

Implement a new Neural Network Classifier

The BRAPH 2 Developers

September 19, 2023

This is the developer tutorial for implementing a new neural network classifier. In this Tutorial, we will explain how to create the generator file `*.gen.m` for a new neural network classifier, which can then be compiled by `braph2genesis`. All kinds of neural network models are (direct or indirect) extensions of the base element `NNBase`. Here, we will use as examples the neural network classifier `NNClassifierMLP` (multi-layer perceptron classifier).

Contents

<i>Implementation of a Neural Network Classifier</i>	2	
<i>Connectivity Data Point for Classification (NNDataPoint_CON_CLA)</i>		11
<i>Implementation of a Data Point with a Graph</i>	19	
<i>Graph Data Point for Regression (NNDataPoint_Graph_REG)</i>	19	
<i>Graph Data Point for Classification (NNDataPoint_Graph_CLA)</i>	26	
<i>Implementation of a Data Point with Graph Measures</i>	35	
<i>Graph Measure Data Point for Regression (NNDataPoint_Measure_REG)</i>		35
<i>Graph Measure Data Point for Classification (NNDataPoint_Measure_CLA)</i>		42

Implementation of a Neural Network Classifier

We will start by implementing in detail NNClassifierMLP, which is a direct extension of NNBase. A multi-layer perceptron classifier NNClassifierMLP comprises a multi-layer perceptron classifier model and a given dataset.

Code 1: NNClassifierMLP element header. The header section of the generator code for _NNClassifierMLP.gen.m provides the general information about the NNClassifierMLP element.

```

1 %% iheader!
2 NNClassifierMLP < NNBase (nn, multi-layer perceptron classifier) comprises a
    multi-layer perceptron classifier model and a given dataset. ①
3
4 %%% idescription!
5 A neural network multi-layer perceptron classifier (NNClassifierMLP)
    comprises a multi-layer perceptron classifier model and a given dataset
    .
6 NNClassifierMLP trains the multi-layer perceptron classifier with a
    formatted inputs ("CB", channel and batch) derived from the given
    dataset.
```

① defines NNClassifierMLP as a subclass of NNBase. The moniker will be nn

Code 2: NNClassifierMLP element prop update. The props_update section of the generator code for _NNClassifierMLP.gen.m updates the properties of the NNClassifierMLP element. This defines the core properties of the data point.

```

1 %% iprops_update!
2
3 %%% iprop!
4 NAME (constant, string) is the name of the neural network multi-layer
    perceptron classifier.
5 %%%% idefault!
6 'NNClassifierMLP'
7
8 %%% iprop!
9 DESCRIPTION (constant, string) is the description of the neural network
    multi-layer perceptron classifier.
10 %%%% idefault!
11 'A neural network multi-layer perceptron classifier (NNClassifierMLP)
    comprises a multi-layer perceptron classifier model and a given dataset
    . NNClassifierMLP trains the multi-layer perceptron classifier with a
    formatted inputs ("CB", channel and batch) derived from the given
    dataset.'
```

```

12
13 %%% iprop!
14 TEMPLATE (parameter, item) is the template of the neural network multi-layer
    perceptron classifier.
15 %%%% isettings!
16 'NNClassifierMLP'
17
18 %%% iprop!
19 ID (data, string) is a few-letter code for the neural network multi-layer
    perceptron classifier.
20 %%%% idefault!
21 'NNClassifierMLP ID'
```

```

22
23 %% iprop!
24 LABEL (metadata, string) is an extended label of the neural network multi-
    layer perceptron classifier.
25 %%%% idefault!
26 'NNClassifierMLP label'
27
28 %% iprop!
29 NOTES (metadata, string) are some specific notes about the neural network
    multi-layer perceptron classifier.
30 %%%% idefault!
31 'NNClassifierMLP notes'
32
33 %% iprop! ①
34 D (data, item) is the dataset to train the neural network model, and its
    data point class DP_CLASS defaults to one of the compatible classes
    within the set of DP_CLASSES.
35 %%%% isettings!
36 'NNDataset'
37 %%%% idefault!
38 NNDataset('DP_CLASS', 'NNDatapoint_CON_CLA')
39
40 %% iprop!
41 DP_CLASSES (parameter, classlist) is the list of compatible data points.
42 %%%% idefault! ②
43 {'NNDatapoint_CON_CLA' 'NNDatapoint_CON_FUN_MP_CLA' 'NNDatapoint_Graph_CLA'
    'NNDatapoint_Measure_CLA'}
44
45 %% iprop!
46 INPUTS (query, cell) constructs the data in the CB (channel-batch) format.
47 %%%% icalculate! ③
48 % inputs = nn.get('inputs', D) returns a cell array with the
49 % inputs for all data points in dataset D.
50 if isempty(varargin)
51     value = {};
52     return
53 end
54 d = varargin{1};
55 inputs_group = d.get('INPUTS');
56 if isempty(inputs_group)
57     value = {};
58 else
59     flattened_inputs_group = [];
60     for i = 1:length(inputs_group)
61         inputs_individual = inputs_group{i};
62         flattened_inputs_individual = [];
63         while ~isempty(inputs_individual)
64             currentData = inputs_individual{end}; % Get the last element
        from the stack
65             inputs_individual = inputs_individual(1:end-1); % Remove the
        last element
66
67             if iscell(currentData)
68                 % If it's a cell array, add its contents to the stack
69                 inputs_individual = [inputs_individual currentData{:}];
70             else
71                 % If it's numeric or other data, append it to the vector
72                 flattened_inputs_individual = [currentData(:);
        flattened_inputs_individual];
73             end

```

① defines NNDataset which contains the NNDatapoint to train this classifier

② defines the compatible NNDatapoint classes with this NNClassifierMLP.

③ is a query that transforms the input data of NNDatapoint to the CB (channel-batch) format by flattening its included cells.

```

74     end
75     flattened_inputs_group = [flattened_inputs_group;
76     flattened_inputs_individual'];
77     end
78     value = {flattened_inputs_group};
79 end
80 %%% iprop!
81 TARGETS (query, cell) constructs the targets in the CB (channel-batch)
82     format with one-hot vectors.
83 %%% icalculate! ④
84 % targets = nn.get('TARGETS', D) returns a cell array with the
85 % targets for all data points in dataset D with one-hot vectors.
86 if isempty(varargin)
87     value = {};
88 return
89 end
90 d = varargin{1};
91 target_ids = nn.get('TARGET_IDS', d);
92 value = onehotencode(categorical(target_ids), 2);
93
94 %%% iprop!
95 MODEL (result, net) is a trained neural network model.
96 %%% icalculate! ⑤
97 inputs = cell2mat(nn.get('INPUTS', nn.get('D'))); ⑥
98 targets = nn.get('TARGET_IDS', nn.get('D')); ⑦
99 if isempty(inputs) || isempty(targets)
100     value = network();
101 else
102     number_features = size(inputs, 2);
103     number_targets = size(targets, 2);
104     targets = categorical(targets);
105     number_classes = numel(categories(targets));
106
107     layers = nn.get('LAYERS'); ⑧
108     nn_architecture = [featureInputLayer(number_features, 'Name', 'Input')];
109     for i = 1:length(layers)
110         nn_architecture = [nn_architecture
111             fullyConnectedLayer(layers(i), 'Name', ['Dense_' num2str(i)])
112             batchNormalizationLayer('Name', ['BatchNormalization_' num2str(i)
113             ])]
114         dropoutLayer('Name', ['Dropout_' num2str(i)])
115     ];
116 end
117 nn_architecture = [nn_architecture
118     reluLayer('Name', 'Relu_output')
119     fullyConnectedLayer(number_classes, 'Name', 'Dense_output')
120     softmaxLayer
121     classificationLayer('Name', 'Output')
122 ];
123 % specify trianing options ⑨
124 options = trainingOptions(nn.get('SOLVER'), ...
125     'MiniBatchSize', nn.get('BATCH'), ...
126     'MaxEpochs', nn.get('EPOCHS'), ...
127     'Shuffle', nn.get('SHUFFLE'), ...
128     'Plots', nn.get('PLOT_TRAINING'), ...
129     'Verbose', nn.get('VERBOSE'));
130

```

④ is a query that construct the one-hot vectors for the target classes.

⑤ trains the classifier with the defined dataset.

⑥ and ⑦ firstly extract the inputs and targets with the corresponding format.

⑧ defines the neural network architecture with user specified number of neurons and number of layers.

⑨ defines the neural network training options.

```

131 % train the neural network (10)
132 value = trainNetwork(inputs, targets, nn_architecture, options);
133 end

```

(10) trains the model with those parameters and the neural network architecture.

Code 3: **NNClassifierMLP element props.** The props section of generator code for `_NNClassifierMLP.gen.m` defines the properties to be used in `NNClassifierMLP`.

```

1 %% iprops!
2
3 %%% iprop!
4 TARGET_IDS (query, stringlist) constructs the target IDs which represent the
   class of each data point.
5 %%% icalculate!
6 % targets = nn.get('TARGET_IDS', D) returns a cell array with the
7 % targets for all data points in dataset D.
8 if isempty(varargin)
9     value = {''};
10    return
11 end
12 d = varargin{1};
13 targets = d.get('TARGETS');
14 if isempty(targets)
15     value = {''};
16 else
17     nn_targets = [];
18     for i = 1:length(targets)
19         target = targets{i};
20         nn_targets = [nn_targets; target];
21     end
22     value = nn_targets;
23 end
24
25 %%% iprop!
26 LAYERS (data, rvector) defines the number of layers and their neurons.
27 %%% idefault!
28 [32 32]
29 %%% igui!
30 pr = PanelPropRVectorSmart('EL', nn, 'PROP', NNClassifierMLP.LAYERS, ...
31     'MIN', 0, 'MAX', 2000, ...
32     'DEFAULT', NNClassifierMLP.getPropDefault('LAYERS'), ...
33     varargin{:});
34
35 %%% iprop!
36 WAITBAR (gui, logical) determines whether to show the waitbar.
37 %%% idefault!
38 true
39
40 %%% iprop!
41 INTERRUPTIBLE (gui, scalar) sets whether the comparison computation is
   interruptible for multitasking.
42 %%% idefault!
43 .001
44
45 %%% iprop!
46 FEATURE_IMPORTANCE (query, cell) evaluates the average significance of each
   feature by iteratively shuffling its values P times and measuring the
   resulting average decrease in model performance.
47 %%% icalculate!

```

```

48 % fi = nn.get('FEATURE_IMPORTANCE', D, P, SEED) retrieves a cell array
    containing
49 % the feature importance values for the trained model, as assessed by
50 % evaluating it on the input dataset D.
51 if isempty(varargin)
52     value = {};
53     return
54 end
55 d = varargin{1};
56 P = varargin{2};
57 seeds = varargin{3};
58
59 inputs = cell2mat(nn.get('INPUTS', d));
60 if isempty(inputs)
61     value = {};
62     return
63 end
64 targets = nn.get('TARGETS', d);
65 net = nn.get('MODEL');
66
67 number_features = size(inputs, 2);
68 original_loss = crossentropy(net.predict(inputs), targets);
69
70 wb = braph2waitbar(nn.get('WAITBAR'), 0, ['Feature importance permutation
    ...']);
71
72 start = tic;
73 for i = 1:1:P
74     rng(seeds(i), 'twister')
75     parfor j = 1:1:number_features
76         scrambled_inputs = inputs;
77         permuted_value = squeeze(normrnd(mean(inputs(:, j)), std(inputs(:, j)
            )), squeeze(size(inputs(:, j)))) + squeeze(randn(size(inputs(:, j)))
            + mean(inputs(:, j)));
78         scrambled_inputs(:, j) = permuted_value;
79         scrambled_loss = crossentropy(net.predict(scrambled_inputs), targets
            );
80         feature_importance(j) = scrambled_loss;
81     end
82
83     feature_importance_all_permutations{i} = feature_importance /
        original_loss;
84
85     braph2waitbar(wb, i / P, ['Feature importance permutation ' num2str(i) '
        of ' num2str(P) ' - ' int2str(toc(start)) '.' int2str(mod(toc(start),
            1) * 10) 's ...'])
86     if nn.get('VERBOSE')
87         disp(['** PERMUTATION FEATURE IMPORTANCE - sampling #' int2str(i) '/'
            ' int2str(P) ' - ' int2str(toc(start)) '.' int2str(mod(toc(start), 1) *
            10) 's'])
88     end
89     if nn.get('INTERRUPTIBLE')
90         pause(nn.get('INTERRUPTIBLE'))
91     end
92 end
93
94 braph2waitbar(wb, 'close')
95
96 value = feature_importance_all_permutations;

```

Code 4: **NNDataPoint_CON_REG element tests**. The tests section from the element generator `_NNDataPoint_CON_REG.gen.m`. A test for creating example files should be prepared to test the properties of the data point. Furthermore, additional test should be prepared for validating the value of input and target for the data point.

```

1 %% itests!
2
3 %%% iexcluded_props! (1)
4 [NNDataPoint_CON_REG.SUB]
5
6 %%% itest!
7 %%% iname!
8 Create example files for regression (2)
9 %%% icode!
10 data_dir = [fileparts(which('NNDataPoint_CON_REG')) filesep 'Example data NN
    REG CON XLS'];
11 if ~isdir(data_dir)
12     mkdir(data_dir);
13
14 % Brain Atlas (3)
15 im_ba = ImporterBrainAtlasXLS('FILE', 'desikan.atlas.xlsx');
16 ba = im_ba.get('BA');
17 ex_ba = ExporterBrainAtlasXLS( ...
18     'BA', ba, ...
19     'FILE', [data_dir filesep() 'atlas.xlsx'] ...
20 );
21 ex_ba.get('SAVE')
22 N = ba.get('BR_DICT').get('LENGTH');
23
24 % saves RNG
25 rng_settings_ = rng(); rng('default')
26
27 sex_options = {'Female' 'Male'};
28
29 % Group (4)
30 K = 2; % degree (mean node degree is 2K)
31 beta = 0.3; % Rewiring probability
32 gr_name = 'CON_Group_XLS';
33 gr_dir = [data_dir filesep() gr_name];
34 mkdir(gr_dir);
35 vois = [
36     {'Subject ID'} {'Age'} {'Sex'}
37     {} {} cell2str(sex_options)}
38 ];
39 for i = 1:100 % subject number
40     sub_id = ['SubjectCON_' num2str(i)];
41     % create WS graphs with random beta
42     beta(i) = rand(1); (5)
43     h = WattsStrogatz(N, K, beta(i)); % create WS graph (6)
44
45     A = full(adjacency(h)); A(1:length(A)+1:numel(A)) = 0; % extract the
    adjacency matrix
46     r = 0 + (0.5 - 0) * rand(size(A)); diffA = A - r; A(A ~= 0) = diffA(
    A ~= 0); % make the adjacency matrix weighted
47     A = max(A, transpose(A)); % make the adjacency matrix symmetric
48
49     writetable(array2table(A), [gr_dir filesep() sub_id '.xlsx'], '

```

(1) List of properties that are excluded from testing.

(2) creates the example connectivity data files for regression analysis.

(3) creates and exports the brain atlas file to the example directory.

(4) creates one group of subjects with specified degree and rewiring probability configurations.

(5) generates random rewiring probability settings for each subject.

(6) and (10) utilize the provided degree and rewiring probability settings to generate corresponding Watts-Strogatz model graphs.

```

    WriteVariableNames', false) ⑦
50
51     % variables of interest
52     age_upperBound = 80;
53     age_lowerBound = 50;
54     age = age_lowerBound + beta(i)*(age_upperBound - age_lowerBound);
    ⑧
55     vois = [vois; {sub_id, age, sex_options(randi(2))}];
56 end
57 writetable(table(vois), [data_dir filesep() gr_name '.vois.xlsx'], '
    WriteVariableNames', false) ⑨
58
59 % reset RNG
60 rng(rng_settings_)
61 end
62 %%% itest_functions!
63 function h = WattsStrogatz(N, K, beta) ⑩
64 % H = WattsStrogatz(N,K,beta) returns a Watts-Strogatz model graph with N
65 % nodes, N*K edges, mean node degree 2*K, and rewiring probability beta.
66 %
67 % beta = 0 is a ring lattice, and beta = 1 is a random graph.
68
69 % Connect each node to its K next and previous neighbors. This constructs
70 % indices for a ring lattice.
71 s = repelem((1:N)', 1, K);
72 t = s + repmat(1:K, N, 1);
73 t = mod(t - 1, N) + 1;
74
75 % Rewire the target node of each edge with probability beta
76 for source = 1:N
77     switchEdge = rand(K, 1) < beta;
78
79     newTargets = rand(N, 1);
80     newTargets(source) = 0;
81     newTargets(s(t == source)) = 0;
82     newTargets(t(source, ~switchEdge)) = 0;
83
84     [~, ind] = sort(newTargets, 'descend');
85     t(source, switchEdge) = ind(1:nz(switchEdge));
86 end
87
88 h = graph(s,t);
89 end
90
91 %%% itest!
92 %%% iname! ⑪
93 Create a NNDataset containing NNDataPoint_CON_REG with simulated data
94 %%% icode!
95 % Load BrainAtlas
96 im_ba = ImporterBrainAtlasXLS( ...
97     'FILE', [fileparts(which('NNDataPoint_CON_REG')) filesep() 'Example data
    NN REG CON XLS' filesep() 'atlas.xlsx'], ...
98     'WAITBAR', true ...
99     );
100
101 ba = im_ba.get('BA');
102
103 % Load Group of SubjectCON
104 im_gr = ImporterGroupSubjectCON_XLS( ...

```

⑦ exports the adjacency matrix of the graph to an Excel file.

⑧ associates the age value with each individual rewiring probability setting.

⑨ exports the variables of interest to an Excel file.

⑪ validates the data point by using assertions to confirm that the input and target calculated values match the connectivity data and the variables of interest in the example files.


```

105     'DIRECTORY', [fileparts(which('NNDatapoint_CON_REG')) filesep 'Example
        data NN REG CON XLS' filesep 'CON_Group_XLS'], ...
106     'BA', ba, ...
107     'WAITBAR', true ...
108     );
109
110 gr = im_gr.get('GR');
111
112 % create an item list of NNDatapoint_CON_REG (12)
113 it_list = cellfun(@(x) NNDatapoint_CON_REG( ...
114     'ID', x.get('ID'), ...
115     'SUB', x, ...
116     'TARGET_IDS', x.get('VOI_DICT').get('KEYS')), ...
117     gr.get('SUB_DICT').get('IT_LIST'), ...
118     'UniformOutput', false);
119
120 % create a NNDatapoint_CON_REG DICT (13)
121 dp_list = IndexedDictionary(...
122     'IT_CLASS', 'NNDatapoint_CON_REG', ...
123     'IT_LIST', it_list ...
124     );
125
126 % create a NNDataset containing the NNDatapoint_CON_REG DICT (14)
127 d = NNDataset( ...
128     'DP_CLASS', 'NNDatapoint_CON_REG', ...
129     'DP_DICT', dp_list ...
130     );
131
132 % Check whether the number of inputs matches (14)
133 assert(length(d.get('INPUTS')) == gr.get('SUB_DICT').get('LENGTH'), ...
134     [BRAPH2.STR ':NNDatapoint_CON_REG:' BRAPH2.FAIL_TEST], ...
135     'NNDatapoint_CON_REG does not construct the dataset correctly. The
        number of the inputs should be the same as the number of imported
        subjects.' ...
136     )
137
138 % Check whether the number of targets matches (15)
139 assert(length(d.get('TARGETS')) == gr.get('SUB_DICT').get('LENGTH'), ...
140     [BRAPH2.STR ':NNDatapoint_CON_REG:' BRAPH2.FAIL_TEST], ...
141     'NNDatapoint_CON_REG does not construct the dataset correctly. The
        number of the targets should be the same as the number of imported
        subjects.' ...
142     )
143
144 % Check whether the content of input for a single datapoint matches (16)
145 for index = 1:1:gr.get('SUB_DICT').get('LENGTH')
146     individual_input = d.get('DP_DICT').get('IT', index).get('INPUT');
147     known_input = {gr.get('SUB_DICT').get('IT', index).get('CON')};
148
149     assert(isequal(individual_input, known_input), ...
150         [BRAPH2.STR ':NNDatapoint_CON_REG:' BRAPH2.FAIL_TEST], ...
151         'NNDatapoint_CON_REG does not construct the dataset correctly. The
            input value is not derived correctly.' ...
152         )
153 end
154
155 %%% itest!

```

(12), (13), and (14) creates an item list for the data points, subsequently generates the data point dictionary using the list, and then constructs the neural network dataset containing these data points.

(14) tests the number of inputs from the dataset matches the number of subjects in the group.

(15) tests the number of targets from the dataset matches the number of subjects in the group.

(16) tests the value of each input from the data point matches the subject's connectivity data.

```
156 %%%% iname! ①7
157 Example training-test regression
158 %%%% icode!
159 % ensure the example data is generated
160 if ~isfile([fileparts(which('NNDataPoint_CON_REG')) filesep 'Example data NN
    REG CON XLS' filesep 'atlas.xlsx'])
161     test_NNDataPoint_CON_REG % create example files
162 end
163
164 example_NN_CON_REG
```

①7 executes the corresponding example scripts to ensure the functionalities.

Connectivity Data Point for Classification (NNDataPoint_CON_CLA)

We can now use NNDataPoint_CON_REG as the basis to implement the NNDataPoint_CON_CLA. The parts of the code that are modified are highlighted.

Code 5: NNDataPoint_CON_CLA element header. The header section of the generator code for _NNDataPoint_CON_CLA.gen.m provides the general information about the NNDataPoint_CON_CLA element.

```

1 %% iheader!
2 NNDataPoint_CON_CLA < NNDataPoint (dp, connectivity classification data
    point) is a data point for classification with connectivity data.
3
4 %%% idescription!
5 A data point for classification with connectivity data (NNDataPoint_CON_CLA)
6 contains the input and target for neural network analysis with a subject
    with connectivity data (SubjectCON).
7 The input is the connectivity data of the subject.
8 The target is obtained from the variables of interest of the subject.

```

Code 6: NNDataPoint_CON_CLA element prop update. The props_update section of the generator code for _NNDataPoint_CON_CLA.gen.m updates the properties of the NNDataPoint_CON_CLA element. This defines the core properties of the data point.

```

1 %% iprops_update!
2
3 %%% iprop!
4 NAME (constant, string) is the name of a data point for classification with
    connectivity data.
5 %%% idefault!
6 'NNDataPoint_CON_CLA'
7
8 %%% iprop!
9 DESCRIPTION (constant, string) is the description of a data point for
    classification with connectivity data.
10 %%% idefault!
11 'A data point for classification with connectivity data (NNDataPoint_CON_CLA
    ) contains the input and target for neural network analysis with a
    subject with connectivity data (SubjectCON). The input is the
    connectivity data of the subject. The target is obtained from the
    variables of interest of the subject.'
12
13 %%% iprop!
14 TEMPLATE (parameter, item) is the template of a data point for
    classification with connectivity data.
15 %%% isettings!
16 'NNDataPoint_CON_CLA'
17
18 %%% iprop!
19 ID (data, string) is a few-letter code for a data point for classification
    with connectivity data.
20 %%% idefault!
21 'NNDataPoint_CON_CLA ID'
22
23 %%% iprop!

```

```

24 LABEL (metadata, string) is an extended label of a data point for
    classification with connectivity data.
25 %%%% idefault!
26 'NNDataPoint_CON_CLA label'
27
28 %%% iprop!
29 NOTES (metadata, string) are some specific notes about a data point for
    classification with connectivity data.
30 %%%% idefault!
31 'NNDataPoint_CON_CLA notes'
32
33 %%% iprop!
34 INPUT (result, cell) is the input value for this data point.
35 %%%% icalculate!
36 value = {dp.get('SUB').get('CON')};
37
38 %%% iprop!
39 TARGET (result, stringlist) is the target values for this data point.
40 %%%% icalculate!
41 value = dp.get('TARGET_IDS');

```

Code 7: NNDataPoint_CON_CLA element props. The props section of generator code for `_NNDataPoint_CON_CLA.gen.m` defines the properties to be used in `NNDataPoint_CON_CLA`.

```

1 %%% iprops!
2
3 %%% iprop!
4 SUB (data, item) is a subject with connectivity data.
5 %%%% isettings!
6 'SubjectCON'
7
8 %%% iprop! ①
9 TARGET_IDS (parameter, stringlist) is a list of variable-of-interest IDs to
    be used as the class targets.

```

① defines the target value using the data point's label in the form of a string list, e.g., 'Group1'.

Code 8: **NNDataPoint_CON_CLA element tests**. The tests section from the element generator `_NNDataPoint_CON_CLA.gen.m`. A test for creating example files should be prepared to test the properties of the data point. Furthermore, additional test should be prepared for validating the value of input and target for the data point.

```

1 %% itests!
2
3 %%% iexcluded_props!
4 [NNDataPoint_CON_CLA.SUB]
5
6 %%% itest!
7 %%% iname!
8 Create example files
9 %%% icode!
10 data_dir = [fileparts(which('NNDataPoint_CON_CLA')) filesep 'Example data NN
    CLA CON XLS'];
11 if ~isdir(data_dir)
12     mkdir(data_dir);
13
14 % Brain Atlas
15 im_ba = ImporterBrainAtlasXLS('FILE', 'desikan_atlas.xlsx');
16 ba = im_ba.get('BA');
17 ex_ba = ExporterBrainAtlasXLS( ...
18     'BA', ba, ...
19     'FILE', [data_dir filesep() 'atlas.xlsx'] ...
20 );
21 ex_ba.get('SAVE')
22 N = ba.get('BR_DICT').get('LENGTH');
23
24 % saves RNG
25 rng_settings_ = rng(); rng('default')
26
27 sex_options = {'Female' 'Male'};
28
29 % Group 1 ①
30 K1 = 2; % degree (mean node degree is 2K) - group 1
31 betal = 0.3; % Rewiring probability - group 1
32 gr1_name = 'CON_Group_1_XLS';
33 gr1_dir = [data_dir filesep() gr1_name];
34 mkdir(gr1_dir);
35 vois1 = [
36     {'Subject ID'} {'Age'} {'Sex'}
37     {} {} cell2str(sex_options)}
38 ];
39 for i = 1:1:50 % subject number
40     sub_id = ['SubjectCON_' num2str(i)];
41
42     h1 = WattsStrogatz(N, K1, betal); % create two WS graph
43     % figure(1) % Plot the two graphs to double-check
44     % plot(h1, 'NodeColor',[1 0 0], 'EdgeColor',[0 0 0], 'EdgeAlpha
    ',0.1, 'Layout','circle');
45     % title(['Group 1: Graph with $N = $ ' num2str(N_nodes) ...
46     %      ' nodes, $K = $ ' num2str(K1) ', and $beta = $ ' num2str(
    betal)], ...
47     %      'Interpreter','latex')
48     % axis equal
49
50     A1 = full(adjacency(h1)); A1(1:length(A1)+1:numel(A1)) = 0; %
    extract the adjacency matrix

```

① creates the first group of simulated data.

```

51     r = 0 + (0.5 - 0)*rand(size(A1)); diffA = A1 - r; A1(A1 ~= 0) =
diffA(A1 ~= 0); % make the adjacency matrix weighted
52     A1 = max(A1, transpose(A1)); % make the adjacency matrix symmetric
53
54     writetable(array2table(A1), [gr1_dir filesep() sub_id '.xlsx'], '
WriteVariableNames', false)
55
56     % variables of interest
57     vois1 = [vois1; {sub_id, randi(90), sex_options(randi(2))}];
58 end
59 writetable(table(vois1), [data_dir filesep() gr1_name '.vois.xlsx'], '
WriteVariableNames', false)
60
61 % Group 2 ②
62 K2 = 2; % degree (mean node degree is 2K) - group 2
63 beta2 = 0.85; % Rewiring probability - group 2 ③
64 gr2_name = 'CON_Group_2_XLS';
65 gr2_dir = [data_dir filesep() gr2_name];
66 mkdir(gr2_dir);
67 vois2 = [
68     {'Subject ID'} {'Age'} {'Sex'}
69     {} {} cell2str(sex_options)
70 ];
71 for i = 51:1:100
72     sub_id = ['SubjectCON_' num2str(i)];
73
74     h2 = WattsStrogatz(N, K2, beta2);
75     % figure(2)
76     % plot(h2, 'NodeColor',[1 0 0], 'EdgeColor',[0 0 0], 'EdgeAlpha
',0.1, 'Layout','circle');
77     % title(['Group 2: Graph with $N = $ ' num2str(N_nodes) ...
% ' nodes, $K = $ ' num2str(K2) ', and $\beta = $ ' num2str(
beta2)], ...
78     % 'Interpreter','latex')
79     % axis equal
80
81     A2 = full(adjacency(h2)); A2(1:length(A2)+1:numel(A2)) = 0;
82     r = 0 + (0.5 - 0)*rand(size(A2)); diffA = A2 - r; A2(A2 ~= 0) =
diffA(A2 ~= 0);
83     A2 = max(A2, transpose(A2));
84
85     writetable(array2table(A2), [gr2_dir filesep() sub_id '.xlsx'], '
WriteVariableNames', false)
86
87     % variables of interest
88     vois2 = [vois2; {sub_id, randi(90), sex_options(randi(2))}];
89 end
90 writetable(table(vois2), [data_dir filesep() gr2_name '.vois.xlsx'], '
WriteVariableNames', false)
91
92
93 % Group 3 ④
94 K3 = 2; % degree (mean node degree is 2K) - group 2
95 beta3 = 0.55; % Rewiring probability - group 2 ⑤
96 gr3_name = 'CON_Group_3_XLS';
97 gr3_dir = [data_dir filesep() gr3_name];
98 mkdir(gr3_dir);
99 vois3 = [
100     {'Subject ID'} {'Age'} {'Sex'}
101     {} {} cell2str(sex_options)
102 ];

```

② and ③ create the second group of simulated data with different rewiring probability parameter.

④ and ⑤ create the third group of simulated data with different rewiring probability parameter.

```

103     for i = 101:1:150
104         sub_id = ['SubjectCON_' num2str(i)];
105
106         h3 = WattsStrogatz(N, K3, beta3);
107         % figure(2)
108         % plot(h2, 'NodeColor',[1 0 0], 'EdgeColor',[0 0 0], 'EdgeAlpha
109         ',0.1, 'Layout','circle');
110         % title(['Group 2: Graph with $N = $ ' num2str(N_nodes) ...
111         %       ' nodes, $K = $ ' num2str(K2) ', and $\beta = $ ' num2str(
112         beta2)], ...
113         %       'Interpreter','latex')
114         % axis equal
115
116         A3 = full(adjacency(h3)); A3(1:length(A3)+1:numel(A3)) = 0;
117         r = 0 + (0.5 - 0)*rand(size(A3)); diffA = A3 - r; A3(A3 ~= 0) =
118         diffA(A3 ~= 0);
119         A3 = max(A3, transpose(A3));
120
121         writetable(array2table(A3), [gr3_dir filesep() sub_id '.xlsx'], '
122         WriteVariableNames', false)
123
124         % variables of interest
125         vois3 = [vois3; {sub_id, randi(90), sex_options(randi(2))}];
126     end
127     writetable(table(vois3), [data_dir filesep() gr3_name '.vois.xlsx'], '
128     WriteVariableNames', false)
129
130 % reset RNG
131 rng(rng_settings_)
132 end
133
134 %%% itest_functions!
135 function h = WattsStrogatz(N,K,beta)
136 % H = WattsStrogatz(N,K,beta) returns a Watts-Strogatz model graph with N
137 % nodes, N*K edges, mean node degree 2*K, and rewiring probability beta.
138 %
139 % beta = 0 is a ring lattice, and beta = 1 is a random graph.
140
141 % Connect each node to its K next and previous neighbors. This constructs
142 % indices for a ring lattice.
143 s = repelem((1:N)',1,K);
144 t = s + repmat(1:K,N,1);
145 t = mod(t-1,N)+1;
146
147 % Rewire the target node of each edge with probability beta
148 for source=1:N
149     switchEdge = rand(K, 1) < beta;
150
151     newTargets = rand(N, 1);
152     newTargets(source) = 0;
153     newTargets(s(t==source)) = 0;
154     newTargets(t(source, ~switchEdge)) = 0;
155
156     [~, ind] = sort(newTargets, 'descend');
157     t(source, switchEdge) = ind(1:nnz(switchEdge));
158 end
159
160 h = graph(s,t);
161 end
162
163 %%% itest!

```

```

159 %%% iname!
160 Create a NNDataset containing NNDataPoint_CON_CLA with simulated data
161 %%% icode!
162 % Load BrainAtlas
163 im_ba = ImporterBrainAtlasXLS( ...
164     'FILE', [fileparts(which('NNDataPoint_CON_CLA')) filesep 'Example data
        NN CLA CON XLS' filesep 'atlas.xlsx'], ...
165     'WAITBAR', true ...
166 );
167
168 ba = im_ba.get('BA');
169
170 % Load Groups of SubjectCON ⑥
171 im_gr1 = ImporterGroupSubjectCON_XLS( ...
172     'DIRECTORY', [fileparts(which('NNDataPoint_CON_CLA')) filesep 'Example
        data NN CLA CON XLS' filesep 'CON_Group_1_XLS'], ...
173     'BA', ba, ...
174     'WAITBAR', true ...
175 );
176
177 gr1 = im_gr1.get('GR');
178
179 im_gr2 = ImporterGroupSubjectCON_XLS( ...
180     'DIRECTORY', [fileparts(which('NNDataPoint_CON_CLA')) filesep 'Example
        data NN CLA CON XLS' filesep 'CON_Group_2_XLS'], ...
181     'BA', ba, ...
182     'WAITBAR', true ...
183 );
184
185 gr2 = im_gr2.get('GR');
186
187 % create item lists of NNDataPoint_CON_CLA ⑦
188 [~, group_folder_name] = fileparts(im_gr1.get('DIRECTORY'));
189 it_list1 = cellfun(@(x) NNDataPoint_CON_CLA( ...
190     'ID', x.get('ID'), ...
191     'SUB', x, ...
192     'TARGET_IDS', {group_folder_name}), ...
193     gr1.get('SUB_DICT').get('IT_LIST'), ...
194     'UniformOutput', false);
195
196 [~, group_folder_name] = fileparts(im_gr2.get('DIRECTORY'));
197 it_list2 = cellfun(@(x) NNDataPoint_CON_CLA( ...
198     'ID', x.get('ID'), ...
199     'SUB', x, ...
200     'TARGET_IDS', {group_folder_name}), ...
201     gr2.get('SUB_DICT').get('IT_LIST'), ...
202     'UniformOutput', false);
203
204 % create NNDataPoint_CON_CLA DICT items
205 dp_list1 = IndexedDictionary(...
206     'IT_CLASS', 'NNDataPoint_CON_CLA', ...
207     'IT_LIST', it_list1 ...
208 );
209
210 dp_list2 = IndexedDictionary(...
211     'IT_CLASS', 'NNDataPoint_CON_CLA', ...
212     'IT_LIST', it_list2 ...
213 );
214
215 % create a NNDataset containing the NNDataPoint_CON_CLA DICT
216 dl = NNDataset( ...

```

⑥ imports two groups of simulated data.

⑦ creates two datasets for the two groups.


```

217     'DP_CLASS', 'NNDataPoint_CON_CLA', ...
218     'DP_DICT', dp_list1 ...
219 );
220
221 d2 = NNDataset( ...
222     'DP_CLASS', 'NNDataPoint_CON_CLA', ...
223     'DP_DICT', dp_list2 ...
224 );
225
226 % Check whether the number of inputs matches ⑧
227 assert(length(d1.get('INPUTS')) == gr1.get('SUB_DICT').get('LENGTH'), ...
228     [BRAPH2.STR ':NNDataPoint_CON_CLA:' BRAPH2.FAIL_TEST], ...
229     'NNDataPoint_CON_CLA does not construct the dataset correctly. The
230     number of the inputs should be the same as the number of imported
231     subjects of group 1.' ...
232 );
233
234 assert(length(d2.get('INPUTS')) == gr2.get('SUB_DICT').get('LENGTH'), ...
235     [BRAPH2.STR ':NNDataPoint_CON_CLA:' BRAPH2.FAIL_TEST], ...
236     'NNDataPoint_CON_CLA does not construct the dataset correctly. The
237     number of the inputs should be the same as the number of imported
238     subjects of group 2.' ...
239 );
240
241 % Check whether the number of targets matches ⑨
242 assert(length(d1.get('TARGETS')) == gr1.get('SUB_DICT').get('LENGTH'), ...
243     [BRAPH2.STR ':NNDataPoint_CON_CLA:' BRAPH2.FAIL_TEST], ...
244     'NNDataPoint_CON_CLA does not construct the dataset correctly. The
245     number of the targets should be the same as the number of imported
246     subjects of group 1.' ...
247 );
248
249 assert(length(d2.get('TARGETS')) == gr2.get('SUB_DICT').get('LENGTH'), ...
250     [BRAPH2.STR ':NNDataPoint_CON_CLA:' BRAPH2.FAIL_TEST], ...
251     'NNDataPoint_CON_CLA does not construct the dataset correctly. The
252     number of the targets should be the same as the number of imported
253     subjects of group 2.' ...
254 );
255
256 % Check whether the content of input for a single datapoint matches ⑩
257 for index = 1:1:gr1.get('SUB_DICT').get('LENGTH')
258     individual_input = d1.get('DP_DICT').get('IT', index).get('INPUT');
259     known_input = {gr1.get('SUB_DICT').get('IT', index).get('CON')};
260
261     assert(isequal(individual_input, known_input), ...
262         [BRAPH2.STR ':NNDataPoint_CON_CLA:' BRAPH2.FAIL_TEST], ...
263         'NNDataPoint_CON_CLA does not construct the dataset correctly. The
264         input value is not derived correctly.' ...
265     );
266 end
267
268 for index = 1:1:gr2.get('SUB_DICT').get('LENGTH')
269     individual_input = d2.get('DP_DICT').get('IT', index).get('INPUT');
270     known_input = {gr2.get('SUB_DICT').get('IT', index).get('CON')};
271
272     assert(isequal(individual_input, known_input), ...
273         [BRAPH2.STR ':NNDataPoint_CON_CLA:' BRAPH2.FAIL_TEST], ...
274         'NNDataPoint_CON_CLA does not construct the dataset correctly. The
275         input value is not derived correctly.' ...
276     );
277 end

```

⑧ tests the number of inputs from the dataset matches the number of subjects in the group.

⑨ tests the number of targets from the dataset matches the number of subjects in the group.

⑩ tests the value of each input from the data point matches the subject's connectivity data.

```

267 end
268
269 %%% itest!
270 %%% iname!
271 Example training-test classification (11)
272 %%% icode!
273 % ensure the example data is generated
274 if ~isfile([fileparts(which('NNDataPoint_CON_CLA')) filesep 'Example data NN
      CLA CON XLS' filesep 'atlas.xlsx'])
275     test_NNDataPoint_CON_CLA % create example files
276 end
277
278 example_NN_CON_CLA

```

(11) executes the corresponding example scripts to ensure the functionalities.

Implementation of a Data Point with a Graph

Graph Data Point for Regression (NNDataPoint_Graph_REG)

Now we implement NNDataPoint_Graph_REG based on previous codes NNDataPoint_CON_REG. This neural network datapoint with graphs utilizes the adjacency matrix extracted from the derived graph of the subject. The modified parts of the code are highlighted.

Code 9: NNDataPoint_Graph_REG element header. The header section of the generator code for _NNDataPoint_Graph_REG.gen.m provides the general information about the NNDataPoint_Graph_REG element.

```

1 %% iheader!
2 NNDataPoint_Graph_REG < NNDataPoint (dp, measure regression) data point) is
  a data point for regression with a graph.
3
4 %%% idescription!
5 A data point for regression with a graph (NNDataPoint_Graph_REG)
6 contains both input and target for neural network analysis.
7 The input is the value of the adjacency matrix extracted from the derived
  graph of the subject.
8 The target is obtained from the variables of interest of the subject.

```

Code 10: NNDataPoint_Graph_REG element prop up-date. The props_update section of the generator code for _NNDataPoint_Graph_REG.gen.m updates the properties of the NNDataPoint_Graph_REG element. This defines the core properties of the data point.

```

1 %% iprops_update!
2
3 %%% iprop!
4 NAME (constant, string) is the name of a data point for regression with a
  graph.
5 %%% idefault!
6 'NNDataPoint_Graph_REG'
7
8 %%% iprop!
9 DESCRIPTION (constant, string) is the description of a data point for
  regression with a graph.
10 %%% idefault!
11 'A data point for regression with a graph (NNDataPoint_Graph_REG) contains
  both input and target for neural network analysis. The input is the
  value of the adjacency matrix extracted from the derived graph of the
  subject. The target is obtained from the variables of interest of the
  subject.'
12
13 %%% iprop!
14 TEMPLATE (parameter, item) is the template of a data point for regression
  with a graph.
15 %%% isettings!
16 'NNDataPoint_Graph_REG'
17
18 %%% iprop!

```

```

19 ID (data, string) is a few-letter code for a data point for regression with
    a graph.
20 %%%% idefault!
21 'NNDataPoint_Graph_REG ID'
22
23 %%%% iprop!
24 LABEL (metadata, string) is an extended label of a data point for regression
    with a graph.
25 %%%% idefault!
26 'NNDataPoint_Graph_REG label'
27
28 %%%% iprop!
29 NOTES (metadata, string) are some specific notes about a data point for
    regression with a graph.
30 %%%% idefault!
31 'NNDataPoint_Graph_REG notes'
32
33 %%%% iprop!
34 INPUT (result, cell) is the input value for this data point.
35 %%%% icalculate!
36 value = dp.get('G').get('A'); ①
37
38 %%%% iprop!
39 TARGET (result, cell) is the target value for this data point.
40 %%%% icalculate!
41 value = cellfun(@(x) dp.get('SUB').get('VOI-DICT').get('IT', x).get('V'), dp
    .get('TARGET_IDS'), 'UniformOutput', false);

```

① extracts the adjacency matrix from a Graph element as the input for this data point. Note that a Graph can be any kind of Graph, including GraphWU, MultigraphBUD, and MultiplexBUT, among others.

Code 11: NNDataPoint_Graph_REG element props. The props section of generator code for `_NNDataPoint_Graph_REG.gen.m` defines the properties to be used in `NNDataPoint_Graph_REG`.

```

1 %%% iprops!
2
3 %%% iprop! ①
4 G (data, item) is a graph.
5 %%%% isettings!
6 'Graph'
7
8 %%% iprop!
9 SUB (data, item) is a subject.
10 %%%% isettings!
11 'Subject'
12
13 %%% iprop!
14 TARGET_IDS (parameter, stringlist) is a list of variable-of-interest IDs to
    be used as the class targets.

```

① defines the Graph element which contains its corresponding adjacency matrix.

Code 12: **NNDataPoint_Graph_REG element tests**. The tests section from the element generator `_NNDataPoint_Graph_REG.gen.m`. A test for creating example files should be prepared to test the properties of the data point. Furthermore, additional test should be prepared for validating the value of input and target for the data point.

```

1 %% itests!
2
3 %%% iexcluded_props!
4 [NNDataPoint_Graph_REG.G NNDataPoint_Graph_REG.SUB]
5
6 %%% itest!
7 %%% iname! ①
8 Construct the data point with the adjacency matrix derived from its weighted
   undirected graph (GraphWU)
9 %%% icode!
10 % ensure the example data is generated
11 if ~isfile([fileparts(which('NNDataPoint_CON_REG')) filesep 'Example data NN
   REG CON XLS' filesep 'atlas.xlsx'])
12     test_NNDataPoint_CON_REG % create example files
13 end
14
15 % Load BrainAtlas
16 im_ba = ImporterBrainAtlasXLS( ...
17     'FILE', [fileparts(which('NNDataPoint_CON_REG')) filesep 'Example data
   NN REG CON XLS' filesep 'atlas.xlsx'], ...
18     'WAITBAR', true ...
19 );
20
21 ba = im_ba.get('BA');
22
23 % Load Groups of SubjectCON
24 im_gr = ImporterGroupSubjectCON_XLS( ...
25     'DIRECTORY', [fileparts(which('NNDataPoint_CON_REG')) filesep 'Example
   data NN REG CON XLS' filesep 'CON_Group_XLS'], ...
26     'BA', ba, ...
27     'WAITBAR', true ...
28 );
29
30 gr = im_gr.get('GR');
31
32 % Analysis CON WU ②
33 a_WU = AnalyzeEnsemble_CON_WU( ...
34     'GR', gr ...
35 );
36
37 a_WU.memorize('G_DICT'); ③
38
39 % create item lists of NNDataPoint_Graph_REG ④
40 it_list = cellfun(@(g, sub) NNDataPoint_Graph_REG( ...
41     'ID', sub.get('ID'), ...
42     'G', g, ...
43     'SUB', sub, ...
44     'TARGET_IDS', sub.get('VOI_DICT').get('KEYS')), ...
45     a_WU.get('G_DICT').get('IT_LIST'), gr.get('SUB_DICT').get('IT_LIST'), ...
46     'UniformOutput', false);
47
48 % create NNDataPoint_Graph_REG DICT items
49 dp_list = IndexedDictionary(...

```

① tests with the GraphWU element which contains weighted undirected adjacency matrix.

② and ③ create the AnalyzeEnsemble_CON_WU element and then memorize its graph dictionary G_DICT.

④ creates the NNDataPoint_Graph_REG element and use the Graph from G_DICT.

```

50     'IT_CLASS', 'NNDataPoint_Graph_REG', ...
51     'IT_LIST', it_list ...
52 );
53
54 % create a NNDataSet containing the NNDataPoint_Graph_REG DICT
55 d = NNDataSet( ...
56     'DP_CLASS', 'NNDataPoint_Graph_REG', ...
57     'DP_DICT', dp_list ...
58 );
59
60 % Check whether the content of input for a single datapoint matches ⑤
61 for index = 1:1:gr.get('SUB_DICT').get('LENGTH')
62     individual_input = d.get('DP_DICT').get('IT', index).get('INPUT');
63     known_input = a_WU.get('G_DICT').get('IT', index).get('A');
64
65     assert(isequal(individual_input, known_input), ...
66         [BRAPH2.STR 'NNDataPoint_Graph_REG:' BRAPH2.FAIL_TEST], ...
67         'NNDataPoint_Graph_REG does not construct the dataset correctly. The
        input value is not derived correctly.' ...
68     )
69 end
70
71 %%% itest!
72 %%% iname! ⑥
73 Construct the data point with the adjacency matrix derived from its binary
    undirected multigraph with fixed densities (MultigraphBUD)
74 %%% icode!
75 % ensure the example data is generated
76 if ~isfile([fileparts(which('NNDataPoint_CON_REG')) filesep 'Example data NN
    REG CON XLS' filesep 'atlas.xlsx'])
77     test_NNDataPoint_CON_REG % create example files
78 end
79
80 % Load BrainAtlas
81 im_ba = ImporterBrainAtlasXLS( ...
82     'FILE', [fileparts(which('NNDataPoint_CON_REG')) filesep 'Example data
    NN REG CON XLS' filesep 'atlas.xlsx'], ...
83     'WAITBAR', true ...
84 );
85
86 ba = im_ba.get('BA');
87
88 % Load Groups of SubjectCON
89 im_gr = ImporterGroupSubjectCON_XLS( ...
90     'DIRECTORY', [fileparts(which('NNDataPoint_CON_REG')) filesep 'Example
    data NN REG CON XLS' filesep 'CON_Group_XLS'], ...
91     'BA', ba, ...
92     'WAITBAR', true ...
93 );
94
95 gr = im_gr.get('GR');
96
97 % Analysis CON WU
98 densities = 0:25:100;
99
100 a_BUD = AnalyzeEnsemble_CON_BUD( ...
101     'DENSITIES', densities, ...
102     'GR', gr ...
103 );
104
105 a_BUD.memorize('G_DICT');

```

⑤ tests whether the value of each input from the data point matches the graph's adjacency matrix.

⑥ tests with the MultigraphBUD element which contains the adjacency matrix of binary undirected graph at fixed densities.

```

106
107 % create item lists of NNDataPoint_Graph_REG
108 it_list = cellfun(@(g, sub) NNDataPoint_Graph_REG( ...
109     'ID', sub.get('ID'), ...
110     'G', g, ...
111     'SUB', sub, ...
112     'TARGET_IDS', sub.get('VOI_DICT').get('KEYS')), ...
113     a_BUD.get('G_DICT').get('IT_LIST'), gr.get('SUB_DICT').get('IT_LIST')
114     , ...
115     'UniformOutput', false);
116
117 % create NNDataPoint_Graph_REG DICT items
118 dp_list = IndexedDictionary(...
119     'IT_CLASS', 'NNDataPoint_Graph_REG', ...
120     'IT_LIST', it_list ...
121 );
122
123 % create a NNDataset containing the NNDataPoint_Graph_REG DICT
124 d = NNDataset( ...
125     'DP_CLASS', 'NNDataPoint_Graph_REG', ...
126     'DP_DICT', dp_list ...
127 );
128
129 % Check whether the content of input for a single datapoint matches
130 for index = 1:1:gr.get('SUB_DICT').get('LENGTH')
131     individual_input = d.get('DP_DICT').get('IT', index).get('INPUT');
132     known_input = a_BUD.get('G_DICT').get('IT', index).get('A');
133
134     assert(isequal(individual_input, known_input), ...
135         [BRAPH2.STR ':NNDataPoint_Graph_REG: ' BRAPH2.FAIL_TEST], ...
136         'NNDataPoint_Graph_REG does not construct the dataset correctly. The
137         input value is not derived correctly.' ...
138         )
139 end
140
141 %%% itest!
142 %%% iname! ⑦
143 Construct the data point with the adjacency matrix derived from its
144     multiplex weighted undirected graph (MultiplexWU)
145 %%% icode!
146 % ensure the example data is generated
147 if ~isfile([fileparts(which('SubjectCON_FUN_MP')) filesep 'Example data
148     CON_FUN_MP_XLS' filesep 'atlas.xlsx'])
149     test_SubjectCON_FUN_MP % create example files
150 end
151
152 % Load BrainAtlas
153 im_ba = ImporterBrainAtlasXLS( ...
154     'FILE', [fileparts(which('SubjectCON_FUN_MP')) filesep 'Example data
155     CON_FUN_MP_XLS' filesep 'atlas.xlsx'], ...
156     'WAITBAR', true ...
157 );
158
159 ba = im_ba.get('BA');
160
161 % Load Groups of SubjectCON
162 im_gr = ImporterGroupSubjectCON_XLS( ...
163     'DIRECTORY', [fileparts(which('SubjectCON_FUN_MP')) filesep 'Example
164     data CON_FUN_MP_XLS' filesep 'CON_FUN_MP_Group_1_XLS.CON'], ...
165     'BA', ba, ...
166     'WAITBAR', true ...

```

⑦ tests with the MultiplexWU element which contains the adjacency matrix of weighted undirected multiplex.

```

161     );
162
163     gr_CON = im_gr.get('GR');
164
165     % Load Groups of SubjectFUN
166     im_gr = ImporterGroupSubjectFUN_XLS( ...
167         'DIRECTORY', [fileparts(which('SubjectCON_FUN_MP')) filesep 'Example
168             data CON_FUN_MP_XLS' filesep 'CON_FUN_MP_Group_1_XLS.FUN'], ...
169         'BA', ba, ...
170         'WAITBAR', true ...
171     );
172     gr_FUN = im_gr.get('GR');
173
174     % Combine Groups of SubjectCON with Groups of SubjectFUN
175     co_gr = CombineGroups_CON_FUN_MP( ...
176         'GR_CON', gr_CON, ...
177         'GR_FUN', gr_FUN, ...
178         'WAITBAR', true ...
179     );
180
181     gr = co_gr.get('GR_CON_FUN_MP');
182
183     % Analysis CON FUN MP WU
184     a_WU = AnalyzeEnsemble_CON_FUN_MP_WU( ...
185         'GR', gr ...
186     );
187
188     a_WU.memorize('G_DICT');
189
190     % create item lists of NNDataPoint_Graph_REG
191     it_list = cellfun(@(g, sub) NNDataPoint_Graph_REG( ...
192         'ID', sub.get('ID'), ...
193         'G', g, ...
194         'SUB', sub, ...
195         'TARGET_IDS', sub.get('VOI_DICT').get('KEYS')), ...
196         a_WU.get('G_DICT').get('IT_LIST'), gr.get('SUB_DICT').get('IT_LIST'), ...
197         'UniformOutput', false);
198
199     % create NNDataPoint_Graph_REG DICT items
200     dp_list = IndexedDictionary(...
201         'IT_CLASS', 'NNDataPoint_Graph_REG', ...
202         'IT_LIST', it_list ...
203     );
204
205     % create a NNDataset containing the NNDataPoint_Graph_REG DICT
206     d = NNDataset( ...
207         'DP_CLASS', 'NNDataPoint_Graph_REG', ...
208         'DP_DICT', dp_list ...
209     );
210
211     % Check whether the content of input for a single datapoint matches
212     for index = 1:1:gr.get('SUB_DICT').get('LENGTH')
213         individual_input = d.get('DP_DICT').get('IT', index).get('INPUT');
214         known_input = a_WU.get('G_DICT').get('IT', index).get('A');
215
216         assert(isequal(individual_input, known_input), ...
217             [BRAPH2.STR ':NNDataPoint_Graph_REG:' BRAPH2.FAIL_TEST], ...
218             'NNDataPoint_Graph_REG does not construct the dataset correctly. The
219             input value is not derived correctly.' ...
220         )

```



```

220 end
221
222 %%% itest!
223 %%% iname! ⑧
224 Example script for binary undirected graph (MultigraphBUT) using
      connectivity data
225 %%% icode!
226 if ~isfile([fileparts(which('NNDatapoint_CON_REG')) filesep 'Example data NN
      REG CON XLS' filesep 'atlas.xlsx'])
227     test_NNDatapoint_CON_REG % create example files
228 end
229 example_NNCV_CON_BUT_REG
230
231 %%% itest!
232 %%% iname! ⑨
233 Example script for binary undirected multiplex at fixed densities (
      MultiplexBUD) using connectivity data and functional data
234 %%% icode!
235 if ~isfile([fileparts(which('NNDatapoint_CON_FUN_MP_REG')) filesep 'Example
      data NN REG CON_FUN_MP XLS' filesep 'atlas.xlsx'])
236     test_NNDatapoint_CON_FUN_MP_REG % create example files
237 end
238 example_NNCV_CON_FUN_MP_BUD_REG
239
240 %%% itest!
241 %%% iname! ⑩
242 Example script for binary undirected multiplex at fixed thresholds (
      MultiplexBUT) using connectivity data and functional data
243 %%% icode!
244 if ~isfile([fileparts(which('NNDatapoint_CON_FUN_MP_REG')) filesep 'Example
      data NN REG CON_FUN_MP XLS' filesep 'atlas.xlsx'])
245     test_NNDatapoint_CON_FUN_MP_REG % create example files
246 end
247 example_NNCV_CON_FUN_MP_BUT_REG

```

⑧ tests with the MultigraphBUT element with the simulated connectivity data.

⑨ tests with the MultiplexBUD element with the simulated connectivity and functional data.

⑩ tests with the MultiplexBUT element with the simulated connectivity and functional data.

Graph Data Point for Classification (NNDataPoint_Graph_CLA)

Now we implement NNDataPoint_Graph_CLA based on previous codes NNDataPoint_CON_CLA. This neural network datapoint with graphs utilizes the adjacency matrix extracted from the derived graph of the subject. The modified parts of the code are highlighted.

Code 13: NNDataPoint_Graph_CLA element header. The header section of the generator code for _NNDataPoint_Graph_CLA.gen.m provides the general information about the NNDataPoint_Graph_CLA element.

```

1 %% iheader!
2 NNDataPoint_Graph_CLA < NNDataPoint (dp, graph classification data point) is
  a data point for classification with a graph.
3
4 %%% idescription!
5 A data point for classification with a graph (NNDataPoint_Graph_CLA)
6 contains both input and target for neural network analysis.
7 The input is the value of the adjacency matrix extracted from the derived
  graph of the subject.
8 The target is obtained from the variables of interest of the subject.

```

Code 14: NNDataPoint_Graph_CLA element prop update. The props_update section of the generator code for _NNDataPoint_Graph_CLA.gen.m updates the properties of the NNDataPoint_Graph_CLA element. This defines the core properties of the data point.

```

1 %% iprops_update!
2
3 %%% iprop!
4 NAME (constant, string) is the name of a data point for classification with
  a graph.
5 %%% idefault!
6 'NNDataPoint_Graph_CLA'
7
8 %%% iprop!
9 DESCRIPTION (constant, string) is the description of a data point for
  classification with a graph.
10 %%% idefault!
11 'A data point for classification with a graph (NNDataPoint_Graph_CLA)
  contains both input and target for neural network analysis. The input
  is the value of the adjacency matrix extracted from the derived graph
  of the subject. The target is obtained from the variables of interest
  of the subject.'
12
13 %%% iprop!
14 TEMPLATE (parameter, item) is the template of a data point for
  classification with a graph.
15 %%% isettings!
16 'NNDataPoint_Graph_CLA'
17
18 %%% iprop!
19 ID (data, string) is a few-letter code for a data point for classification
  with a graph.
20 %%% idefault!

```

```

21 'NNDataPoint_Graph_CLA ID'
22
23 %% iprop!
24 LABEL (metadata, string) is an extended label of a data point for
    classification with a graph.
25 %%%% idefault!
26 'NNDataPoint_Graph_CLA label'
27
28 %% iprop!
29 NOTES (metadata, string) are some specific notes about a data point for
    classification with a graph.
30 %%%% idefault!
31 'NNDataPoint_Graph_CLA notes'
32
33 %% iprop!
34 INPUT (result, cell) is the input value for this data point.
35 %%%% icalculate!
36 value = dp.get('G').get('A'); ①
37
38 %% iprop!
39 TARGET (result, cell) is the target value for this data point.
40 %%%% icalculate!
41 value = dp.get('TARGET_IDS');

```

① extracts the adjacency matrix from a Graph element as the input for this data point. Note that a Graph can be any kind of Graph, including GraphWU, MultigraphBUD, and MultiplexBUT, among others.

Code 15: NNDataPoint_Graph_CLA element props. The props section of generator code for `_NNDataPoint_Graph_CLA.gen.m` defines the properties to be used in `NNDataPoint_Graph_CLA`.

```

1 %% iprops!
2
3 %% iprop! ①
4 G (data, item) is a graph.
5 %%%% issettings!
6 'Graph'
7
8 %% iprop!
9 TARGET_IDS (parameter, stringlist) is a list of variable-of-interest IDs to
    be used as the class targets.

```

① defines the Graph element which contains its corresponding adjacency matrix.

Code 16: **NNDataPoint_Graph_CLA element tests**. The tests section from the element generator `_NNDataPoint_Graph_CLA.gen.m`. A test for creating example files should be prepared to test the properties of the data point. Furthermore, additional test should be prepared for validating the value of input and target for the data point.

```

1 %% itests!
2
3 %%% iexcluded_props!
4 [NNDataPoint_Graph_CLA.G]
5
6 %%% itest!
7 %%% iname! ①
8 Construct the data point with the adjacency matrix derived from its weighted
   undirected graph (GraphWU)
9 %%% icode!
10 % ensure the example data is generated
11 if ~isfile([fileparts(which('NNDataPoint_CON_CLA')) filesep 'Example data NN
   CLA CON XLS' filesep 'atlas.xlsx'])
12     test_NNDataPoint_CON_CLA % create example files
13 end
14
15 % Load BrainAtlas
16 im_ba = ImporterBrainAtlasXLS( ...
17     'FILE', [fileparts(which('NNDataPoint_CON_CLA')) filesep 'Example data
   NN CLA CON XLS' filesep 'atlas.xlsx'], ...
18     'WAITBAR', true ...
19 );
20
21 ba = im_ba.get('BA');
22
23 % Load Groups of SubjectCON
24 im_gr1 = ImporterGroupSubjectCON_XLS( ...
25     'DIRECTORY', [fileparts(which('NNDataPoint_CON_CLA')) filesep 'Example
   data NN CLA CON XLS' filesep 'CON_Group_1_XLS'], ...
26     'BA', ba, ...
27     'WAITBAR', true ...
28 );
29
30 gr1 = im_gr1.get('GR');
31
32 im_gr2 = ImporterGroupSubjectCON_XLS( ...
33     'DIRECTORY', [fileparts(which('NNDataPoint_CON_CLA')) filesep 'Example
   data NN CLA CON XLS' filesep 'CON_Group_2_XLS'], ...
34     'BA', ba, ...
35     'WAITBAR', true ...
36 );
37
38 gr2 = im_gr2.get('GR');
39
40 % Analysis CON WU
41 a_WU1 = AnalyzeEnsemble_CON_WU( ...
42     'GR', gr1 ...
43 );
44
45 a_WU2 = AnalyzeEnsemble_CON_WU( ...
46     'TEMPLATE', a_WU1, ...
47     'GR', gr2 ...
48 );
49

```

① tests with the GraphWU element which contains weighted undirected adjacency matrix.

```

50 a_WU1.memorize('G-DICT');
51 a_WU2.memorize('G-DICT');
52
53 % create item lists of NNDataPoint_Graph_CLA
54 [~, group_folder_name] = fileparts(im_gr1.get('DIRECTORY'));
55 it_list1 = cellfun(@(x) NNDataPoint_Graph_CLA( ...
56     'ID', x.get('ID'), ...
57     'G', x, ...
58     'TARGET_IDS', {group_folder_name}), ...
59     a_WU1.get('G-DICT').get('IT-LIST'), ...
60     'UniformOutput', false);
61
62 [~, group_folder_name] = fileparts(im_gr2.get('DIRECTORY'));
63 it_list2 = cellfun(@(x) NNDataPoint_Graph_CLA( ...
64     'ID', x.get('ID'), ...
65     'G', x, ...
66     'TARGET_IDS', {group_folder_name}), ...
67     a_WU2.get('G-DICT').get('IT-LIST'), ...
68     'UniformOutput', false);
69
70 % create NNDataPoint_Graph_CLA DICT items
71 dp_list1 = IndexedDictionary(...
72     'IT_CLASS', 'NNDataPoint_Graph_CLA', ...
73     'IT_LIST', it_list1 ...
74     );
75
76 dp_list2 = IndexedDictionary(...
77     'IT_CLASS', 'NNDataPoint_Graph_CLA', ...
78     'IT_LIST', it_list2 ...
79     );
80
81 % create a NNDataset containing the NNDataPoint_Graph_CLA DICT
82 d1 = NNDataset( ...
83     'DP_CLASS', 'NNDataPoint_Graph_CLA', ...
84     'DP-DICT', dp_list1 ...
85     );
86
87 d2 = NNDataset( ...
88     'DP_CLASS', 'NNDataPoint_Graph_CLA', ...
89     'DP-DICT', dp_list2 ...
90     );
91
92 % Check whether the content of input for a single datapoint matches
93 for index = 1:1:gr1.get('SUB-DICT').get('LENGTH')
94     individual_input = d1.get('DP-DICT').get('IT', index).get('INPUT');
95     known_input = a_WU1.get('G-DICT').get('IT', index).get('A');
96
97     assert(isequal(individual_input, known_input), ...
98         [BRAPH2.STR ':NNDataPoint_Graph_CLA:' BRAPH2.FAIL_TEST], ...
99         'NNDataPoint_Graph_CLA does not construct the dataset correctly. The
100         input value is not derived correctly.' ...
101         )
102 end
103
104 for index = 1:1:gr2.get('SUB-DICT').get('LENGTH')
105     individual_input = d2.get('DP-DICT').get('IT', index).get('INPUT');
106     known_input = a_WU2.get('G-DICT').get('IT', index).get('A');
107
108     assert(isequal(individual_input, known_input), ...
109         [BRAPH2.STR ':NNDataPoint_Graph_CLA:' BRAPH2.FAIL_TEST], ...
110         'NNDataPoint_Graph_CLA does not construct the dataset correctly. The

```

```

        input value is not derived correctly.' ...
    )
110 end
111 end
112
113 %% itest!
114 %%% iname! ②
115 Construct the data point with the adjacency matrix derived from its binary
    undirected multigraph with fixed densities (MultigraphBUD)
116 %%% icode!
117 % ensure the example data is generated
118 if ~isfile([fileparts(which('NNDataPoint_CON_CLA')) filesep 'Example data NN
    CLA CON XLS' filesep 'atlas.xlsx'])
119     test_NNDataPoint_CON_CLA % create example files
120 end
121
122 % Load BrainAtlas
123 im_ba = ImporterBrainAtlasXLS( ...
124     'FILE', [fileparts(which('NNDataPoint_CON_CLA')) filesep 'Example data
    NN CLA CON XLS' filesep 'atlas.xlsx'], ...
125     'WAITBAR', true ...
126 );
127
128 ba = im_ba.get('BA');
129
130 % Load Groups of SubjectCON
131 im_gr1 = ImporterGroupSubjectCON_XLS( ...
132     'DIRECTORY', [fileparts(which('NNDataPoint_CON_CLA')) filesep 'Example
    data NN CLA CON XLS' filesep 'CON_Group_1_XLS'], ...
133     'BA', ba, ...
134     'WAITBAR', true ...
135 );
136
137 gr1 = im_gr1.get('GR');
138
139 im_gr2 = ImporterGroupSubjectCON_XLS( ...
140     'DIRECTORY', [fileparts(which('NNDataPoint_CON_CLA')) filesep 'Example
    data NN CLA CON XLS' filesep 'CON_Group_2_XLS'], ...
141     'BA', ba, ...
142     'WAITBAR', true ...
143 );
144
145 gr2 = im_gr2.get('GR');
146
147 % Analysis CON WU
148 densities = 0:25:100;
149
150 a_BUD1 = AnalyzeEnsemble_CON_BUD( ...
151     'DENSITIES', densities, ...
152     'GR', gr1 ...
153 );
154
155 a_BUD2 = AnalyzeEnsemble_CON_BUD( ...
156     'TEMPLATE', a_BUD1, ...
157     'GR', gr2 ...
158 );
159
160 a_BUD1.memorize('G-DICT');
161 a_BUD2.memorize('G-DICT');
162
163 % create item lists of NNDataPoint_Graph_CLA
164 [~, group_folder_name] = fileparts(im_gr1.get('DIRECTORY'));

```

② tests with the MultigraphBUD element which contains binary undirected adjacency matrix at fixed densities.

```

165 it_list1 = cellfun(@(x) NNDataPoint_Graph_CLA( ...
166     'ID', x.get('ID'), ...
167     'G', x, ...
168     'TARGET_IDS', {group_folder_name}), ...
169     a_BUD1.get('G_DICT').get('IT_LIST'), ...
170     'UniformOutput', false);
171
172 [~, group_folder_name] = fileparts(im_gr2.get('DIRECTORY'));
173 it_list2 = cellfun(@(x) NNDataPoint_Graph_CLA( ...
174     'ID', x.get('ID'), ...
175     'G', x, ...
176     'TARGET_IDS', {group_folder_name}), ...
177     a_BUD2.get('G_DICT').get('IT_LIST'), ...
178     'UniformOutput', false);
179
180 % create NNDataPoint_Graph_CLA DICT items
181 dp_list1 = IndexedDictionary(...
182     'IT_CLASS', 'NNDataPoint_Graph_CLA', ...
183     'IT_LIST', it_list1 ...
184     );
185
186 dp_list2 = IndexedDictionary(...
187     'IT_CLASS', 'NNDataPoint_Graph_CLA', ...
188     'IT_LIST', it_list2 ...
189     );
190
191 % create a NNDataset containing the NNDataPoint_Graph_CLA DICT
192 d1 = NNDataset( ...
193     'DP_CLASS', 'NNDataPoint_Graph_CLA', ...
194     'DP_DICT', dp_list1 ...
195     );
196
197 d2 = NNDataset( ...
198     'DP_CLASS', 'NNDataPoint_Graph_CLA', ...
199     'DP_DICT', dp_list2 ...
200     );
201
202 % Check whether the content of input for a single datapoint matches
203 for index = 1:1:gr1.get('SUB_DICT').get('LENGTH')
204     individual_input = d1.get('DP_DICT').get('IT', index).get('INPUT');
205     known_input = a_BUD1.get('G_DICT').get('IT', index).get('A');
206
207     assert(isequal(individual_input, known_input), ...
208         [BRAPH2.STR ':NNDataPoint_Graph_CLA:' BRAPH2.FAIL_TEST], ...
209         'NNDataPoint_Graph_CLA does not construct the dataset correctly. The
210         input value is not derived correctly.' ...
211     )
212 end
213
214 for index = 1:1:gr2.get('SUB_DICT').get('LENGTH')
215     individual_input = d2.get('DP_DICT').get('IT', index).get('INPUT');
216     known_input = a_BUD2.get('G_DICT').get('IT', index).get('A');
217
218     assert(isequal(individual_input, known_input), ...
219         [BRAPH2.STR ':NNDataPoint_Graph_CLA:' BRAPH2.FAIL_TEST], ...
220         'NNDataPoint_Graph_CLA does not construct the dataset correctly. The
221         input value is not derived correctly.' ...
222     )
223 end
224
225 %%% itest!

```

```

224 %%% iname! ③
225 Construct the data point with the adjacency matrix derived from its
      multiplex weighted undirected graph (MultiplexWU)
226 %%% icode!
227 % ensure the example data is generated
228 if ~isfile([fileparts(which('SubjectCON_FUN_MP')) filesep 'Example data
      CON_FUN_MP_XLS' filesep 'atlas.xlsx'])
229     test_SubjectCON_FUN_MP % create example files
230 end
231
232 % Load BrainAtlas
233 im_ba = ImporterBrainAtlasXLS( ...
234     'FILE', [fileparts(which('SubjectCON_FUN_MP')) filesep 'Example data
      CON_FUN_MP_XLS' filesep 'atlas.xlsx'], ...
235     'WAITBAR', true ...
236 );
237
238 ba = im_ba.get('BA');
239
240 % Load Groups of SubjectCON
241 im_gr1 = ImporterGroupSubjectCON_XLS( ...
242     'DIRECTORY', [fileparts(which('SubjectCON_FUN_MP')) filesep 'Example
      data CON_FUN_MP_XLS' filesep 'CON_FUN_MP_Group_1_XLS.CON'], ...
243     'BA', ba, ...
244     'WAITBAR', true ...
245 );
246
247 gr1_CON = im_gr1.get('GR');
248
249 im_gr2 = ImporterGroupSubjectCON_XLS( ...
250     'DIRECTORY', [fileparts(which('SubjectCON_FUN_MP')) filesep 'Example
      data CON_FUN_MP_XLS' filesep 'CON_FUN_MP_Group_2_XLS.CON'], ...
251     'BA', ba, ...
252     'WAITBAR', true ...
253 );
254
255 gr2_CON = im_gr2.get('GR');
256
257 % Load Groups of SubjectFUN
258 im_gr1 = ImporterGroupSubjectFUN_XLS( ...
259     'DIRECTORY', [fileparts(which('SubjectCON_FUN_MP')) filesep 'Example
      data CON_FUN_MP_XLS' filesep 'CON_FUN_MP_Group_1_XLS.FUN'], ...
260     'BA', ba, ...
261     'WAITBAR', true ...
262 );
263
264 gr1_FUN = im_gr1.get('GR');
265
266 im_gr2 = ImporterGroupSubjectFUN_XLS( ...
267     'DIRECTORY', [fileparts(which('SubjectCON_FUN_MP')) filesep 'Example
      data CON_FUN_MP_XLS' filesep 'CON_FUN_MP_Group_2_XLS.FUN'], ...
268     'BA', ba, ...
269     'WAITBAR', true ...
270 );
271
272 gr2_FUN = im_gr2.get('GR');
273
274 % Combine Groups of SubjectCON with Groups of SubjectFUN
275 co_gr1 = CombineGroups_CON_FUN_MP( ...
276     'GR_CON', gr1_CON, ...
277     'GR_FUN', gr1_FUN, ...

```

③ tests with the MultiplexWU element which contains weighted undirected adjacency matrix from multiplex graph.


```

278     'WAITBAR', true ...
279 );
280
281 gr1 = co_gr1.get('GR_CON_FUN_MP');
282
283 co_gr2 = CombineGroups_CON_FUN_MP( ...
284     'GR_CON', gr2_CON, ...
285     'GR_FUN', gr2_FUN, ...
286     'WAITBAR', true ...
287 );
288
289 gr2 = co_gr2.get('GR_CON_FUN_MP');
290
291 % Analysis CON FUN MP WU
292 a_WU1 = AnalyzeEnsemble_CON_FUN_MP_WU( ...
293     'GR', gr1 ...
294 );
295
296 a_WU2 = AnalyzeEnsemble_CON_FUN_MP_WU( ...
297     'TEMPLATE', a_WU1, ...
298     'GR', gr2 ...
299 );
300
301 a_WU1.memorize('G_DICT');
302 a_WU2.memorize('G_DICT');
303
304 % create item lists of NNDataPoint_Graph_CLA
305 [~, group_folder_name] = fileparts(im_gr1.get('DIRECTORY'));
306 it_list1 = cellfun(@(x) NNDataPoint_Graph_CLA( ...
307     'ID', x.get('ID'), ...
308     'G', x, ...
309     'TARGET_IDS', {group_folder_name}), ...
310     a_WU1.get('G_DICT').get('IT_LIST'), ...
311     'UniformOutput', false);
312
313 [~, group_folder_name] = fileparts(im_gr2.get('DIRECTORY'));
314 it_list2 = cellfun(@(x) NNDataPoint_Graph_CLA( ...
315     'ID', x.get('ID'), ...
316     'G', x, ...
317     'TARGET_IDS', {group_folder_name}), ...
318     a_WU2.get('G_DICT').get('IT_LIST'), ...
319     'UniformOutput', false);
320
321 % create NNDataPoint_Graph_CLA DICT items
322 dp_list1 = IndexedDictionary(...
323     'IT_CLASS', 'NNDataPoint_Graph_CLA', ...
324     'IT_LIST', it_list1 ...
325 );
326
327 dp_list2 = IndexedDictionary(...
328     'IT_CLASS', 'NNDataPoint_Graph_CLA', ...
329     'IT_LIST', it_list2 ...
330 );
331
332 % create a NNDataset containing the NNDataPoint_Graph_CLA DICT
333 d1 = NNDataset( ...
334     'DP_CLASS', 'NNDataPoint_Graph_CLA', ...
335     'DP_DICT', dp_list1 ...
336 );
337
338 d2 = NNDataset( ...

```

```

339     'DP_CLASS', 'NNDatapoint_Graph_CLA', ...
340     'DP_DICT', dp_list2 ...
341 );
342
343 % Check whether the content of input for a single datapoint matches
344 for index = 1:1:gr1.get('SUB_DICT').get('LENGTH')
345     individual_input = d1.get('DP_DICT').get('IT', index).get('INPUT');
346     known_input = a_WU1.get('G_DICT').get('IT', index).get('A');
347
348     assert(isequal(individual_input, known_input), ...
349         [BRAPH2.STR ':NNDatapoint_Graph_CLA:' BRAPH2.FAIL_TEST], ...
350         'NNDatapoint_Graph_CLA does not construct the dataset correctly. The
351         input value is not derived correctly.' ...
352     )
353 end
354
355 for index = 1:1:gr2.get('SUB_DICT').get('LENGTH')
356     individual_input = d2.get('DP_DICT').get('IT', index).get('INPUT');
357     known_input = a_WU2.get('G_DICT').get('IT', index).get('A');
358
359     assert(isequal(individual_input, known_input), ...
360         [BRAPH2.STR ':NNDatapoint_Graph_CLA:' BRAPH2.FAIL_TEST], ...
361         'NNDatapoint_Graph_CLA does not construct the dataset correctly. The
362         input value is not derived correctly.' ...
363     )
364 end
365
366 %%% itest!
367 %%% iname! ④
368 Example script for weighted undirected graph (GraphWU) using connectivity
369 data
370 %%% icode!
371 if ~isfile([fileparts(which('NNDatapoint_CON_CLA')) filesep 'Example data NN
372     CLA CON XLS' filesep 'atlas.xlsx'])
373     test_NNDatapoint_CON_CLA % create example files
374 end
375 example_NNCV_CON_WU_CLA
376
377 %%% itest!
378 %%% iname! ⑤
379 Example script for binary undirected graph at fixed densities (MultigraphBUD
380 ) using connectivity data
381 %%% icode!
382 if ~isfile([fileparts(which('NNDatapoint_CON_CLA')) filesep 'Example data NN
383     CLA CON XLS' filesep 'atlas.xlsx'])
384     test_NNDatapoint_CON_CLA % create example files
385 end
386 example_NNCV_CON_BUD_CLA
387
388 %%% itest!
389 %%% iname! ⑥
390 Example script for weighted undirected multiplex (MultiplexWU) using
391 connectivity data and functional data
392 %%% icode!
393 if ~isfile([fileparts(which('NNDatapoint_CON_FUN_MP_CLA')) filesep 'Example
394     data NN CLA CON_FUN_MP XLS' filesep 'atlas.xlsx'])
395     test_NNDatapoint_CON_FUN_MP_CLA % create example files
396 end
397 example_NNCV_CON_FUN_MP_WU_CLA

```

④ tests with the GraphWU element with simulated data.

⑤ tests with the MultigraphBUD element with simulated data.

⑥ tests with the MultiplexWU element with simulated data.

Implementation of a Data Point with Graph Measures

Graph Measure Data Point for Regression (NNDataPoint_Measure_REG)

Now we implement NNDataPoint_Measure_REG based on previous codes NNDataPoint_Graph_REG. This neural network datapoint utilizes graph measures obtained from the adjacency matrix from the derived graph of the subject. The modified parts of the code are highlighted.

Code 17: NNDataPoint_Measure_REG element

header. The header section of the generator code for `_NNDataPoint_Measure_REG.gen.m` provides the general information about the NNDataPoint_Measure_REG element.

```

1 %% iheader!
2 NNDataPoint_Measure_REG < NNDataPoint (dp, measure regression data point)
   is a data point for regression with graph measures.
3
4 %% idescription!
5 A data point for regression with graph measures (NNDataPoint_Measure_REG)
6 contains both input and target for neural network analysis.
7 The input is the value of the graph measures (e.g. Degree, DegreeAv, and
   Distance),
8 calculated from the derived graph of the subject.
9 The target is obtained from the variables of interest of the subject.

```

Code 18: NNDataPoint_Measure_REG element prop up-

date. The props_update section of the generator code for `_NNDataPoint_Measure_REG.gen.m` updates the properties of the NNDataPoint_Measure_REG element. This defines the core properties of the data point.

```

1 %% iprops_update!
2
3 %% iprop!
4 NAME (constant, string) is the name of a data point for regression with
   graph measures.
5 %%%% idefault!
6 'NNDataPoint_Measure_REG'
7
8 %% iprop!
9 DESCRIPTION (constant, string) is the description of a data point for
   regression with graph measures.
10 %%%% idefault!
11 'A data point for regression with graph measures (NNDataPoint_Measure_REG)
   contains both input and target for neural network analysis. The input
   is the value of the graph measures (e.g. Degree, DegreeAv, and Distance
   ), calculated from the derived graph of the subject. The target is
   obtained from the variables of interest of the subject.'
12
13 %% iprop!
14 TEMPLATE (parameter, item) is the template of a data point for regression
   with graph measures.
15 %%%% isettings!
16 'NNDataPoint_Measure_REG'

```

```

17
18 %% iprop!
19 ID (data, string) is a few-letter code for a data point for regression with
    graph measures.
20 %%%% idefault!
21 'NNDataPoint_Measure_REG ID'
22
23 %% iprop!
24 LABEL (metadata, string) is an extended label of a data point for regression
    with graph measures.
25 %%%% idefault!
26 'NNDataPoint_Measure_REG label'
27
28 %% iprop!
29 NOTES (metadata, string) are some specific notes about a data point for
    regression with graph measures.
30 %%%% idefault!
31 'NNDataPoint_Measure_REG notes'
32
33 %% iprop!
34 INPUT (result, cell) is the input value for this data point.
35 %%%% icalculate!
36 value = cellfun(@(m_class) dp.get('G').get('MEASURE', m_class).get('M'), dp.
    get('M_LIST'), 'UniformOutput', false); ①
37
38 %% iprop!
39 TARGET (result, cell) is the target value for this data point.
40 %%%% icalculate!
41 value = cellfun(@(x) dp.get('SUB').get('VOI_DICT').get('IT', x).get('V'), dp
    .get('TARGET_IDS'), 'UniformOutput', false);

```

① calculates or extract the graph measures, which are specified with M_LIST from a Graph element for this data point. Note that a Graph can be any kind of Graph, including GraphWU, MultigraphBUD, and MultiplexBUT, among others.

Code 19: NNDataPoint_Measure_REG element props. The props section of generator code for `_NNDataPoint_Measure_REG.gen.m` defines the properties to be used in `NNDataPoint_Measure_REG`.

```

1 %% iprops!
2
3 %%%% iprop!
4 G (data, item) is a graph.
5 %%%% isettings!
6 'Graph'
7
8 %%%% iprop! ①
9 M_LIST (parameter, classlist) is a list of graph measure to be used as the
    input
10
11 %% iprop!
12 SUB (data, item) is a subject.
13 %%%% isettings!
14 'Subject'
15
16 %% iprop!
17 TARGET_IDS (parameter, stringlist) is a list of variable-of-interest IDs to
    be used as the class targets.

```

① defines the graph measure list which will be obtained as INPUT for this data point.

Code 20: **NNDataPoint_Measure_REG element tests**. The tests section from the element generator `_NNDataPoint_Measure_REG.gen.m`. A test for creating example files should be prepared to test the properties of the data point. Furthermore, additional test should be prepared for validating the value of input and target for the data point.

```

1 %% itests!
2
3 %%% iexcluded_props!
4 [NNDataPoint_Measure_REG.G NNDataPoint_Measure_REG.SUB]
5
6 %%% itest!
7 %%% iname! ①
8 Construct the data point with the adjacency matrix derived from its weighted
   undirected graph (GraphWU)
9 %%% icode!
10 % ensure the example data is generated
11 if ~isfile([fileparts(which('NNDataPoint_CON_REG')) filesep 'Example data NN
   REG CON XLS' filesep 'atlas.xlsx'])
12     test_NNDataPoint_CON_REG % create example files
13 end
14
15 % Load BrainAtlas
16 im_ba = ImporterBrainAtlasXLS( ...
17     'FILE', [fileparts(which('NNDataPoint_CON_REG')) filesep 'Example data
   NN REG CON XLS' filesep 'atlas.xlsx'], ...
18     'WAITBAR', true ...
19 );
20
21 ba = im_ba.get('BA');
22
23 % Load Groups of SubjectCON
24 im_gr = ImporterGroupSubjectCON_XLS( ...
25     'DIRECTORY', [fileparts(which('NNDataPoint_CON_REG')) filesep 'Example
   data NN REG CON XLS' filesep 'CON_Group_XLS'], ...
26     'BA', ba, ...
27     'WAITBAR', true ...
28 );
29
30 gr = im_gr.get('GR');
31
32 % Analysis CON WU
33 a_WU = AnalyzeEnsemble_CON_WU( ...
34     'GR', gr ...
35 );
36
37 a_WU.get('MEASUREENSEMBLE', 'Degree').get('M'); ②
38 a_WU.get('MEASUREENSEMBLE', 'DegreeAv').get('M'); ③
39 a_WU.get('MEASUREENSEMBLE', 'Distance').get('M'); ④
40
41 % create item lists of NNDataPoint_Measure_REG
42 it_list = cellfun(@(g, sub) NNDataPoint_Measure_REG( ...
43     'ID', sub.get('ID'), ...
44     'G', g, ...
45     'M_LIST', {'Degree' 'DegreeAv' 'Distance'}, ...
46     'SUB', sub, ...
47     'TARGET_IDS', sub.get('VOI_DICT').get('KEYS')), ...
48     a_WU.get('G_DICT').get('IT_LIST'), gr.get('SUB_DICT').get('IT_LIST'), ...
49     'UniformOutput', false);

```

①, ②, ③, and ④ test adding various kinds of graph measure with the GraphWU element which contains weighted undirected adjacency matrix.

```

50
51 % create NNDataPoint_Measure_REG DICT items
52 dp_list = IndexedDictionary(...
53     'IT_CLASS', 'NNDataPoint_Measure_REG', ...
54     'IT_LIST', it_list ...
55 );
56
57 % create a NNDataset containing the NNDataPoint_Measure_REG DICT
58 d = NNDataset( ...
59     'DP_CLASS', 'NNDataPoint_Measure_REG', ...
60     'DP_DICT', dp_list ...
61 );
62
63 % Check whether the content of input for a single datapoint matches
64 for index = 1:1:gr.get('SUB_DICT').get('LENGTH')
65     individual_input = d.get('DP_DICT').get('IT', index).get('INPUT');
66     known_input = cellfun(@(m) m.get('M'), a_WU.get('G_DICT').get('IT',
67         index).get('M_DICT').get('IT_LIST'), 'UniformOutput', false);
68
69     assert(isequal(individual_input, known_input), ...
70         [BRAPH2.STR ':NNDataPoint_Measure_REG:' BRAPH2.FAIL_TEST], ...
71         'NNDataPoint_Measure_REG does not construct the dataset correctly.
72         The input value is not derived correctly.' ...
73     )
74 end
75
76 %%% itest!
77 %%% iname! ⑤
78 Construct the data point with the adjacency matrix derived from its binary
79 undirected multigraph with fixed densities (MultigraphBUD)
80 %%% icode!
81 % ensure the example data is generated
82 if ~isfile([fileparts(which('NNDataPoint_CON_REG')) filesep 'Example data NN
83     REG CON XLS' filesep 'atlas.xlsx'])
84     test_NNDataPoint_CON_REG % create example files
85 end
86
87 % Load BrainAtlas
88 im_ba = ImporterBrainAtlasXLS( ...
89     'FILE', [fileparts(which('NNDataPoint_CON_REG')) filesep 'Example data
90     NN REG CON XLS' filesep 'atlas.xlsx'], ...
91     'WAITBAR', true ...
92 );
93
94 ba = im_ba.get('BA');
95
96 % Load Groups of SubjectCON
97 im_gr = ImporterGroupSubjectCON_XLS( ...
98     'DIRECTORY', [fileparts(which('NNDataPoint_CON_REG')) filesep 'Example
99     data NN REG CON XLS' filesep 'CON_Group_XLS'], ...
100     'BA', ba, ...
101     'WAITBAR', true ...
102 );
103
104 gr = im_gr.get('GR');
105
106 % Analysis CON WU
107 densities = 0:25:100;
108
109 a_BUD = AnalyzeEnsemble_CON_BUD( ...
110     'DENSITIES', densities, ...

```

⑤ test adding various kinds of graph measure with the MultigraphBUD.

```

105     'GR', gr ...
106 );
107
108 a_BUD.get('MEASUREENSEMBLE', 'Degree').get('M');
109 a_BUD.get('MEASUREENSEMBLE', 'DegreeAv').get('M');
110 a_BUD.get('MEASUREENSEMBLE', 'Distance').get('M');
111
112 % create item lists of NNDataPoint_Measure_REG
113 it_list = cellfun(@(g, sub) NNDataPoint_Measure_REG( ...
114     'ID', sub.get('ID'), ...
115     'G', g, ...
116     'M_LIST', {'Degree' 'DegreeAv' 'Distance'}, ...
117     'SUB', sub, ...
118     'TARGET_IDS', sub.get('VOI_DICT').get('KEYS')), ...
119     a_BUD.get('G_DICT').get('IT_LIST'), gr.get('SUB_DICT').get('IT_LIST')
120     , ...
121     'UniformOutput', false);
122
123 % create NNDataPoint_CON_CLA_DICT items
124 dp_list = IndexedDictionary(...
125     'IT_CLASS', 'NNDataPoint_Measure_REG', ...
126     'IT_LIST', it_list ...
127 );
128
129 % create a NNDataset containing the NNDataPoint_Measure_REG_DICT
130 d = NNDataset( ...
131     'DP_CLASS', 'NNDataPoint_Measure_REG', ...
132     'DP_DICT', dp_list ...
133 );
134
135 % Check whether the content of input for a single datapoint matches
136 for index = 1:1:gr.get('SUB_DICT').get('LENGTH')
137     individual_input = d.get('DP_DICT').get('IT', index).get('INPUT');
138     known_input = cellfun(@(m) m.get('M'), a_BUD.get('G_DICT').get('IT',
139         index).get('M_DICT').get('IT_LIST'), 'UniformOutput', false);
140
141     assert(isequal(individual_input, known_input), ...
142         [BRAPH2.STR ':NNDataPoint_Measure_REG:' BRAPH2.FAIL_TEST], ...
143         'NNDataPoint_Measure_REG does not construct the dataset correctly.
144         The input value is not derived correctly.' ...
145         )
146 end
147
148 %%% itest!
149 %%% iname! ⑥
150 Construct the data point with the adjacency matrix derived from its
151     multiplex weighted undirected graph (MultiplexWU)
152 %%% icode!
153 % ensure the example data is generated
154 if ~isfile([fileparts(which('SubjectCON_FUN_MP')) filesep 'Example data
155     CON_FUN_MP_XLS' filesep 'atlas.xlsx'])
156     test_SubjectCON_FUN_MP % create example files
157 end
158
159 % Load BrainAtlas
160 im_ba = ImporterBrainAtlasXLS( ...
161     'FILE', [fileparts(which('SubjectCON_FUN_MP')) filesep 'Example data
162     CON_FUN_MP_XLS' filesep 'atlas.xlsx'], ...
163     'WAITBAR', true ...
164 );
165

```

⑥ test adding various kinds of graph measure with the MultiplexWU.

```

160 ba = im_ba.get('BA');
161
162 % Load Groups of SubjectCON
163 im_gr = ImporterGroupSubjectCON_XLS( ...
164     'DIRECTORY', [fileparts(which('SubjectCON_FUN_MP')) filesep 'Example
165     data CON_FUN_MP_XLS' filesep 'CON_FUN_MP_Group_1.XLS.CON'], ...
166     'BA', ba, ...
167     'WAITBAR', true ...
168 );
169 gr_CON = im_gr.get('GR');
170
171 % Load Groups of SubjectFUN
172 im_gr = ImporterGroupSubjectFUN_XLS( ...
173     'DIRECTORY', [fileparts(which('SubjectCON_FUN_MP')) filesep 'Example
174     data CON_FUN_MP_XLS' filesep 'CON_FUN_MP_Group_1.XLS.FUN'], ...
175     'BA', ba, ...
176     'WAITBAR', true ...
177 );
178 gr_FUN = im_gr.get('GR');
179
180 % Combine Groups of SubjectCON with Groups of SubjectFUN
181 co_gr = CombineGroups_CON_FUN_MP( ...
182     'GR_CON', gr_CON, ...
183     'GR_FUN', gr_FUN, ...
184     'WAITBAR', true ...
185 );
186
187 gr = co_gr.get('GR_CON_FUN_MP');
188
189 % Analysis CON FUN MP WU
190 a_WU = AnalyzeEnsemble_CON_FUN_MP_WU( ...
191     'GR', gr ...
192 );
193
194 % To be added the multiplex measures
195 a_WU.get('MEASUREENSEMBLE', 'Degree').get('M');
196 a_WU.get('MEASUREENSEMBLE', 'DegreeAv').get('M');
197 a_WU.get('MEASUREENSEMBLE', 'Distance').get('M');
198
199 % create item lists of NNDataPoint_Measure_REG
200 it_list = cellfun(@(g, sub) NNDataPoint_Measure_REG( ...
201     'ID', sub.get('ID'), ...
202     'G', g, ...
203     'M_LIST', {'Degree' 'DegreeAv' 'Distance'}, ...
204     'SUB', sub, ...
205     'TARGET_IDS', sub.get('VOI_DICT').get('KEYS')), ...
206     a_WU.get('G_DICT').get('IT_LIST'), gr.get('SUB_DICT').get('IT_LIST'), ...
207     'UniformOutput', false);
208
209 % create NNDataPoint_Measure_REG DICT items
210 dp_list = IndexedDictionary(...
211     'IT_CLASS', 'NNDataPoint_Measure_REG', ...
212     'IT_LIST', it_list ...
213 );
214
215 % create a NNDataset containing the NNDataPoint_Measure_REG DICT
216 d = NNDataset( ...
217     'DP_CLASS', 'NNDataPoint_Measure_REG', ...
218     'DP_DICT', dp_list ...

```



```

219 );
220
221 % Check whether the content of input for a single datapoint matches
222 for index = 1:1:gr.get('SUB_DICT').get('LENGTH')
223     individual_input = d.get('DP_DICT').get('IT', index).get('INPUT');
224     known_input = cellfun(@(m) m.get('M'), a_WU.get('G_DICT').get('IT',
        index).get('M_DICT').get('IT_LIST'), 'UniformOutput', false);
225
226     assert(isequal(individual_input, known_input), ...
227         [BRAPH2.STR ':NNDatapoint_Measure_REG:' BRAPH2.FAIL_TEST], ...
228         'NNDatapoint_Measure_REG does not construct the dataset correctly.
        The input value is not derived correctly.' ...
229     )
230 end
231
232 %%% itest!
233 %%% iname! ⑦
234 Example script for weighted undirected graph (GraphWU) using connectivity
    data
235 %%% icode!
236 if ~isfile([fileparts(which('NNDatapoint_CON_REG')) filesep 'Example data NN
    REG CON XLS' filesep 'atlas.xlsx'])
237     test_NNDatapoint_CON_REG % create example files
238 end
239 example_NNCV_CON_WU_M_REG
240
241 %%% itest!
242 %%% iname! ⑧
243 Example script for weighted undirected multiplex (MultiplexWU) using
    connectivity data and functional data
244 %%% icode!
245 if ~isfile([fileparts(which('NNDatapoint_CON_FUN_MP_REG')) filesep 'Example
    data NN REG CON_FUN_MP XLS' filesep 'atlas.xlsx'])
246     test_NNDatapoint_CON_FUN_MP_REG % create example files
247 end
248 example_NNCV_CON_FUN_MP_WU_M_REG

```

⑦ test adding various kinds of graph measure with the GraphWU using example data.

⑧ test adding various kinds of graph measure with the MultiplexWU using example data.

Graph Measure Data Point for Classification (NNDataPoint_Measure_CLA)

Now we implement NNDataPoint_Measure_CLA based on previous codes NNDataPoint_Graph_CLA. This neural network datapoint utilizes graph measures obtained from the adjacency matrix from the derived graph of the subject. The modified parts of the code are highlighted.

Code 21: NNDataPoint_Measure_CLA element

header. The header section of the generator code for `_NNDataPoint_Measure_CLA.gen.m` provides the general information about the NNDataPoint_Measure_CLA element.

```

1 %% iheader!
2 NNDataPoint_Measure_CLA < NNDataPoint (dp, measure classification data point
   ) is a data point for classification with graph measures.
3
4 %%% idescription!
5 A data point for classification with graph measures (NNDataPoint_Measure_CLA
   )
6 contains both input and target for neural network analysis.
7 The input is the value of the graph measures (e.g. Degree, DegreeAv, and
   Distance),
8 calculated from the derived graph of the subject.
9 The target is obtained from the variables of interest of the subject.

```

Code 22: NNDataPoint_Measure_CLA element prop up-

date. The props_update section of the generator code for `_NNDataPoint_Measure_CLA.gen.m` updates the properties of the NNDataPoint_Measure_CLA element. This defines the core properties of the data point.

```

1 %% iprops_update!
2
3 %%% iprop!
4 NAME (constant, string) is the name of a data point for classification with
   graph measures.
5 %%% idefault!
6 'NNDataPoint_Measure_CLA'
7
8 %%% iprop!
9 DESCRIPTION (constant, string) is the description of a data point for
   classification with graph measures.
10 %%% idefault!
11 'A data point for classification with graph measures (
    NNDataPoint_Measure_CLA) contains both input and target for neural
    network analysis. The input is the value of the graph measures (e.g.
    Degree, DegreeAv, and Distance), calculated from the derived graph of
    the subject. The target is obtained from the variables of interest of
    the subject.'
12
13 %%% iprop!
14 TEMPLATE (parameter, item) is the template of a data point for
   classification with graph measures.
15 %%% isettings!
16 'NNDataPoint_Measure_CLA'
17

```

```

18 %%% iprop!
19 ID (data, string) is a few-letter code for a data point for classification
    with graph measures.
20 %%% idefault!
21 'NNDataPoint_Measure_CLA ID'
22
23 %%% iprop!
24 LABEL (metadata, string) is an extended label of a data point for
    classification with graph measures.
25 %%% idefault!
26 'NNDataPoint_Measure_CLA label'
27
28 %%% iprop!
29 NOTES (metadata, string) are some specific notes about a data point for
    classification with graph measures.
30 %%% idefault!
31 'NNDataPoint_Measure_CLA notes'
32
33 %%% iprop!
34 INPUT (result, cell) is the input value for this data point.
35 %%% icalculate!
36 value = cellfun(@(m_class) dp.get('G').get('MEASURE', m_class).get('M'), dp.
    get('M_LIST'), 'UniformOutput', false); ①
37
38 %%% iprop!
39 TARGET (result, cell) is the target value for this data point.
40 %%% icalculate!
41 value = dp.get('TARGET_IDS');

```

Code 23: NNDataPoint_Measure_CLA element props. The props section of generator code for `_NNDataPoint_Measure_CLA.gen.m` defines the properties to be used in `NNDataPoint_Measure_CLA`.

```

1 %%% iprops!
2
3 %%% iprop!
4 G (data, item) is a graph.
5 %%% isettings!
6 'Graph'
7
8 %%% iprop! ①
9 M_LIST (parameter, classlist) is a list of graph measure to be used as the
    input
10
11 %%% iprop!
12 TARGET_IDS (parameter, stringlist) is a list of variable-of-interest IDs to
    be used as the class targets.

```

① calculates or extract the graph measures, which are specified with `M_LIST` from a Graph element for this data point. Note that a Graph can be any kind of Graph, including `GraphWU`, `MultigraphBUD`, and `MultiplexBUT`, among others.

① defines the graph measure list which will be obtained as `INPUT` for this data point.

Code 24: **NNDataPoint_Measure_CLA element tests**. The tests section from the element generator `_NNDataPoint_Measure_CLA.gen.m`. A test for creating example files should be prepared to test the properties of the data point. Furthermore, additional test should be prepared for validating the value of input and target for the data point.

```

1 %% itests!
2
3 %%% iexcluded_props!
4 [NNDataPoint_Measure_CLA.G]
5
6 %%% itest!
7 %%% iname! ①
8 Construct the data point with the graph measures derived from its weighted
   undirected graph (GraphWU)
9 %%% icode!
10 % ensure the example data is generated
11 if ~isfile([fileparts(which('NNDataPoint_CON_CLA')) filesep 'Example data NN
   CLA CON XLS' filesep 'atlas.xlsx'])
12     test_NNDataPoint_CON_CLA % create example files
13 end
14
15 % Load BrainAtlas
16 im_ba = ImporterBrainAtlasXLS( ...
17     'FILE', [fileparts(which('NNDataPoint_CON_CLA')) filesep 'Example data
   NN CLA CON XLS' filesep 'atlas.xlsx'], ...
18     'WAITBAR', true ...
19 );
20
21 ba = im_ba.get('BA');
22
23 % Load Groups of SubjectCON
24 im_gr1 = ImporterGroupSubjectCON_XLS( ...
25     'DIRECTORY', [fileparts(which('NNDataPoint_CON_CLA')) filesep 'Example
   data NN CLA CON XLS' filesep 'CON_Group_1_XLS'], ...
26     'BA', ba, ...
27     'WAITBAR', true ...
28 );
29
30 gr1 = im_gr1.get('GR');
31
32 im_gr2 = ImporterGroupSubjectCON_XLS( ...
33     'DIRECTORY', [fileparts(which('NNDataPoint_CON_CLA')) filesep 'Example
   data NN CLA CON XLS' filesep 'CON_Group_2_XLS'], ...
34     'BA', ba, ...
35     'WAITBAR', true ...
36 );
37
38 gr2 = im_gr2.get('GR');
39
40 % Analysis CON WU
41 a_WU1 = AnalyzeEnsemble_CON_WU( ...
42     'GR', gr1 ...
43 );
44
45 a_WU2 = AnalyzeEnsemble_CON_WU( ...
46     'TEMPLATE', a_WU1, ...
47     'GR', gr2 ...
48 );
49

```

①, ②, ③, and ④ test adding various kinds of graph measure with the GraphWU element which contains weighted undirected adjacency matrix.

```

50 a_WU1.get('MEASUREENSEMBLE', 'Degree').get('M'); ②
51 a_WU1.get('MEASUREENSEMBLE', 'DegreeAv').get('M'); ③
52 a_WU1.get('MEASUREENSEMBLE', 'Distance').get('M'); ④
53
54 a_WU2.get('MEASUREENSEMBLE', 'Degree').get('M');
55 a_WU2.get('MEASUREENSEMBLE', 'DegreeAv').get('M');
56 a_WU2.get('MEASUREENSEMBLE', 'Distance').get('M');
57
58 % create item lists of NNDataPoint_Graph_CLA
59 [~, group_folder_name] = fileparts(im_gr1.get('DIRECTORY'));
60 it_list1 = cellfun(@(x) NNDataPoint_Measure_CLA( ...
61     'ID', x.get('ID'), ...
62     'G', x, ...
63     'M_LIST', {'Degree' 'DegreeAv' 'Distance'}, ...
64     'TARGET_IDS', {group_folder_name}), ...
65     a_WU1.get('G_DICT').get('IT_LIST'), ...
66     'UniformOutput', false);
67
68 [~, group_folder_name] = fileparts(im_gr2.get('DIRECTORY'));
69 it_list2 = cellfun(@(x) NNDataPoint_Measure_CLA( ...
70     'ID', x.get('ID'), ...
71     'G', x, ...
72     'M_LIST', {'Degree' 'DegreeAv' 'Distance'}, ...
73     'TARGET_IDS', {group_folder_name}), ...
74     a_WU2.get('G_DICT').get('IT_LIST'), ...
75     'UniformOutput', false);
76
77 % create NNDataPoint_Graph_CLA DICT items
78 dp_list1 = IndexedDictionary(...
79     'IT_CLASS', 'NNDataPoint_Measure_CLA', ...
80     'IT_LIST', it_list1 ...
81     );
82
83 dp_list2 = IndexedDictionary(...
84     'IT_CLASS', 'NNDataPoint_Measure_CLA', ...
85     'IT_LIST', it_list2 ...
86     );
87
88 % create a NNDataset containing the NNDataPoint_Measure_CLA DICT
89 d1 = NNDataset( ...
90     'DP_CLASS', 'NNDataPoint_Measure_CLA', ...
91     'DP_DICT', dp_list1 ...
92     );
93
94 d2 = NNDataset( ...
95     'DP_CLASS', 'NNDataPoint_Measure_CLA', ...
96     'DP_DICT', dp_list2 ...
97     );
98
99 % Check whether the content of input for a single datapoint matches
100 for index = 1:1:gr1.get('SUB_DICT').get('LENGTH')
101     individual_input = d1.get('DP_DICT').get('IT', index).get('INPUT');
102     known_input = cellfun(@(m) m.get('M'), a_WU1.get('G_DICT').get('IT',
103         index).get('M_DICT').get('IT_LIST'), 'UniformOutput', false);
104
105     assert(isequal(individual_input, known_input), ...
106         [BRAPH2.STR ':NNDataPoint_Measure_CLA:' BRAPH2.FAIL_TEST], ...
107         'NNDataPoint_Measure_CLA does not construct the dataset correctly.
108         The input value is not derived correctly.' ...
109         )
110 end

```

```

109
110 for index = 1:1:gr2.get('SUB_DICT').get('LENGTH')
111     individual_input = d2.get('DP_DICT').get('IT', index).get('INPUT');
112     known_input = cellfun(@(m) m.get('M'), a_WU2.get('G_DICT').get('IT',
113         index).get('M_DICT').get('IT_LIST'), 'UniformOutput', false);
114
115     assert(isequal(individual_input, known_input), ...
116         [BRAPH2.STR 'NNDatapoint_Measure_CLA:' BRAPH2.FAIL_TEST], ...
117         'NNDatapoint_Measure_CLA does not construct the dataset correctly.
118         The input value is not derived correctly.' ...
119     )
120 end
121
122 %%% itest!
123 %%% iname! ⑤
124 Construct the data point with the graph measures derived from its binary
125 undirected multigraph with fixed densities (MultigraphBUD)
126 %%% icode!
127 % ensure the example data is generated
128 if ~isfile([fileparts(which('NNDatapoint_CON_CLA')) filesep 'Example data NN
129     CLA CON XLS' filesep 'atlas.xlsx'])
130     test_NNDatapoint_CON_CLA % create example files
131 end
132
133 % Load BrainAtlas
134 im_ba = ImporterBrainAtlasXLS( ...
135     'FILE', [fileparts(which('NNDatapoint_CON_CLA')) filesep 'Example data
136     NN CLA CON XLS' filesep 'atlas.xlsx'], ...
137     'WAITBAR', true ...
138 );
139
140 ba = im_ba.get('BA');
141
142 % Load Groups of SubjectCON
143 im_gr1 = ImporterGroupSubjectCON_XLS( ...
144     'DIRECTORY', [fileparts(which('NNDatapoint_CON_CLA')) filesep 'Example
145     data NN CLA CON XLS' filesep 'CON_Group_1_XLS'], ...
146     'BA', ba, ...
147     'WAITBAR', true ...
148 );
149
150 gr1 = im_gr1.get('GR');
151
152 im_gr2 = ImporterGroupSubjectCON_XLS( ...
153     'DIRECTORY', [fileparts(which('NNDatapoint_CON_CLA')) filesep 'Example
154     data NN CLA CON XLS' filesep 'CON_Group_2_XLS'], ...
155     'BA', ba, ...
156     'WAITBAR', true ...
157 );
158
159 gr2 = im_gr2.get('GR');
160
161 % Analysis CON WU
162 densities = 0:25:100;
163
164 a_BUD1 = AnalyzeEnsemble_CON_BUD( ...
165     'DENSITIES', densities, ...
166     'GR', gr1 ...
167 );
168
169 a_BUD2 = AnalyzeEnsemble_CON_BUD( ...

```

⑤ test adding various kinds of graph measure with the MultigraphBUD.

```

163     'TEMPLATE', a_BUD1, ...
164     'GR', gr2 ...
165 );
166
167 a_BUD1.get('MEASUREENSEMBLE', 'Degree').get('M');
168 a_BUD1.get('MEASUREENSEMBLE', 'DegreeAv').get('M');
169 a_BUD1.get('MEASUREENSEMBLE', 'Distance').get('M');
170
171 a_BUD2.get('MEASUREENSEMBLE', 'Degree').get('M');
172 a_BUD2.get('MEASUREENSEMBLE', 'DegreeAv').get('M');
173 a_BUD2.get('MEASUREENSEMBLE', 'Distance').get('M');
174
175 % create item lists of NNDataPoint_Graph_CLA
176 [~, group_folder_name] = fileparts(im_gr1.get('DIRECTORY'));
177 it_list1 = cellfun(@(x) NNDataPoint_Measure_CLA( ...
178     'ID', x.get('ID'), ...
179     'G', x, ...
180     'M_LIST', {'Degree' 'DegreeAv' 'Distance'}, ...
181     'TARGET_IDS', {group_folder_name}), ...
182     a_BUD1.get('G_DICT').get('IT_LIST'), ...
183     'UniformOutput', false);
184
185 [~, group_folder_name] = fileparts(im_gr2.get('DIRECTORY'));
186 it_list2 = cellfun(@(x) NNDataPoint_Measure_CLA( ...
187     'ID', x.get('ID'), ...
188     'G', x, ...
189     'M_LIST', {'Degree' 'DegreeAv' 'Distance'}, ...
190     'TARGET_IDS', {group_folder_name}), ...
191     a_BUD2.get('G_DICT').get('IT_LIST'), ...
192     'UniformOutput', false);
193
194 % create NNDataPoint_Graph_CLA DICT items
195 dp_list1 = IndexedDictionary(...
196     'IT_CLASS', 'NNDataPoint_Measure_CLA', ...
197     'IT_LIST', it_list1 ...
198 );
199
200 dp_list2 = IndexedDictionary(...
201     'IT_CLASS', 'NNDataPoint_Measure_CLA', ...
202     'IT_LIST', it_list2 ...
203 );
204
205 % create a NNDataset containing the NNDataPoint_Measure_CLA DICT
206 d1 = NNDataset( ...
207     'DP_CLASS', 'NNDataPoint_Measure_CLA', ...
208     'DP_DICT', dp_list1 ...
209 );
210
211 d2 = NNDataset( ...
212     'DP_CLASS', 'NNDataPoint_Measure_CLA', ...
213     'DP_DICT', dp_list2 ...
214 );
215
216 % Check whether the content of input for a single datapoint matches
217 for index = 1:1:gr1.get('SUB_DICT').get('LENGTH')
218     individual_input = d1.get('DP_DICT').get('IT', index).get('INPUT');
219     known_input = cellfun(@(m) m.get('M'), a_BUD1.get('G_DICT').get('IT',
220         index).get('M_DICT').get('IT_LIST'), 'UniformOutput', false);
221
222     assert(isequal(individual_input, known_input), ...
223         [BRAPH2.STR ':NNDataPoint_Measure_CLA:' BRAPH2.FAIL_TEST], ...

```

```

223     'NNDataPoint_Measure_CLA does not construct the dataset correctly.
    The input value is not derived correctly.' ...
224 )
225 end
226
227 for index = 1:1:gr2.get('SUB_DICT').get('LENGTH')
228     individual_input = d2.get('DP_DICT').get('IT', index).get('INPUT');
229     known_input = cellfun(@(m) m.get('M'), a_BUD2.get('G_DICT').get('IT',
        index).get('M_DICT').get('IT_LIST'), 'UniformOutput', false);
230
231     assert(isequal(individual_input, known_input), ...
232         [BRAPH2.STR 'NNDataPoint_Measure_CLA:' BRAPH2.FAIL_TEST], ...
233         'NNDataPoint_Measure_CLA does not construct the dataset correctly.
        The input value is not derived correctly.' ...
234     )
235 end
236
237 %%% itest!
238 %%% iname!
239 Construct the data point with the graph measures derived from its multiplex
    weighted undirected graph (MultiplexWU)
240 %%% icode!
241 % ensure the example data is generated
242 if ~isfile([fileparts(which('SubjectCON_FUN_MP')) filesep 'Example data
    CON_FUN_MP XLS' filesep 'atlas.xlsx'])
243     test_SubjectCON_FUN_MP % create example files
244 end
245
246 % Load BrainAtlas
247 im_ba = ImporterBrainAtlasXLS( ...
248     'FILE', [fileparts(which('SubjectCON_FUN_MP')) filesep 'Example data
    CON_FUN_MP XLS' filesep 'atlas.xlsx'], ...
249     'WAITBAR', true ...
250 );
251
252 ba = im_ba.get('BA');
253
254 % Load Groups of SubjectCON
255 im_gr1 = ImporterGroupSubjectCON_XLS( ...
256     'DIRECTORY', [fileparts(which('SubjectCON_FUN_MP')) filesep 'Example
    data CON_FUN_MP XLS' filesep 'CON_FUN_MP_Group_1.XLS.CON'], ...
257     'BA', ba, ...
258     'WAITBAR', true ...
259 );
260
261 gr1_CON = im_gr1.get('GR');
262
263 im_gr2 = ImporterGroupSubjectCON_XLS( ...
264     'DIRECTORY', [fileparts(which('SubjectCON_FUN_MP')) filesep 'Example
    data CON_FUN_MP XLS' filesep 'CON_FUN_MP_Group_2.XLS.CON'], ...
265     'BA', ba, ...
266     'WAITBAR', true ...
267 );
268
269 gr2_CON = im_gr2.get('GR');
270
271 % Load Groups of SubjectFUN
272 im_gr1 = ImporterGroupSubjectFUN_XLS( ...
273     'DIRECTORY', [fileparts(which('SubjectCON_FUN_MP')) filesep 'Example
    data CON_FUN_MP XLS' filesep 'CON_FUN_MP_Group_1.XLS.FUN'], ...
274     'BA', ba, ...

```



```

275     'WAITBAR', true ...
276 );
277
278 gr1_FUN = im_gr1.get('GR');
279
280 im_gr2 = ImporterGroupSubjectFUN_XLS( ...
281     'DIRECTORY', [fileparts(which('SubjectCON_FUN_MP')) filesep 'Example
282     data CON_FUN_MP_XLS' filesep 'CON_FUN_MP_Group_2_XLS.FUN'], ...
283     'BA', ba, ...
284     'WAITBAR', true ...
285 );
286
287 gr2_FUN = im_gr2.get('GR');
288
289 % Combine Groups of SubjectCON with Groups of SubjectFUN
290 co_gr1 = CombineGroups_CON_FUN_MP( ...
291     'GR_CON', gr1_CON, ...
292     'GR_FUN', gr1_FUN, ...
293     'WAITBAR', true ...
294 );
295
296 gr1 = co_gr1.get('GR_CON_FUN_MP');
297
298 co_gr2 = CombineGroups_CON_FUN_MP( ...
299     'GR_CON', gr2_CON, ...
300     'GR_FUN', gr2_FUN, ...
301     'WAITBAR', true ...
302 );
303
304 gr2 = co_gr2.get('GR_CON_FUN_MP');
305
306 % Analysis CON FUN MP WU
307 a_WU1 = AnalyzeEnsemble_CON_FUN_MP_WU( ...
308     'GR', gr1 ...
309 );
310
311 a_WU2 = AnalyzeEnsemble_CON_FUN_MP_WU( ...
312     'TEMPLATE', a_WU1, ...
313     'GR', gr2 ...
314 );
315
316 % To be added the multiplex measures
317 a_WU1.get('MEASUREENSEMBLE', 'Degree').get('M');
318 a_WU1.get('MEASUREENSEMBLE', 'DegreeAv').get('M');
319 a_WU1.get('MEASUREENSEMBLE', 'Distance').get('M');
320
321 a_WU2.get('MEASUREENSEMBLE', 'Degree').get('M');
322 a_WU2.get('MEASUREENSEMBLE', 'DegreeAv').get('M');
323 a_WU2.get('MEASUREENSEMBLE', 'Distance').get('M');
324
325 % create item lists of NNDataPoint_Graph_CLA
326 [~, group_folder_name] = fileparts(im_gr1.get('DIRECTORY'));
327 it_list1 = cellfun(@(x) NNDataPoint_Measure_CLA( ...
328     'ID', x.get('ID'), ...
329     'G', x, ...
330     'M_LIST', {'Degree' 'DegreeAv' 'Distance'}, ...
331     'TARGET_IDS', {group_folder_name}), ...
332     a_WU1.get('G_DICT').get('IT_LIST'), ...
333     'UniformOutput', false);
334
335 [~, group_folder_name] = fileparts(im_gr2.get('DIRECTORY'));

```

```

335 it_list2 = cellfun(@(x) NNDataPoint_Measure_CLA( ...
336     'ID', x.get('ID'), ...
337     'G', x, ...
338     'M_LIST', {'Degree' 'DegreeAv' 'Distance'}, ...
339     'TARGET_IDS', {group_folder_name}), ...
340     a_WU2.get('G_DICT').get('IT_LIST'), ...
341     'UniformOutput', false);
342
343 % create NNDataPoint_Graph_CLA DICT items
344 dp_list1 = IndexedDictionary(...
345     'IT_CLASS', 'NNDataPoint_Measure_CLA', ...
346     'IT_LIST', it_list1 ...
347     );
348
349 dp_list2 = IndexedDictionary(...
350     'IT_CLASS', 'NNDataPoint_Measure_CLA', ...
351     'IT_LIST', it_list2 ...
352     );
353
354 % create a NNDataset containing the NNDataPoint_Measure_CLA DICT
355 d1 = NNDataset( ...
356     'DP_CLASS', 'NNDataPoint_Measure_CLA', ...
357     'DP_DICT', dp_list1 ...
358     );
359
360 d2 = NNDataset( ...
361     'DP_CLASS', 'NNDataPoint_Measure_CLA', ...
362     'DP_DICT', dp_list2 ...
363     );
364
365 % Check whether the content of input for a single datapoint matches
366 for index = 1:l:gr1.get('SUB_DICT').get('LENGTH')
367     individual_input = d1.get('DP_DICT').get('IT', index).get('INPUT');
368     known_input = cellfun(@(m) m.get('M'), a_WU1.get('G_DICT').get('IT',
369         index).get('M_DICT').get('IT_LIST'), 'UniformOutput', false);
370
371     assert(isequal(individual_input, known_input), ...
372         [BRAPH2.STR ':NNDataPoint_Measure_CLA:' BRAPH2.FAIL_TEST], ...
373         'NNDataPoint_Measure_CLA does not construct the dataset correctly.
374         The input value is not derived correctly.' ...
375         )
376 end
377
378 for index = 1:l:gr2.get('SUB_DICT').get('LENGTH')
379     individual_input = d2.get('DP_DICT').get('IT', index).get('INPUT');
380     known_input = cellfun(@(m) m.get('M'), a_WU2.get('G_DICT').get('IT',
381         index).get('M_DICT').get('IT_LIST'), 'UniformOutput', false);
382
383     assert(isequal(individual_input, known_input), ...
384         [BRAPH2.STR ':NNDataPoint_Measure_CLA:' BRAPH2.FAIL_TEST], ...
385         'NNDataPoint_Measure_CLA does not construct the dataset correctly.
386         The input value is not derived correctly.' ...
387         )
388 end
389
390 %%% itest!
391 %%% iname! ⑥
392 Example script for binary undirected graph at fixed densities (GraphBUD)
393 using connectivity data
394 %%% icode!
395 if ~isfile([fileparts(which('NNDataPoint_CON_CLA')) filesep 'Example data NN

```

⑥ test adding various kinds of graph measure with the MultigraphBUD using example connectivity data.

```

        CLA CON XLS' filesep 'atlas.xlsx'])
391 test_NNDataPoint_CON_CLA % create example files
392 end
393 example_NNCV_CON_BUD_M_CLA
394
395 %%% itest!
396 %%% iname! ⑦
397 Example script for binary undirected graph at fixed thresholds (
        MultigraphBUT) using connectivity data
398 %%% icode!
399 if ~isfile([fileparts(which('NNDataPoint_CON_CLA')) filesep 'Example data NN
        CLA CON XLS' filesep 'atlas.xlsx'])
400     test_NNDataPoint_CON_CLA % create example files
401 end
402 example_NNCV_CON_BUT_M_CLA
403
404 %%% itest!
405 %%% iname! ⑧
406 Example script for binary undirected graph at fixed densities (MultigraphBUD
        ) using connectivity data
407 %%% icode!
408 if ~isfile([fileparts(which('NNDataPoint_CON_CLA')) filesep 'Example data NN
        CLA CON XLS' filesep 'atlas.xlsx'])
409     test_NNDataPoint_CON_CLA % create example files
410 end
411 example_NNCV_CON_BUD_M_CLA
412
413 %%% itest!
414 %%% iname! ⑨
415 Example script for binary undirected multiplex at fixed densities (
        MultiplexBUD) using connectivity data and functional data
416 %%% icode!
417 if ~isfile([fileparts(which('NNDataPoint_CON_FUN_MP_CLA')) filesep 'Example
        data NN CLA CON_FUN_MP XLS' filesep 'atlas.xlsx'])
418     test_NNDataPoint_CON_FUN_MP_CLA % create example files
419 end
420 example_NNCV_CON_FUN_MP_BUD_M_CLA
421
422 %%% itest!
423 %%% iname! ⑩
424 Example script for binary undirected multiplex at fixed thresholds (
        MultiplexBUT) using connectivity data and functional data
425 %%% icode!
426 if ~isfile([fileparts(which('NNDataPoint_CON_FUN_MP_CLA')) filesep 'Example
        data NN CLA CON_FUN_MP XLS' filesep 'atlas.xlsx'])
427     test_NNDataPoint_CON_FUN_MP_CLA % create example files
428 end
429 example_NNCV_CON_FUN_MP_BUT_M_CLA

```

⑦ test adding various kinds of graph measure with the MultigraphBUT using example data.

⑧ test adding various kinds of graph measure with the MultigraphBUD using example connectivity data.

⑨ test adding various kinds of graph measure with the MultiplexBUD using example functional data.

⑩ test adding various kinds of graph measure with the MultiplexBUT using example functional data.