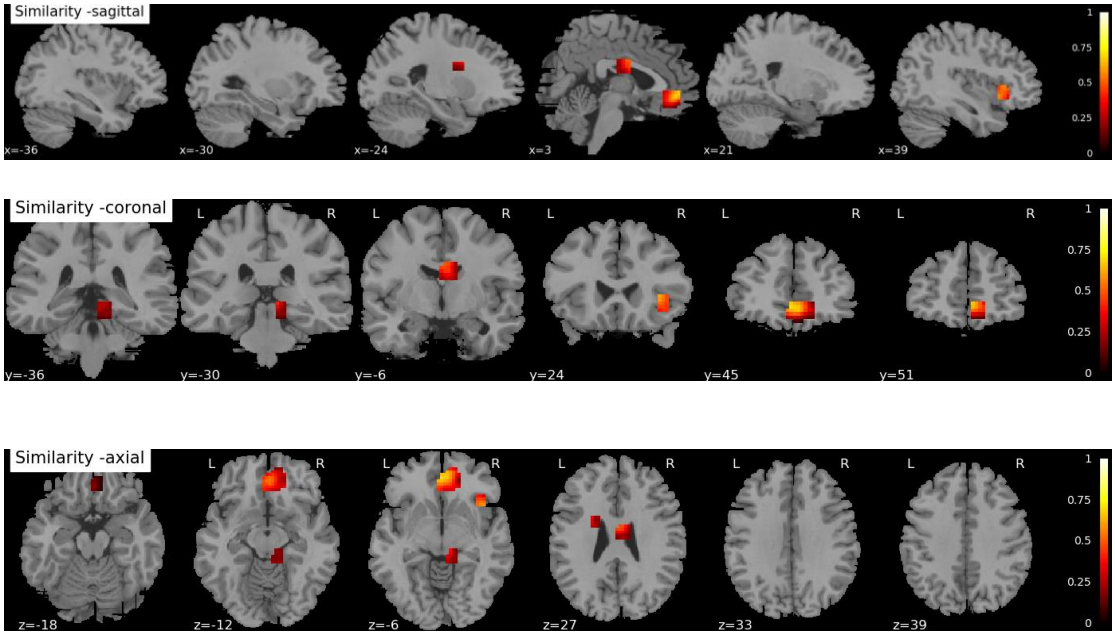
slice: *array.*

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.



◆ plot_brainrsa_glass (img, threshold=None)

A function for plotting the 2-D projection of the RSA-result.

Parameters:

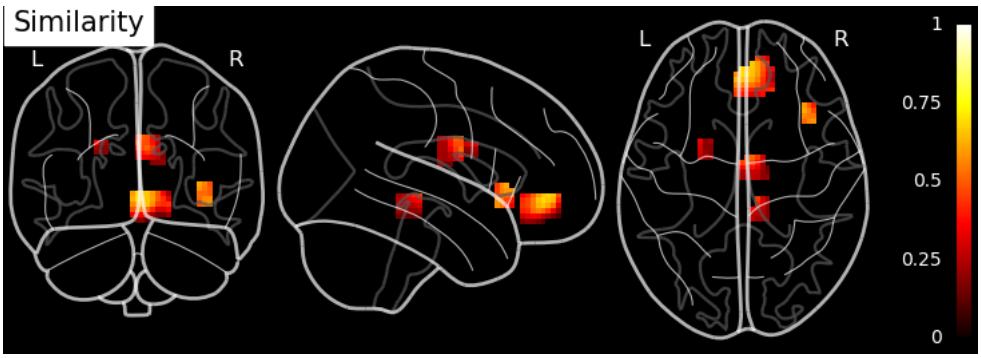
img: *string.*

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.



◆ plot_brainrsa_surface (img, threshold=None)

A function for plotting the RSA-result into a brain surface.

Parameters:

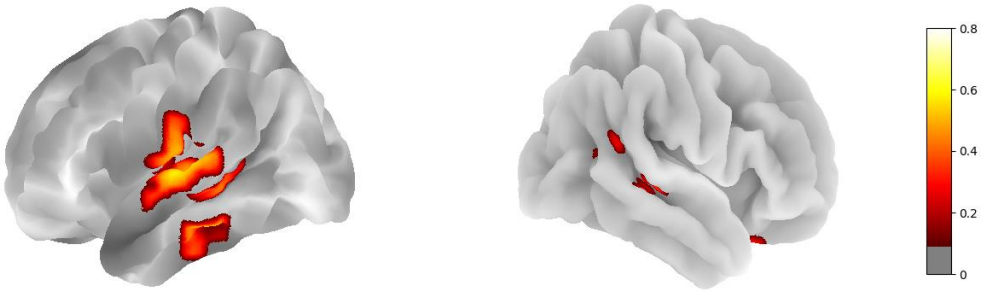
img: *string.*

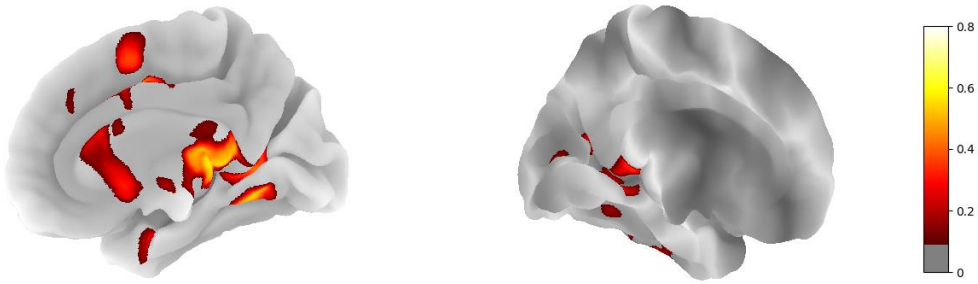
The file path of the .nii file of the RSA results.

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.





◆ plot_brainrsa_rlts (img, threshold=None, slice=[6, 6, 6], background=None)

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

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.

Part 9: Others

Module: ***stuff.py***

"A module for some simple but important processes"

- limtozero(x)

  A function for zeroing the value close to zero.

  Parameters:

  x: *float.*

  A value.

  Returns:

  0

- get_affine(file_name)

  A function for getting the affine of the fMRI-img.

  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.

- fwe_correct(p)

  A function for FWE correction for fMRI RSA results.

  Parameters:

p: *array.*

The p-value map (3-D).

Returns:

correctp: *array*

The FWE corrected p-value map.

◆ fdr_correct(p)

A function for FDR correction for fMRI RSA results.

Parameters:

p: *array.*

The p-value map (3-D).

Returns:

correctp: *array.*

The FDR corrected p-value map.

◆ correct_by_threshold(img, threshold)

A function for correcting the fMRI RSA results by 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.

- get_bg_ch2()

A function for getting the path of 'ch2.nii.gz'.

Returns:

path: *string.*

The absolute file path of the file "ch2.nii.gz".

- get_bg_ch2bet()

A function for getting the path of 'ch2bet.nii.gz'.

Returns:

path: *string.*

The absolute file path of the file "ch2bet.nii.gz".

- datamask(fmri_data, mask_data)

A function for filtering the data by a ROI mask.

Parameters:

fmri_data: *array.*

The fmri data.

The shape of fmri_data must be [nx, ny, nz]. nx, ny, nz represent the size of the fMRI data.

mask_data: *array.*

The fmri data.

The shape of mask_data must be [nx, ny, nz]. nx, ny, nz represent the size of the fMRI data.

Parameters:

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.

Part 10: Demo

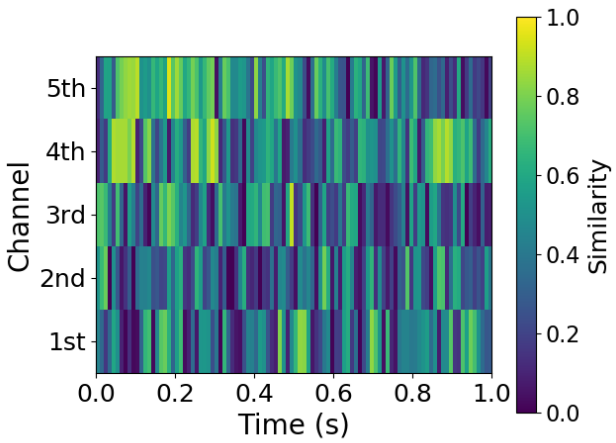
The EEG/MEG Demo

Here 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.) MNE-Python has been used to load this dataset.

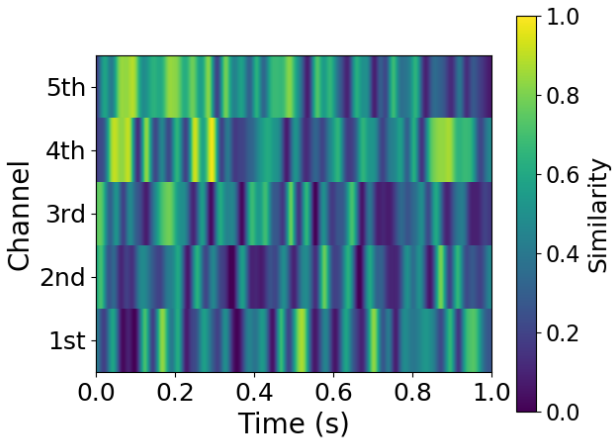
```
# -*- coding: utf-8 -*-

' a demo based on visual-92-categories-task MEG data '
# Users can learn how to use Neurora to do research based on EEG/MEG etc
data.

__author__ = 'Zitong Lu'

import numpy as np
import os.path as op
from pandas import read_csv
import mne
from mne.io import read_raw_fif
from mne.datasets import visual_92_categories
from neurora.nps_cal import nps
from neurora.rdm_cal import eegRDM
from neurora.rdm_corr import rdm_correlation_spearman
from neurora.corr_cal_by_rdm import rdms_corr
from neurora.rsa_plot import plot_rdm, plot_corrs_by_time,
plot_nps_hotmap, plot_corrs_hotmap

"""**********      Section 1: loading example data       **********"""
""" Here, we use MNE-Python toolbox for loading data and processing """
""" you can learn this process from MNE-Python (https://mne-
tools.github.io/stable/index.html) """

data_path = visual_92_categories.data_path()
fname = op.join(data_path, 'visual_stimuli.csv')
conds = read_csv(fname)
conditions = []
```

```
for c in conds.values:
    cond_tags = list(c[:2])
    cond_tags += [('not-' if i == 0 else '') + conds.columns[k]
                  for k, i in enumerate(c[2:], 2)]
    conditions.append('/'.join(map(str, cond_tags)))
event_id = dict(zip(conditions, conds.trigger + 1))
print(event_id)
sub_id = [0, 1, 2]
megdata = np.zeros([3, 92, 306, 1101], dtype=np.float32)
subindex = 0
for id in sub_id:
    fname = op.join(data_path, 'sample_subject_'+str(id)+'_tsss_mc.fif')
    raw = read_raw_fif(fname)
    events = mne.find_events(raw, min_duration=.002)
    events = events[events[:, 2] <= 92]
    subdata = np.zeros([92, 306, 1101], dtype=np.float32)
    for i in range(92):
        epochs = mne.Epochs(raw, events=events, event_id=i + 1,
baseline=None,
                            tmin=-0.1, tmax=1, preload=True)
        data = epochs.average().data
        subdata[i] = data
    megdata[subindex] = subdata
    subindex = subindex + 1

# the shape of MEG data: megdata is [3, 92, 306, 1101]
# n_subs = 3, n_conditions = 92, n_channels = 306, n_timepoints = 1101
(-100ms to 1000ms)


"""**********       Section 2: Preprocessing        **********"""

# shape of megdata: [n_subs, n_cons, n_chls, n_ts] -> [n_cons, n_subs,
n_chls, n_ts]
megdata = np.transpose(megdata, (1, 0, 2, 3))

# shape of megdata: [n_cons, n_subs, n_chls, n_ts] -> [n_cons, n_subs,
n_trials, n_chls, n_ts]
# here data is averaged, so set n_trials = 1
megdata = np.reshape(megdata, [92, 3, 1, 306, 1101])


"""****   Section 3: Calculating the neural pattern similarity   ****"""
```

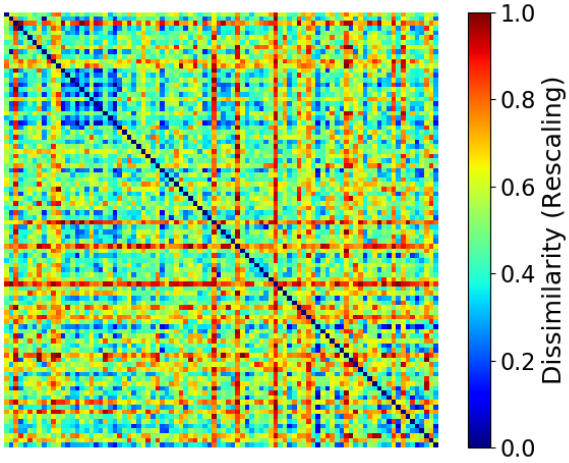
```
# Get data under different condition
# Here we calculate the neural pattern similarity (NPS) between two
stimulus
# Seeing Humanface vs. Seeing Non-Humanface

# get data under "humanface" condtion
megdata_humanface = megdata[12:24]
# get data under "nonhumanface" condition
megdata_nonhumanface = megdata[36:48]

# Average the data
avg_megdata_humanface = np.average(megdata_humanface, axis=0)
avg_megdata_nonhumanface = np.average(megdata_nonhumanface, axis=0)

# Create NPS input data
# Here we extract the data from first 5 channels between 0ms and 1000ms
nps_data = np.zeros([2, 3, 1, 5, 1000]) # n_cons=2, n_subs=3, n_chls=5,
n_ts=1000
nps_data[0] = avg_megdata_humanface[:, :, :5, 100:1100] # the start time
of the data is -100ms
nps_data[1] = avg_megdata_nonhumanface[:, :, :5, 100:1100] # so 100:1200
corresponds 0ms-1000ms

# Calculate the NPS with a 10ms time-window
# (raw sampling requency is 1000Hz, so here
time_win=10ms/(1s/1000Hz)/1000=10)
nps = nps(nps_data, time_win=10, time_step=10)

# Plot the NPS results
plot_nps_hotmap(nps, time_unit=[0, 0.01], abs=True)
```



```
# Smooth the results and plot
plot_nps_hotmap(nps, time_unit=[0, 0.01], abs=True, smooth=True)
```

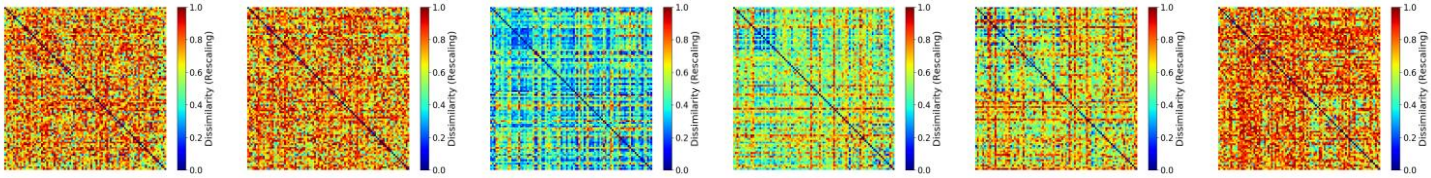



```
"""******    Section 4: Calculating single RDM and Plotting     *****"""

# Calculate the RDM based on the data during 190ms-210ms
rdm = eegRDM(megdata[:, :, :, :, 290:310])

# Plot this RDM
plot_rdm(rdm, rescale=True)
```




```
"""********    Section 5: Calculating RDMs and Plotting     ********"""

# Calculate the RDMs by a 10ms time-window
# (raw sampling requency is 1000Hz, so here
time_win=10ms/(1s/1000Hz)/1000=10)
rdms = eegRDM(megdata, time_opt=1, time_win=10, time_step=10)
```

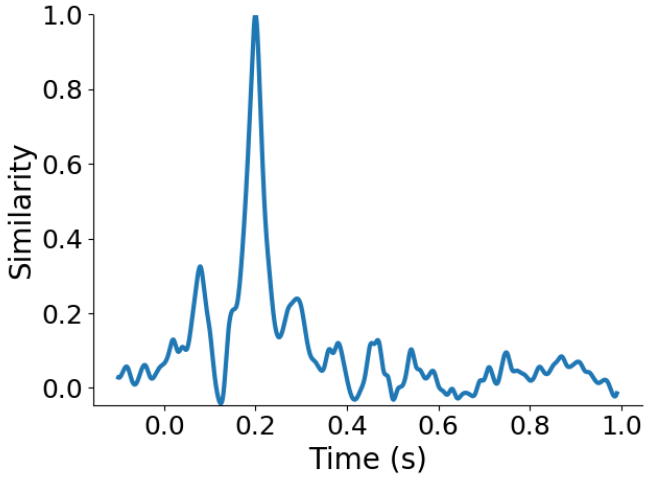
```
# Plot the RDM of -100ms, 0ms, 100ms, 200ms, 300ms, 400ms
times = [0, 10, 20, 30, 40, 50]
for t in times:
    plot_rdm(rdms[t], rescale=True)
```



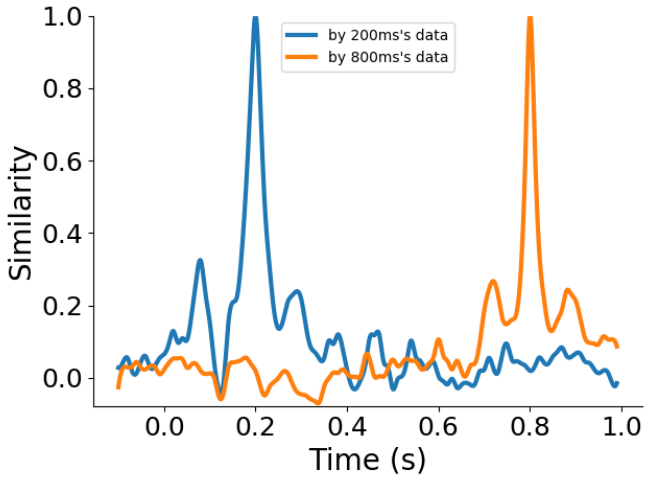

```
"""*****   Section 6: Calculating the Similarity between two RDMs  *****"""

# RDM of 200ms
rdm_sample1 = rdms[30]
# RDM of 800ms
rdm_sample2 = rdms[90]

# calculate the correlation coefficient between these two RDMs
corr = rdm_correlation_spearman(rdm_sample1, rdm_sample2, rescale=True)
print(corr)
```

```
SpearmanrResult(correlation=0.02665680483550596, pvalue=0.08462337954774739)
```

```
"""******   Section 7: Calculating the Similarity and Plotting    ******"""

# Calculate the representational similarity between 200ms and all the
time points
corrs1 = rdms_corr(rdm_sample1, rdms)

# Plot the corrs1
corrs1 = np.reshape(corrs1, [1, 110, 2])
plot_corrs_by_time(corrs1, time_unit=[-0.1, 0.01])
```

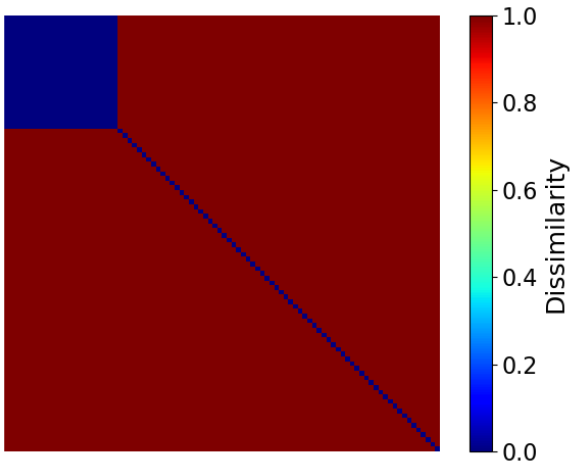



```
# Calculate and Plot multi-corrs
corrs2 = rdms_corr(rdm_sample2, rdms)
corrs = np.zeros([2, 110, 2])
corrs[0] = corrs1
corrs[1] = corrs2
labels = ["by 200ms's data", "by 800ms's data"]
plot_corrs_by_time(corrs, labels=labels, time_unit=[-0.1, 0.01])
```

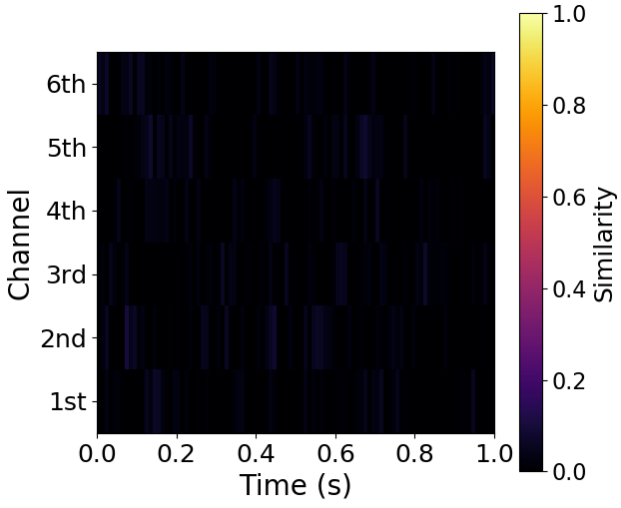



```
"""*****    Section 8: Calculating the RDMs for each channels    *****"""

# Calculate the RDMs for the first six channels by a 10ms time-window
between 0ms and 1000ms
rdms_chls = eegRDM(megdata[:, :, :, :6, 100:1100], chl_opt=1,
time_opt=1, time_win=10, time_step=10)

# Create a 'human-related' coding model RDM
```

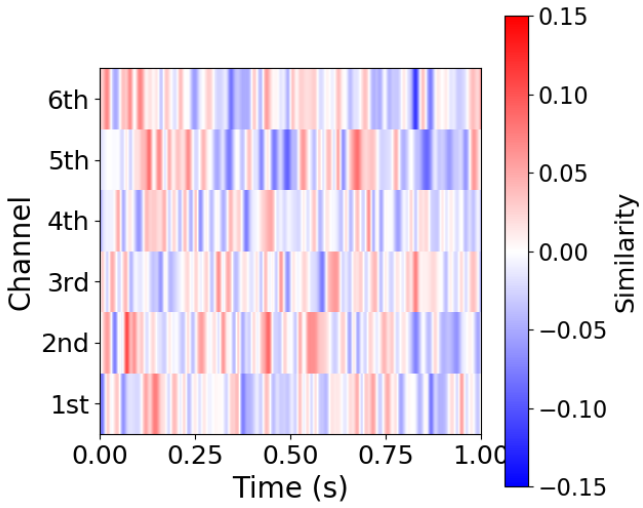
```
model_rdm = np.ones([92, 92])
for i in range(92):
    for j in range(92):
        if (i < 24) and (j < 24):
            model_rdm[i, j] = 0
    model_rdm[i, i] = 0

# Plot this coding model RDM
plot_rdm(model_rdm)
```




```
# Calculate the representational similarity between the neural
activities and the coding model for each channel
corrs_chls = rdms_corr(model_rdm, rdms_chls)

# Plot the representational similarity results
plot_corrs_hotmap(corrs_chls, time_unit=[0, 0.01])
```



```
# Set more parameters and re-plot
plot_corrs_hotmap(corrs_chls, time_unit=[0, 0.01], lim=[-0.15, 0.15],
smooth=True, cmap='bwr')
```



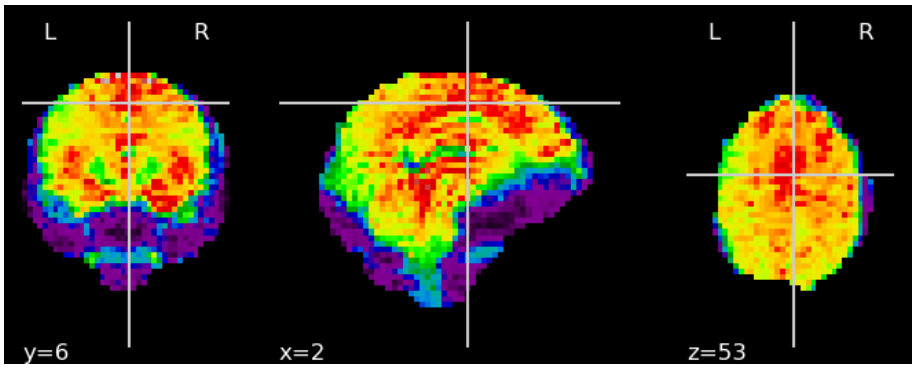
The fMRI Demo

Here 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.) Nilearn has been used to load this dataset and plot some results in this demo.

```
# -*- coding: utf-8 -*-

' a demo based on Haxby fMRI data '
# Users can learn how to use Neurora to do research based on fMRI data.

__author__ = 'Zitong Lu'

from nilearn import datasets, plotting
from nilearn.image import index_img, mean_img
import numpy as np
import pandas as pd
import nibabel as nib
from neurora.stuff import get_affine, datamask
from neurora.nps_cal import nps_fmri, nps_fmri_roi
from neurora.rsa_plot import plot_rdm
```

```
from neurora.rdm_cal import fmriRDM_roi, fmriRDM
from neurora.corr_cal_by_rdm import fmrirdms_corr
from neurora.corr_to_nii import corr_save_nii


"""**********       Section 1: Loading example data        **********"""
""" Here, we use Nilearn toolbox for loading data and processing """
""" you can learn this process from Nilearn
(http://nilearn.github.io/index.html) """

# load Haxby dataset (here, we only use subject2's data for this
example)
haxby_dataset = datasets.fetch_haxby()

# load the fMRI data filename & mask data filename
func_filename = haxby_dataset.func[0]
mask_filename = haxby_dataset.mask

# read label information of the experiment
labelinfo = pd.read_csv(haxby_dataset.session_target[0], sep=' ')
labels = labelinfo['labels']



"""**********          Section 2: Preprocessing           **********"""

# get mask data NumPy array
maskdata = nib.load(mask_filename).get_data()

# get the size of the data
nx, ny, nz = maskdata.shape

# labels of seven ategories
categories = ["face", "cat", "house", "chair", "shoe", "bottle",
"scissors"]
# numbe of conidtions: 7
ncon = len(categories)

# get fmri data under 7 conditions
# here we average the data under different conditions
fmri_data = np.full([ncon, nx, ny, nz], np.nan)

for i in range(ncon):
```

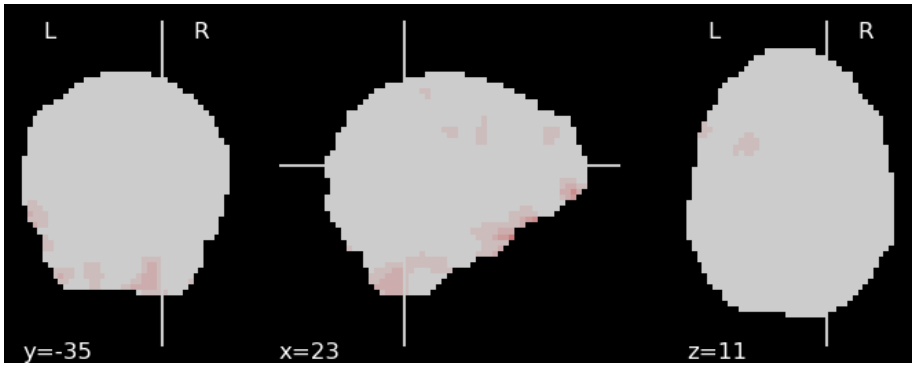
```
    img = mean_img(index_img(func_filename,
labels.isin([categories[i]])))
    fmri_data[i] = datamask(img.get_data(), maskdata)

# get fmri data under 'face'-condition
face_img = nib.Nifti1Image(fmri_data[0], affine=img.affine)
# have a look
plotting.plot_epi(face_img)
plotting.show()
```



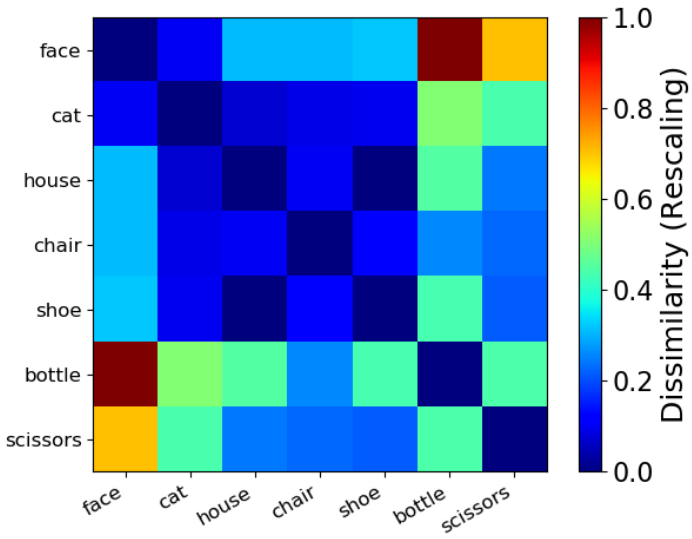

```
# reshaoe the data: [ncon, nx, ny, nz] -> [ncon, nsubs, nx, ny, nz]
# here just one subject's data
fmri_data = np.reshape(fmri_data, [ncon, 1, nx, ny, nz])



"""**Section 3: Calculating the neural pattern similarity (for ROI)**"""

# get mask of 'mask_face' in the dataset
mask_face_filename = haxby_dataset.mask_face[0]
mask_face_data = nib.load(mask_face_filename).get_data()

# get input data under two condition
# here, "face"-condition vs. "cat"-condition
nps_fmri_data = fmri_data[[0, 6]]

# calculate the neural pattern similarity (NPS) for ROI between two
stimulus
nps_roi = nps_fmri_roi(nps_fmri_data, mask_face_data)

# print the NPS result
print(nps_roi)
```

```
9.994484681410596982e-01, 5.880839542727496111e-43
```

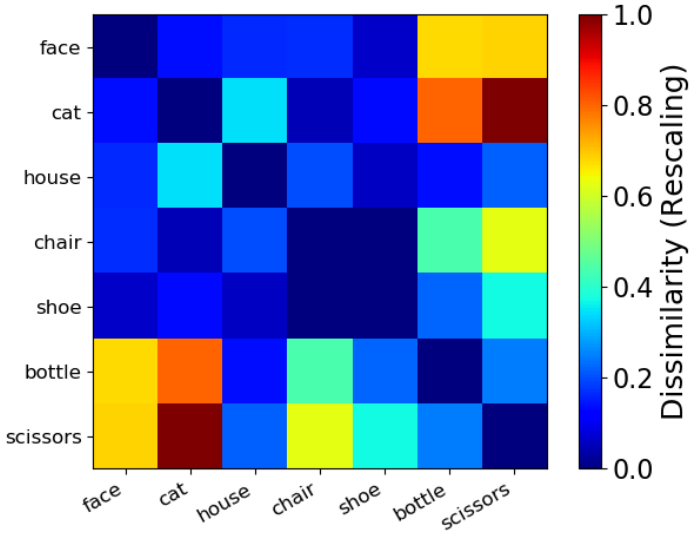
```
"""Section 4: Calculating the neural pattern similarity (Searchlight)"""

# calculate the neural pattern similarity (NPS) between two stimulus
nps = nps_fmri(nps_fmri_data)

# convert the NPS results into a .nii file
savefilename = "nps_img"
affine = get_affine(mask_filename)
corr_save_nii(nps, filename=savefilename, affine=affine, size=[nx, ny,
nz], plotrlt=True, plotrlt=False)

# have a look
plotting.plot_epi(savefilename+".nii")
plotting.show()
```




```
"""***    Section 5: Calculating the RDM for ROI and Plotting      ***"""

# get mask of "mask_vt" in the dataset
mask_vt_filename = haxby_dataset.mask_face[0]
mask_vt_data = nib.load(mask_vt_filename).get_data()

# calculate the RDM for ROI
rdm_roi = fmriRDM_roi(fmri_data, mask_vt_data)

# plot the RDM
plot_rdm(rdm_roi, rescale=True, conditions=categories)
```

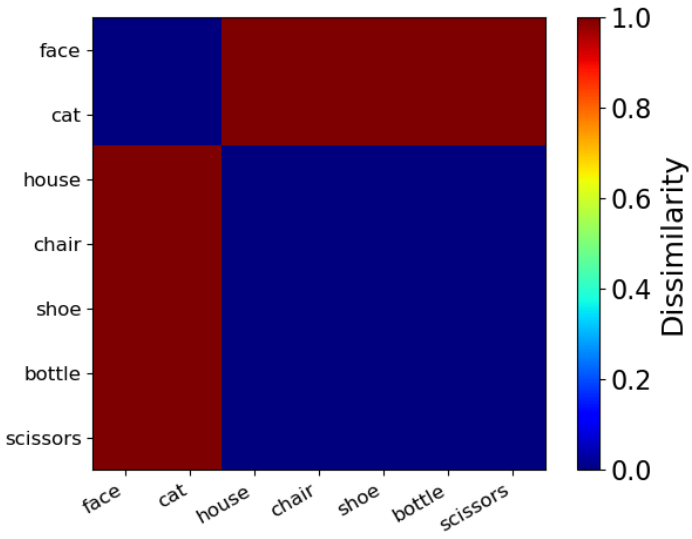



```
"""**  Section 6: Calculating the RDM by Searchlight and Plotting   **"""

# calculate the RDMs by Searchlight
fmri_RDMs = fmriRDM(fmri_data)

# plot one of the RDMs
plot_rdm(fmri_RDMs[20, 30, 30], rescale=True, conditions=categories)
```



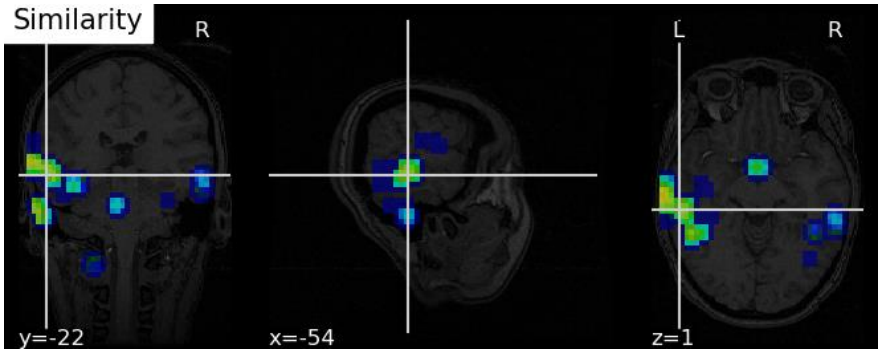

```
"""*** Section 7: Calculating the representational similarities    ***"""
"""***             between a coding model and neural activities    ***"""

# Create a RDM for "animate-inanimate" coding model
# which means the representations of animate matters are highly similar
# and the representations of inanimate matters are highly similar
```

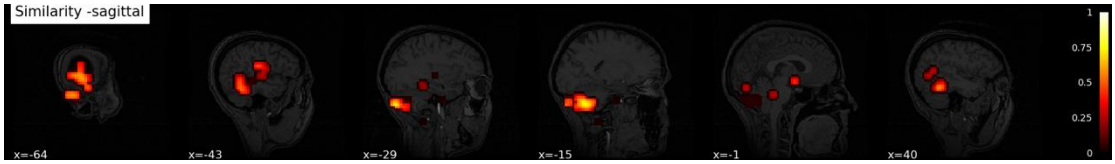
```
model_RDM = np.array([[0, 0, 1, 1, 1, 1, 1],
                      [0, 0, 1, 1, 1, 1, 1],
                      [1, 1, 0, 0, 0, 0, 0],
                      [1, 1, 0, 0, 0, 0, 0],
                      [1, 1, 0, 0, 0, 0, 0],
                      [1, 1, 0, 0, 0, 0, 0],
                      [1, 1, 0, 0, 0, 0, 0]])

# plot the model RDM
plot_rdm(model_RDM, conditions=categories)
```

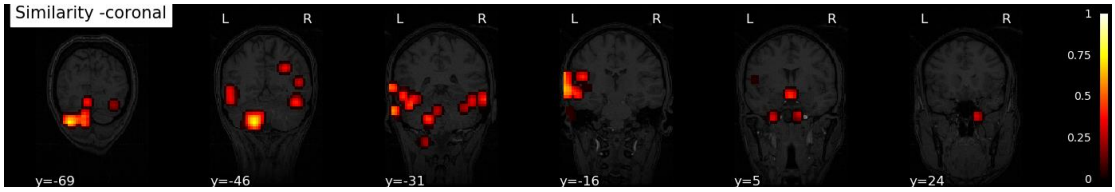


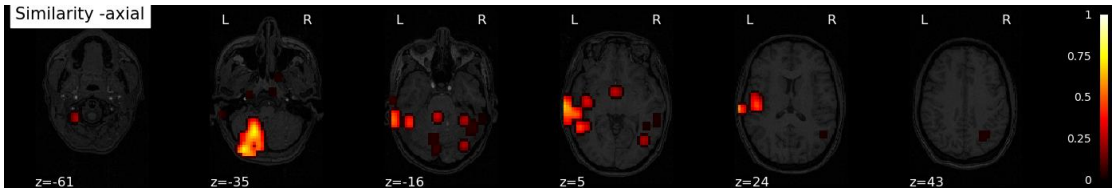

```
# calculate the similarities between model RDM and searchlight RDMs
corrs = fmrirdms_corr(model_RDM, fmri_RDMs)



"""*****      Section 8: Saving the RSA result and Plotting      *****"""

# load the filename of anatomical image as the background for plotting
ant_filename = haxby_dataset.anat[0]

# get the affine info
affine = get_affine(mask_filename)

# save the RSA result as a .nii file
# and visualize the result automatically
# p < 0.05, FDR-correct
rsarltfilename = "demo2_rsarlt_img"
```

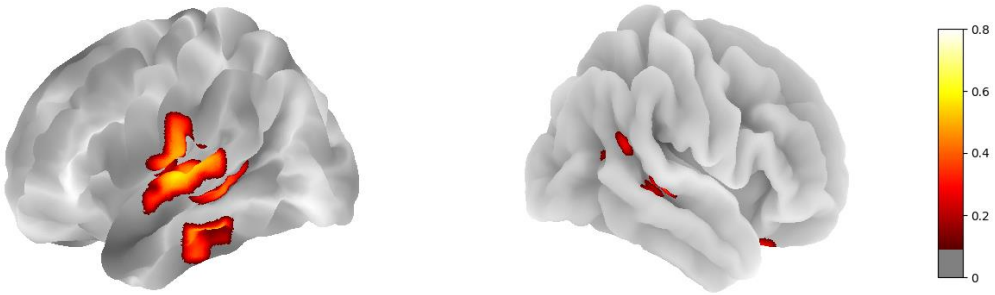
```
img = corr_save_nii(corrs, filename=rsarltfilename, affine=affine,
corr_mask=mask_filename, size=[40, 64, 64], p=0.001, plotrlt=True,
img_background=ant_filename)
```

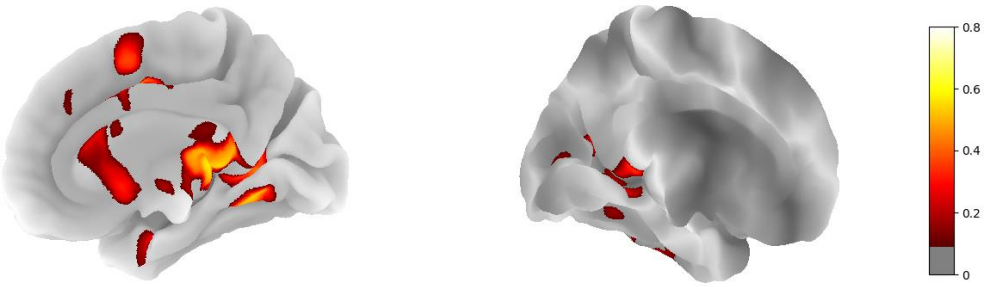




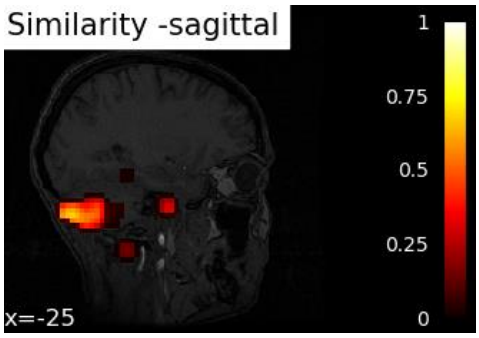









```
# Users can plot the RSA results independently by functions below
from neurora.rsa_plot import plot_brainrsa_regions
from neurora.rsa_plot import plot_brainrsa_montage
from neurora.rsa_plot import plot_brainrsa_glass
from neurora.rsa_plot import plot_brainrsa_surface

# here use a [5, 5, 5] cube to remove the significant area smaller than
it
# before filtering
plot_brainrsa_montage(rsarltfilename, slice=[[-25], 0, 0],
background=ant_filename)
```




```
# after filtering
plot_brainrsa_montage(rsarltfilename, threshold=125, slice=[[-25], 0,
0], background=ant_filename)
```