NeuroGraph

Release 2.1.0

Anwar Said

Feb 28, 2024

NEUROGRAPH:

1 Mental States 3

2 Cognitive Traits 5

3 Installation 7

4 Introduction by Example 9
4.1 Loading Benchmark datasets 9

5 Preprocessing Examples 11

6 Preprocessing Human Connectome Project (HCP1200) Dataset 13
6.1 Download and preprocess static datasets 13
6.2 Download and preprocess dynamic datasets 14

7 Load Benchmark Datasets 17

8 NeuroGraph Preprocessing Functionalities 19

9 NeuroGraph Utilities 23

10 Indices and tables 25

Python Module Index 27

Index 29

NeuroGraph is a collection of graph-based neuroimaging datasets that span multiple categories of demographics, mental states and cognitive traits. The following provides an overview of these categories and their associated datasets.

The data is made available in accordance with the WU-Minn HCP Consortium Open Access Data Use Terms (Step 4), which can be found at https://www.humanconnectome.org/study/hcp-young-adult/document/wu-minn-hcp-consortium-open-access-data-use-terms.

Demographics category includes gender and age estimation. The gender attribute facilitates a binary classification with the categories being male and female. Age is categorized into three distinct groups as in: 22-25, 26-30, and 31-35 years. We introduce four datasets named: HCP-Gender, HCP-Age, DynHCP-Gender, and DynHCP-Age under this category. The first two are static graph datasets while the last two are the corresponding dynamic graph datasets.

CHAPTER

ONE

MENTAL STATES

The mental state decoding involves seven tasks: Emotion Processing, Gambling, Language, Motor, Relational Processing, Social Cognition, and Working Memory. Each task is designed to help delineate a core set of functions relevant to different facets of the relation between human brain, cognition and behavior. Under this category, we present two datasets: HCP-Activity, a static representation, and DynHCP-Activity, its dynamic counterpart.

CHAPTER

TWO

COGNITIVE TRAITS

The cognitive traits category of our dataset comprises two significant traits: working memory (List Sorting) and fluid intelligence evaluation with PMAT24. Working memory refers to an individual's capacity to temporarily hold and manipulate information, a crucial aspect that influences higher cognitive functions such as reasoning, comprehension, and learning. Fluid intelligence represents the ability to solve novel problems, independent of any knowledge from the past. It demonstrates the capacity to analyze complex relationships, identify patterns, and derive solutions in dynamic situations. The prediction of both these traits, quantified as continuous variables in our dataset, are treated as regression problem. We aim to predict the performance or scores related to these cognitive traits based on the functional connectome graphs. We generate four datasets under cognitive traits: HCP Fluid Intelligence (HCP-FI), HCP Working Memory (HCP-WM), DynHCP-FI and DynHCP-WM.

CHAPTER

THREE

INSTALLATION

NeuroGraph is available for Python 3 and can be easily installed with pip

```
pip install NeuroGraph
```

NeuroGraph is developed on top of PyG and requires PyG to be installed. To install PyG, please follow the instructions provided in the PyG documentation here.

CHAPTER

FOUR

INTRODUCTION BY EXAMPLE

We will briefly introduce the fundamental concepts of NeuroGraph through self-contained examples. We closely follow the data representation format of PyG. Therefore, interested readers are referred to the PyG documentation for an introduction to the graph machine learning and PyG's data representation formats.

4.1 Loading Benchmark datasets

NeuroGraph provides two classes for loading static and dynamic benchmark datastes.

4.1.1 Loading Static Benchmarks

NeuroGraph utilizes the *PyG InMemoryDataset* class to facilitate the loading of datasets. this allows an easy-to-use interface for applying graph machine learning pipelines. For example, the *HCPGender* benchmark can be loaded as follows:

```
from NeuroGraph.datasets import NeuroGraphDataset
dataset = NeuroGraphDataset(root="data/", name= "HCPGender")
print(dataset.num_classes)
print(dataset.num_features)
```

4.1.2 Loading Dynamic Dataset

To efficiently store and utilize the dynamic datasets in *PyG* ` Batch format, we provide the corresponding functionality. Here is an example of loading the *DynHCPGender* dataset:

The dataset is a list of dynamic graphs represented in the *PyG* batch format, making it compatible with graph machine learning pipelines.

CHAPTER

FIVE

PREPROCESSING EXAMPLES

To bridge the gap betwee NeuroGraph and graph machine learning domains, NeuroGraph offers tools to easily preprocess and construct graph-based neuroimaging datasets. Here, we demonstrate how to preprocess your own data to construct functional connectomes and generate corresponding graphs-based representations.

The corresponding *Adjacency matrix* and *PyG* data objects can be created from the functional_connectome as follows.

```
from NeuroGraph import utils
adj = utils.construct_adj(fc, threshold= 5) # construct the adjacency matrix
data = utils.construct_data(fc, label= 1,threshold = 5) # construct PyG data object
```

We use correlation as node features while constructing data object from functional connectome.

The following is the source code for processing one fMRI scan with corresponding regressor using our preprocessing pipeline.

```
from NeuroGraph import utils
import numpy as np
from nilearn.image import load_img
img = load_img("data/raw/1.nii.gz") # 1.nii.gz is fMRI scan
regs = np.loadtxt("data/raw/1.txt") # 1.txt is the movement regressor
fmri = img.get_fdata()
fc = utils.preprocess(fmri, regs, n_rois= 100)
adj = utils.construct_adj(fc, threshold= 5) # construct the adjacency matrix
data = utils.construct_data(fc, label = 1,threshold = 5) # construct torch Data object
```

Our preprocessing pipeline consists of five steps and can also be applied seperately in steps.

```
from NeuroGraph import utils
import numpy as np
from nilearn.image import load_img

img = load_img("data/raw/1.nii.gz")
regs = np.loadtxt("data/raw/1.txt")
fmri = img.get_fdata()
parcells = utils.parcellation(fmri,n_rois = 100) ## this uses schaefer atlas by default
Y = utils.remove_drifts(parcells)
Y = utils.regress_head_motions(Y,regs)
fc = utils.construct_corr(Y)
adj = utils.construct_adj(fc, threshold= 5) # construct the adjacency matrix
data = utils.construct_data(fc, label = 1,threshold = 5)
```

CHAPTER

SIX

PREPROCESSING HUMAN CONNECTOME PROJECT (HCP1200) DATASET

NeuroGraph utilizes the HCP1200 dataset as a primary data source for exploring the dataset generation search space and constructing benchmarks. The HCP1200 dataset can be accessed from the HCP website by accepting the data usage terms. Additionally, the dataset is also available on an AWS S3 bucket, which can be accessed once authorization has been obtained from HCP. In this section, we provide various functions that allow you to crawl and preprocess the HCP datasets, enabling the construction of graph-based neuroimaging datasets. These functions streamline the process of obtaining and preparing the data for further analysis and modeling.

6.1 Download and preprocess static datasets

```
from NeuroGraph.preprocess import Brain_Connectome_Rest_Download
import boto3

root = "data/"
name = "HCPGender"
threshold = 5
path_to_data = "data/raw/HCPGender"  # store the raw downloaded scans
n_rois = 100
n_jobs = 5 # this script runs in parallel and requires the number of jobs is an input

ACCESS_KEY = ''  # your connectomeDB credentials
SECRET_KEY = ''
s3 = boto3.client('s3', aws_access_key_id=ACCESS_KEY, aws_secret_access_key=SECRET_KEY)
# this function requires both HCP_behavioral.csv and ids.pkl files under the root␣
↪directory. Both files have been provided and can be found under the data directory
rest_dataset = Brain_Connectome_Rest_Download(root,name,n_rois, threshold,path_to_data,n_
↪jobs,s3)
```

The provided function facilitates the download of data from the AWS S3 bucket, performs preprocessing steps, and generates a graph-based dataset. It is important to note that the *rest_dataset* used in this function consists of four labels: gender, age, working memory, and fluid intelligence. To create separate datasets based on these labels, the following functionalities can be used.

```
from NeuroGraph import preprocess

rest_dataset = preprocess.Brain_Connectome_Rest_Download(root,name,n_rois, threshold,
↪path_to_data,n_jobs,s3)
gender_dataset = preprocess.Gender_Dataset(root, "HCPGender",rest_dataset)
```

(continues on next page)

(continued from previous page)

```
age_dataset = preprocess.Age_Dataset(root, "HCPAge",rest_dataset)
wm_datast = preprocess.WM_Dataset(root, "HCPWM",rest_dataset)
fi_datast = preprocess.FI_Dataset(root, "HCPFI",rest_dataset)
```

To construct the State dataset, the following functionalities can be used.

```
from NeuroGraph import preprocess

state_dataset = preprocess.Brain_Connectome_State_Download(root, dataset_name,rois,␣
↪threshold,path_to_data,n_jobs,s3)
```

If you have the data locally, then the following functionalities can be used to preprocess the data.

```
from NeuroGraph import preprocess

rest_dataset = preprocess.Brain_Connectome_Rest(root, name, n_rois, threshold, path_to_
↪data, n_jobs)
```

Similarly, for constructing the State dataset, the following function can be used.

```
from NeuroGraph import preprocess

state_dataset = preprocess.Brain_Connectome_State(root, name, n_rois, threshold, path_to_
↪data, n_jobs)
```

6.2 Download and preprocess dynamic datasets

We also offer similar functionalities for constructing dynamic datasets. You can create a dynamic REST dataset from the data stored locally as follows.

```
from NeuroGraph import preprocess

ngd = Dyn_Prep(fmri, regs, n_rois=100, window_size=50, stride=3, dynamic_length=None)
dataset = ngd.dataset
labels = ngd.labels
print(len(dataset),len(labels))
```

Here the dataset is a list containing dynamic graphs in the form of PyG Batch, which can be easily fed into graph machine learning pipelines. The following examples demonstrate how a dynamic REST dataset can be downloaded and preprocessed on the fly.

```
from NeuroGraph import preprocess

dyn_obj = preporcess.Dyn_Down_Prep(root, name,s3,n_rois = 100, threshold = 10, window_
↪size = 50,stride == 3, dynamic_length=150)
dataset = dyn_obj.data_dict
```

Dyn_Down_Prep class downloads and preprocess the rest dataset and provides a dictionary that contains a list of dynamic graphs against each id. The dataset can be further prprocessed as follows to construct each benchmark.

```
from NeuroGraph import preprocess

dyn_obj = preporcess.Dyn_Down_Prep(root, name,s3,n_rois = 100, threshold = 10, window_
↪size = 50,stride == 3, dynamic_length=150)
dataset = dyn_obj.data_dict
gender_dataset, labels = [],[]
for k,v in dataset.items():
    if v is None:
        continue
    l = v[0].y
    gender = int(l[0].item())
    sub = []
    for d in v:
        new_data = Data(x = d.x, edge_index = d.edge_index, y = gender)
        sub.append(new_data)
    batch = Batch.from_data_list(sub)
    gender_dataset.append(batch)
    labels.append(gender)
print("gender dataset created with {} {} number of instances".format(len(gender_dataset),
↪ len(labels)))
new_dataset = {'labels':labels, "batches":gender_dataset}

age_dataset, labels = [],[]
for k,v in dataset.items():
    if v is None:
        continue
    l = v[0].y
    age = int(l[1].item())
    if age <=2:  ### Ignoring subjects with age >=36
        sub = []
        for d in v:
            new_data = Data(x = d.x, edge_index = d.edge_index, y = age)
            sub.append(new_data)
        batch = Batch.from_data_list(sub)
        age_dataset.append(batch)
        labels.append(gender)
print("Age dataset created with {} {} number of instances".format(len(age_dataset),␣
↪len(labels)))
new_dataset = {'labels':labels, "batches":age_dataset}

wm_dataset, labels = [],[]
for k,v in dataset.items():
    if v is None:
        continue
    l = v[0].y
    wm = int(l[2].item())
    if wm is not None: ## there are some None which should be removed
        sub = []
        for d in v:
    #         print(d)
            new_data = Data(x = d.x, edge_index = d.edge_index, y = wm)
            sub.append(new_data)
        batch = Batch.from_data_list(sub)
```

(continues on next page)

(continued from previous page)

```
        wm_dataset.append(batch)
        labels.append(gender)
print("Working memory dataset created with {} {} number of instances".format(len(wm_
↪dataset), len(labels)))
new_dataset = {'labels':labels, "batches":wm_dataset}

fi_dataset, labels = [],[]
for k,v in dataset.items():
    if v is None:
        continue
    l = v[0].y
    fi = int(l[3].item())
    if not math.isnan(fi): ## there are some None which should be removed
        sub = []
        for d in v:
    #         print(d)
            new_data = Data(x = d.x, edge_index = d.edge_index, y = fi)
            sub.append(new_data)
        batch = Batch.from_data_list(sub)
        fi_dataset.append(batch)
        labels.append(gender)
print("Fluid intelligence dataset created with {} {} number of instances".format(len(fi_
↪dataset), len(labels)))
new_dataset = {'labels':labels, "batches":fi_dataset}
```

CHAPTER

SEVEN

LOAD BENCHMARK DATASETS

class NeuroGraphDataset(*root: str*, *name: str*, *transform: Callable | None = None*, *pre_transform: Callable | None = None*, *pre_filter: Callable | None = None*)

Bases: `InMemoryDataset`

The NeuroGraph benchmark datasets from the "NeuroGraph: Benchmarks for Graph Machine Learning in Brain Connectomics" paper. *`NeuroGraphDataset`* holds a collection of five neuroimaging graph learning datasets that span multiple categories of demographics, mental states, and cognitive traits. See the documentation and the Github for more details.

Dataset	#Graphs	Task
`HCPTask`	7,443	Graph Classification
`HCPGender`	1,078	Graph Classification
`HCPAge`	1,065	Graph Classification
`HCPFI`	1,071	Graph Regression
`HCPWM`	1,078	Graph Regression

Args:

root (str): Root directory where the dataset should be saved. name (str): The name of the dataset (one of `"HCPGender"`,

`"HCPTask"`, `"HCPAge"`, `"HCPFI"`, `"HCPWM"`).

transform (callable, optional): A function/transform that takes in an
`torch_geometric.data.Data` object and returns a transformed version. The data object will be transformed before every access. (default: `None`)

pre_transform (callable, optional): A function/transform that takes in
an `torch_geometric.data.Data` object and returns a transformed version. The data object will be transformed before being saved to disk. (default: `None`)

pre_filter (callable, optional): A function that takes in an
`torch_geometric.data.Data` object and returns a boolean value, indicating whether the data object should be included in the final dataset. (default: `None`)

download()

Downloads the dataset to the `self.raw_dir` folder.

```
filenames = {'HCPAge':  'lzzks4472czy9f9vc8aikp7pdbknmtfe.zip', 'HCPFI':
'g2md9h9snh7jh6eeay02k1kr9m4ido9f.zip', 'HCPGender':
'r6hlz2arm7yiy6v6981cv2nzq3b0meax.zip', 'HCPTask':
'8wzz4y17wpxg2stip7iybtmymnybwvma.zip', 'HCPWM':
'xtmpa6712fidi94x6kevpsddf9skuoxy.zip'}
```

process()

Processes the dataset to the `self.processed_dir` folder.

property processed_dir: str

property processed_file_names: str

The name of the files in the `self.processed_dir` folder that must be present in order to skip processing.

property raw_dir: str

property raw_file_names: str

The name of the files in the `self.raw_dir` folder that must be present in order to skip downloading.

url = 'https://vanderbilt.box.com/shared/static'

class NeuroGraphDynamic(*root*, *name*)

Bases: `object`

Graph-based neuroimaging benchmark datasets, e.g., "DynHCPGender", "DynHCPAge", "DynHCPActivity", "DynHCPWM", or "DynHCPFI"

Args:

root (str): Root directory where the dataset should be saved. name (str): The name of the dataset.

Returns:

list: A list of graphs in PyTorch Geometric (pyg) format. Each graph contains a list of dynamic graphs batched in pyg batch.

download()

filenames = {'DynHCPActivity': '2so3fnfqakeu6hktz322o3nm2c8ocus7.zip', 'DynHCPAge': '195f9teg4t4apn6kl6hbc4ib4g9addtq.zip', 'DynHCPFI': 'un7w3ohb2mmyjqt1ou2wm3g87y1lfuuo.zip', 'DynHCPGender': 'mj0z6unea34lfz1hkdwsinj7g22yohxn.zip', 'DynHCPWM': 'mxy8fq3ghm60q6h7uhnu80pgvfxs6xo2.zip'}

load_data()

url = 'https://vanderbilt.box.com/shared/static'

CHAPTER

EIGHT

NEUROGRAPH PREPROCESSING FUNCTIONALITIES

class Age_Dataset(*root*, *dataset_name*, *dataset*, *transform=None*, *pre_transform=None*, *pre_filter=None*)

Bases: InMemoryDataset

process()

Processes the dataset to the self.processed_dir folder.

property processed_file_names

The name of the files in the self.processed_dir folder that must be present in order to skip processing.

class Brain_Connectome_Rest(*root*, *name*, *n_rois*, *threshold*, *path_to_data*, *n_jobs*, *transform=None*, *pre_transform=None*, *pre_filter=None*)

Bases: InMemoryDataset

construct_adj_postive_perc(*corr*)

construct adjacency matrix from the given correlation matrix and threshold

extract_from_3d_no(*volume*, *fmri*)

Extract time-series data from a 3d atlas with non-overlapping ROIs.

Inputs:

path_to_atlas = '/path/to/atlas.nii.gz' path_to_fMRI = '/path/to/fmri.nii.gz'

Output:

returns extracted time series # volumes x # ROIs

get_data_obj(*iid*, *behavioral_data*, *path_to_data*, *volume*)

process()

Processes the dataset to the self.processed_dir folder.

property processed_file_names

The name of the files in the self.processed_dir folder that must be present in order to skip processing.

class Brain_Connectome_Rest_Download(*root*, *name*, *n_rois*, *threshold*, *path_to_data*, *n_jobs*, *s3*, *transform=None*, *pre_transform=None*, *pre_filter=None*)

Bases: InMemoryDataset

construct_Adj_postive_perc(*corr*)

extract_from_3d_no(*volume*, *fmri*)

Extract time-series data from a 3d atlas with non-overlapping ROIs.

Inputs:

path_to_atlas = '/path/to/atlas.nii.gz' path_to_fMRI = '/path/to/fmri.nii.gz'

Output:
returns extracted time series # volumes x # ROIs

get_data_obj(*iid*, *behavioral_data*, *BUCKET_NAME*, *volume*)

process()

Processes the dataset to the `self.processed_dir` folder.

property processed_file_names

The name of the files in the `self.processed_dir` folder that must be present in order to skip processing.

class Brain_Connectome_Task(*root*, *dataset_name*, *n_rois*, *threshold*, *path_to_data*, *n_jobs*, *transform=None*, *pre_transform=None*, *pre_filter=None*)

Bases: `InMemoryDataset`

construct_adj_postive_perc(*corr*)

construct adjacency matrix from the given correlation matrix and threshold

extract_from_3d_no(*volume*, *fmri*)

Extract time-series data from a 3d atlas with non-overlapping ROIs.

Inputs:
path_to_atlas = '/path/to/atlas.nii.gz' path_to_fMRI = '/path/to/fmri.nii.gz'

Output:
returns extracted time series # volumes x # ROIs

get_data_obj_task(*iid*, *target_path*, *volume*)

process()

Processes the dataset to the `self.processed_dir` folder.

property processed_file_names

The name of the files in the `self.processed_dir` folder that must be present in order to skip processing.

class Brain_Connectome_Task_Download(*root*, *dataset_name*, *n_rois*, *threshold*, *path_to_data*, *n_jobs*, *s3*, *transform=None*, *pre_transform=None*, *pre_filter=None*)

Bases: `InMemoryDataset`

construct_Adj_postive_perc(*corr*)

extract_from_3d_no(*volume*, *fmri*)

Extract time-series data from a 3d atlas with non-overlapping ROIs.

Inputs:
path_to_atlas = '/path/to/atlas.nii.gz' path_to_fMRI = '/path/to/fmri.nii.gz'

Output:
returns extracted time series # volumes x # ROIs

get_data_obj_task(*iid*, *BUCKET_NAME*, *volume*)

process()

Processes the dataset to the `self.processed_dir` folder.

property processed_file_names

The name of the files in the `self.processed_dir` folder that must be present in order to skip processing.

class Dyn_Down_Prep(*root*, *name*, *s3*, *n_rois=100*, *threshold=10*, *window_size=50*, *stride=3*, *dynamic_length=150*)

Bases: object

construct_Adj_postive_perc(*corr*)

construct_dataset()

extract_from_3d_no(*fmri*)

Extract time-series data from a 3d atlas with non-overlapping ROIs.

Inputs:
path_to_atlas = '/path/to/atlas.nii.gz' path_to_fMRI = '/path/to/fmri.nii.gz'

Output:
returns extracted time series # volumes x # ROIs

get_dynamic_data_object(*iid*)

process_dynamic_fc(*timeseries*, *y*, *sampling_init=None*, *self_loop=True*)

Dyn_Prep(*fmri*, *regs*, *n_rois=100*, *window_size=50*, *stride=3*, *dynamic_length=None*)

Preprocess fMRI data using NeuroGraph preprocessing pipeline and construct dynamic functional connectome matrices

Args:

fmri (numpy array): fmri image regs (numpy array): regressor array rois (int): {100, 200, 300, 400, 500, 600, 700, 800, 900, 1000}, optional, Number of regions of interest. Default=100. window_size (int) : the length of the window, default = 50 stride (int): default: 3 dynamic_length (int) : length of the timeseries to be considered for dynamic graphs. For memory and compution efficiency, we set dynamic length = 50, default = None, if None, consider the whole timeseries object

class FI_Dataset(*root*, *dataset_name*, *dataset*, *transform=None*, *pre_transform=None*, *pre_filter=None*)

Bases: InMemoryDataset

process()

Processes the dataset to the self.processed_dir folder.

property processed_file_names

The name of the files in the self.processed_dir folder that must be present in order to skip processing.

class Gender_Dataset(*root*, *dataset_name*, *dataset*, *transform=None*, *pre_transform=None*, *pre_filter=None*)

Bases: InMemoryDataset

process()

Processes the dataset to the self.processed_dir folder.

property processed_file_names

The name of the files in the self.processed_dir folder that must be present in order to skip processing.

class WM_Dataset(*root*, *dataset_name*, *dataset*, *transform=None*, *pre_transform=None*, *pre_filter=None*)

Bases: InMemoryDataset

process()

Processes the dataset to the self.processed_dir folder.

property processed_file_names

The name of the files in the self.processed_dir folder that must be present in order to skip processing.

CHAPTER

NINE

NEUROGRAPH UTILITIES

construct_adj(*corr*, *threshold=5*)

create adjacency matrix from functional connectome matrix

Args:

corr (n x n numpy matrix): functional connectome matrix

Threshold (int (1- 100)): threshold for controling graph density.

the more higher the threshold, the more denser the graph. default: 5

construct_corr(*m*)

This function construct correlation matrix from the preprocessed fmri matrix Args.

m (numpy array): a preprocessed numpy matrix return: correlation matrix

construct_data(*corr*, *label*, *threshold=5*)

create pyg data object from functional connectome matrix. We use correlation as node features Args:

corr (n x n numpy matrix): functional connectome matrix

Threshold (int (1- 100)): threshold for controling graph density.

the more higher the threshold, the more denser the graph. default: 5

parcellation(*fmri*, *n_rois=1000*)

Prepfrom brain parcellation

Args:

fmri (numpy array): fmri image rois (int): {100, 200, 300, 400, 500, 600, 700, 800, 900, 1000}, optional, Number of regions of interest. Default=1000.

preprocess(*fmri*, *regs*, *n_rois=1000*)

Preprocess fMRI data using NeuroGraph preprocessing pipeline

Args:

fmri (numpy array): fmri image regs (numpy array): regressor array rois (int): {100, 200, 300, 400, 500, 600, 700, 800, 900, 1000}, optional, Number of regions of interest. Default=1000.

regress_head_motions(*Y*, *regs*)

This function regress out six rigid- body head motion parameters, along with their derivatives, from the fMRI data

Args: Y (numpy array)): fmri image regs (numpy array): movement regressor

remove_drifts(*Y*)

This function removes the scanner drifts in the fMRI signals that arise from instrumental factors. By eliminating these trends, we enhance the signal-to-noise ratio and increase the sensitivity to neural activity.

CHAPTER

TEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

n

NeuroGraph.datasets, 17
NeuroGraph.preprocess, 19
NeuroGraph.utils, 23

INDEX

A

Age_Dataset (*class in NeuroGraph.preprocess*), 19

B

Brain_Connectome_Rest (*class in NeuroGraph.preprocess*), 19
Brain_Connectome_Rest_Download (*class in NeuroGraph.preprocess*), 19
Brain_Connectome_Task (*class in NeuroGraph.preprocess*), 20
Brain_Connectome_Task_Download (*class in NeuroGraph.preprocess*), 20

C

construct_adj() (*in module NeuroGraph.utils*), 23
construct_adj_postive_perc() (*Brain_Connectome_Rest method*), 19
construct_Adj_postive_perc() (*Brain_Connectome_Rest_Download method*), 19
construct_adj_postive_perc() (*Brain_Connectome_Task method*), 20
construct_Adj_postive_perc() (*Brain_Connectome_Task_Download method*), 20
construct_Adj_postive_perc() (*Dyn_Down_Prep method*), 21
construct_corr() (*in module NeuroGraph.utils*), 23
construct_data() (*in module NeuroGraph.utils*), 23
construct_dataset() (*Dyn_Down_Prep method*), 21

D

download() (*NeuroGraphDataset method*), 17
download() (*NeuroGraphDynamic method*), 18
Dyn_Down_Prep (*class in NeuroGraph.preprocess*), 20
Dyn_Prep() (*in module NeuroGraph.preprocess*), 21

E

extract_from_3d_no() (*Brain_Connectome_Rest method*), 19
extract_from_3d_no() (*Brain_Connectome_Rest_Download method*), 19
extract_from_3d_no() (*Brain_Connectome_Task method*), 20
extract_from_3d_no() (*Brain_Connectome_Task_Download method*), 20
extract_from_3d_no() (*Dyn_Down_Prep method*), 21

F

FI_Dataset (*class in NeuroGraph.preprocess*), 21
filenames (*NeuroGraphDataset attribute*), 17
filenames (*NeuroGraphDynamic attribute*), 18

G

Gender_Dataset (*class in NeuroGraph.preprocess*), 21
get_data_obj() (*Brain_Connectome_Rest method*), 19
get_data_obj() (*Brain_Connectome_Rest_Download method*), 20
get_data_obj_task() (*Brain_Connectome_Task method*), 20
get_data_obj_task() (*Brain_Connectome_Task_Download method*), 20
get_dynamic_data_object() (*Dyn_Down_Prep method*), 21

L

load_data() (*NeuroGraphDynamic method*), 18

M

module
    NeuroGraph.datasets, 17
    NeuroGraph.preprocess, 19
    NeuroGraph.utils, 23

N

NeuroGraph.datasets
    module, 17
NeuroGraph.preprocess

module, 19
NeuroGraph.utils
module, 23
NeuroGraphDataset (*class in NeuroGraph.datasets*), 17
NeuroGraphDynamic (*class in NeuroGraph.datasets*), 18

P

parcellation() (*in module NeuroGraph.utils*), 23
preprocess() (*in module NeuroGraph.utils*), 23
process() (*Age_Dataset method*), 19
process() (*Brain_Connectome_Rest method*), 19
process() (*Brain_Connectome_Rest_Download method*), 20
process() (*Brain_Connectome_Task method*), 20
process() (*Brain_Connectome_Task_Download method*), 20
process() (*FI_Dataset method*), 21
process() (*Gender_Dataset method*), 21
process() (*NeuroGraphDataset method*), 18
process() (*WM_Dataset method*), 21
process_dynamic_fc() (*Dyn_Down_Prep method*), 21
processed_dir (*NeuroGraphDataset property*), 18
processed_file_names (*Age_Dataset property*), 19
processed_file_names (*Brain_Connectome_Rest property*), 19
processed_file_names (*Brain_Connectome_Rest_Download property*), 20
processed_file_names (*Brain_Connectome_Task property*), 20
processed_file_names (*Brain_Connectome_Task_Download property*), 20
processed_file_names (*FI_Dataset property*), 21
processed_file_names (*Gender_Dataset property*), 21
processed_file_names (*NeuroGraphDataset property*), 18
processed_file_names (*WM_Dataset property*), 21

R

raw_dir (*NeuroGraphDataset property*), 18
raw_file_names (*NeuroGraphDataset property*), 18
regress_head_motions() (*in module NeuroGraph.utils*), 23
remove_drifts() (*in module NeuroGraph.utils*), 23

U

url (*NeuroGraphDataset attribute*), 18
url (*NeuroGraphDynamic attribute*), 18

W

WM_Dataset (*class in NeuroGraph.preprocess*), 21