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

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.

```
%% iheader!
NNClassifierMLP < NNBase (nn, multi-layer perceptron classifier) comprises a
      multi-layer perceptron classifier model and a given dataset.  (1)

%%% idescription!
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.
```

(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.

```
%% iprops_update!

%%% iprop!
NAME (constant, string) is the name of the neural network multi-layer
     perceptron classifier.
%%%% idefault!
'NNClassifierMLP'

%%% iprop!
DESCRIPTION (constant, string) is the description of the neural network
     multi-layer perceptron classifier.
%%%% idefault!
'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.'

%%% iprop!
TEMPLATE (parameter, item) is the template of the neural network multi-layer
      perceptron classifier.
%%%% isettings!
'NNClassifierMLP'

%%% iprop!
ID (data, string) is a few-letter code for the neural network multi-layer
     perceptron classifier.
%%%% idefault!
'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 = 1:1: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.

```
        end
        flattened_inputs_group = [flattened_inputs_group;
     flattened_inputs_individual'];
    end
    value = {flattened_inputs_group};
end

%%% iprop!
TARGETS (query, cell) constructs the targets in the CB (channel-batch)
     format with one-hot vectors.
%%%% icalculate! (4)
% targets = nn.get('TARGETS', D) returns a cell array with the
%  targets for all data points in dataset D with one-hot vectors.
if isempty(varargin)
    value = {};
    return
end
d = varargin{1};

target_ids = nn.get('TARGET_IDS', d);
value = onehotencode(categorical(target_ids), 2);

%%% iprop!
MODEL (result, net) is a trained neural network model.
%%%% icalculate! (5)
inputs = cell2mat(nn.get('INPUTS', nn.get('D'))); (6)
targets = nn.get('TARGET_IDS', nn.get('D')); (7)
if isempty(inputs) || isempty(targets)
    value = network();
else
    number_features = size(inputs, 2);
    number_targets = size(targets, 2);
    targets = categorical(targets);
    number_classes = numel(categories(targets));

    layers = nn.get('LAYERS'); (8)
    nn_architecture = [featureInputLayer(number_features, 'Name', 'Input')];
    for i = 1:1:length(layers)
        nn_architecture = [nn_architecture
            fullyConnectedLayer(layers(i), 'Name', ['Dense_' num2str(i)])
            batchNormalizationLayer('Name', ['BatchNormalization_' num2str(i
     )])
            dropoutLayer('Name', ['Dropout_' num2str(i)])
            ];
    end
    nn_architecture = [nn_architecture
        reluLayer('Name', 'Relu_output')
        fullyConnectedLayer(number_classes, 'Name', 'Dense_output')
        softmaxLayer
        classificationLayer('Name', 'Output')
        ];

    % specify trianing options  (9)
    options = trainingOptions(nn.get('SOLVER'), ...
        'MiniBatchSize', nn.get('BATCH'), ...
        'MaxEpochs', nn.get('EPOCHS'), ...
        'Shuffle', nn.get('SHUFFLE'), ...
        'Plots', nn.get('PLOT_TRAINING'), ...
        'Verbose', nn.get('VERBOSE'));

```

(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 = 1:1: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) detemines 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!
```

```
% fi = nn.get('FEATURE_IMPORTANCE', D, P, SEED) retrieves a cell array containing
%  the feature importance values for the trained model, as assessed by
%  evaluating it on the input dataset D.
if isempty(varargin)
    value = {};
    return
end
d = varargin{1};
P = varargin{2};
seeds = varargin{3};

inputs = cell2mat(nn.get('INPUTS', d));
if isempty(inputs)
    value = {};
    return
end
targets = nn.get('TARGETS', d);
net = nn.get('MODEL');

number_features = size(inputs, 2);
original_loss = crossentropy(net.predict(inputs), targets);

wb = braph2waitbar(nn.get('WAITBAR'), 0, ['Feature importance permutation ...']);

start = tic;
for i = 1:1:P
    rng(seeds(i), 'twister')
    parfor j = 1:1:number_features
        scrambled_inputs = inputs;
        permuted_value = squeeze(normrnd(mean(inputs(:, j)), std(inputs(:, j)), squeeze(size(inputs(:, j)))) + squeeze(randn(size(inputs(:, j)))) + mean(inputs(:, j));
        scrambled_inputs(:, j) = permuted_value;
        scrambled_loss = crossentropy(net.predict(scrambled_inputs), targets);
        feature_importance(j) = scrambled_loss;
    end

    feature_importance_all_permutations{i} = feature_importance / original_loss;

    braph2waitbar(wb, i / P, ['Feature importance permutation ' num2str(i) ' of ' num2str(P) ' - ' int2str(toc(start)) '.' int2str(mod(toc(start), 1) * 10) 's ...'])
    if nn.get('VERBOSE')
        disp(['** PERMUTATION FEATURE IMPORTANCE - sampling #' int2str(i) '/' int2str(P) ' - ' int2str(toc(start)) '.' int2str(mod(toc(start), 1) * 10) 's'])
    end
    if nn.get('INTERRUPTIBLE')
        pause(nn.get('INTERRUPTIBLE'))
    end
end

braph2waitbar(wb, 'close')

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.

```
%% itests!

%%% iexcluded_props! (1)
[NNDataPoint_CON_REG.SUB]

%%% itest!
%%%% iname!
Create example files for regression (2)
%%%% icode!
data_dir = [fileparts(which('NNDataPoint_CON_REG')) filesep 'Example data NN
      REG CON XLS'];
if ~isdir(data_dir)
    mkdir(data_dir);

    % Brain Atlas (3)
    im_ba = ImporterBrainAtlasXLS('FILE', 'desikan_atlas.xlsx');
    ba = im_ba.get('BA');
    ex_ba = ExporterBrainAtlasXLS( ...
        'BA', ba, ...
        'FILE', [data_dir filesep() 'atlas.xlsx'] ...
        );
    ex_ba.get('SAVE')
    N = ba.get('BR_DICT').get('LENGTH');

    % saves RNG
    rng_settings_ = rng(); rng('default')

    sex_options = {'Female' 'Male'};

    % Group (4)
    K = 2; % degree (mean node degree is 2K)
    beta = 0.3; % Rewiring probability
    gr_name = 'CON_Group_XLS';
    gr_dir = [data_dir filesep() gr_name];
    mkdir(gr_dir);
    vois = [
        {{'Subject ID'} {'Age'} {'Sex'}}
        {{} {} cell2str(sex_options)}
        ];
    for i = 1:1:100 % subject number
        sub_id = ['SubjectCON_' num2str(i)];
        % create WS graphs with random beta
        beta(i) = rand(1); (5)
        h = WattsStrogatz(N, K, beta(i)); % create WS graph (6)

        A = full(adjacency(h)); A(1:length(A)+1:numel(A)) = 0; % extract the
      adjacency matrix
        r = 0 + (0.5 - 0) * rand(size(A)); diffA = A - r; A(A ~= 0) = diffA(
     A ~= 0); % make the adjacency matrix weighted
        A = max(A, transpose(A)); % make the adjacency matrix symmetric

        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:nnz(switchEdge));
86      end
87
88      h = graph(s,t);
89  end
90
91  %%% itest!
92  %%%% iname! (11)
93  Create a NNDataset containg 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.

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

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.

```
%% !header!
NNDataPoint_CON_CLA < NNDataPoint (dp, connectivity classification data
     point) is a data point for classification with connectivity data.

%%% !description!
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.
```

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.

```
%% !props_update!

%%% !prop!
NAME (constant, string) is the name of a data point for classification with
     connectivity data.
%%%% !default!
'NNDataPoint_CON_CLA'

%%% !prop!
DESCRIPTION (constant, string) is the description of a data point for
     classification with connectivity data.
%%%% !default!
'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.'

%%% !prop!
TEMPLATE (parameter, item) is the template of a data point for
     classification with connectivity data.
%%%% !settings!
'NNDataPoint_CON_CLA'

%%% !prop!
ID (data, string) is a few-letter code for a data point for classification
     with connectivity data.
%%%% !default!
'NNDataPoint_CON_CLA ID'

%%% !prop!
```

```
LABEL (metadata, string) is an extended label of a data point for
    classification with connectivity data.
%%%% idefault!
'NNDataPoint_CON_CLA label'

%%% iprop!
NOTES (metadata, string) are some specific notes about a data point for
    classification with connectivity data.
%%%% idefault!
'NNDataPoint_CON_CLA notes'

%%% iprop!
INPUT (result, cell) is the input value for this data point.
%%%% icalculate!
value = {dp.get('SUB').get('CON')};

%%% iprop!
TARGET (result, stringlist) is the target values for this data point.
%%%% icalculate!
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`.

```
%% iprops!

%%% iprop!
SUB (data, item) is a subject with connectivity data.
%%%% isettings!
'SubjectCON'

%%% iprop! (1)
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.

```
%% itests!

%%% iexcluded_props!
[NNDataPoint_CON_CLA.SUB]

%%% itest!
%%%% iname!
Create example files
%%%% icode!
data_dir = [fileparts(which('NNDataPoint_CON_CLA')) filesep 'Example data NN
      CLA CON XLS'];
if ~isdir(data_dir)
    mkdir(data_dir);

    % Brain Atlas
    im_ba = ImporterBrainAtlasXLS('FILE', 'desikan_atlas.xlsx');
    ba = im_ba.get('BA');
    ex_ba = ExporterBrainAtlasXLS( ...
        'BA', ba, ...
        'FILE', [data_dir filesep() 'atlas.xlsx'] ...
        );
    ex_ba.get('SAVE')
    N = ba.get('BR_DICT').get('LENGTH');

    % saves RNG
    rng_settings_ = rng(); rng('default')

    sex_options = {'Female' 'Male'};

    % Group 1  (1)
    K1 = 2; % degree (mean node degree is 2K) - group 1
    beta1 = 0.3; % Rewiring probability - group 1
    gr1_name = 'CON_Group_1_XLS';
    gr1_dir = [data_dir filesep() gr1_name];
    mkdir(gr1_dir);
    vois1 = [
        {{'Subject ID'} {'Age'} {'Sex'}}
        {{} {} cell2str(sex_options)}
        ];
    for i = 1:1:50 % subject number
        sub_id = ['SubjectCON_' num2str(i)];

        h1 = WattsStrogatz(N, K1, beta1); % create two WS graph
        % figure(1) % Plot the two graphs to double-check
        % plot(h1, 'NodeColor',[1 0 0], 'EdgeColor',[0 0 0], 'EdgeAlpha
     ',0.1, 'Layout','circle');
        % title(['Group 1: Graph with $N = $ ' num2str(N_nodes) ...
        %     ' nodes, $K = $ ' num2str(K1) ', and $\beta = $ ' num2str(
     beta1)], ...
        %     'Interpreter','latex')
        % axis equal

        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) ...
78          %     ' nodes, $K = $ ' num2str(K2) ', and $\beta = $ ' num2str(
     beta2)], ...
79          %     'Interpreter','latex')
80          % axis equal
81
82          A2 = full(adjacency(h2)); A2(1:length(A2)+1:numel(A2)) = 0;
83          r = 0 + (0.5 - 0)*rand(size(A2)); diffA = A2 - r; A2(A2 ~= 0) =
     diffA(A2 ~= 0);
84          A2 = max(A2, transpose(A2));
85
86          writetable(array2table(A2), [gr2_dir filesep() sub_id '.xlsx'], '
     WriteVariableNames', false)
87
88          % variables of interest
89          vois2 = [vois2; {sub_id, randi(90), sex_options(randi(2))}];
90      end
91      writetable(table(vois2), [data_dir filesep() gr2_name '.vois.xlsx'], '
     WriteVariableNames', false)
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.

```
    for i = 101:1:150
        sub_id = ['SubjectCON_' num2str(i)];

        h3 = WattsStrogatz(N, K3, beta3);
        % figure(2)
        % plot(h2, 'NodeColor',[1 0 0], 'EdgeColor',[0 0 0], 'EdgeAlpha',0.1, 'Layout','circle');
        % title(['Group 2: Graph with $N = $ ' num2str(N_nodes) ...
        %     ' nodes, $K = $ ' num2str(K2) ', and $\beta = $ ' num2str(beta2)], ...
        %     'Interpreter','latex')
        % axis equal

        A3 = full(adjacency(h3)); A3(1:length(A3)+1:numel(A3)) = 0;
        r = 0 + (0.5 - 0)*rand(size(A3)); diffA = A3 - r; A3(A3 ~= 0) = diffA(A3 ~= 0);
        A3 = max(A3, transpose(A3));

        writetable(array2table(A3), [gr3_dir filesep() sub_id '.xlsx'], 'WriteVariableNames', false)

        % variables of interest
        vois3 = [vois3; {sub_id, randi(90), sex_options(randi(2))}];
    end
    writetable(table(vois3), [data_dir filesep() gr3_name '.vois.xlsx'], 'WriteVariableNames', false)

    % reset RNG
    rng(rng_settings_)
end

%%% itest_functions!
function h = WattsStrogatz(N,K,beta)
% H = WattsStrogatz(N,K,beta) returns a Watts-Strogatz model graph with N
% nodes, N*K edges, mean node degree 2*K, and rewiring probability beta.
%
% beta = 0 is a ring lattice, and beta = 1 is a random graph.

% Connect each node to its K next and previous neighbors. This constructs
% indices for a ring lattice.
s = repelem((1:N)',1,K);
t = s + repmat(1:K,N,1);
t = mod(t-1,N)+1;

% Rewire the target node of each edge with probability beta
for source=1:N
    switchEdge = rand(K, 1) < beta;

    newTargets = rand(N, 1);
    newTargets(source) = 0;
    newTargets(s(t==source)) = 0;
    newTargets(t(source, ~switchEdge)) = 0;

    [~, ind] = sort(newTargets, 'descend');
    t(source, switchEdge) = ind(1:nnz(switchEdge));
end

h = graph(s,t);
end

%%% itest!
```

```
%%%% iname!
Create a NNDataset containg NNDataPoint_CON_CLA with simulated data
%%%% icode!
% Load BrainAtlas
im_ba = ImporterBrainAtlasXLS( ...
    'FILE', [fileparts(which('NNDataPoint_CON_CLA')) filesep 'Example data
     NN CLA CON XLS' filesep 'atlas.xlsx'], ...
    'WAITBAR', true ...
    );

ba = im_ba.get('BA');

% Load Groups of SubjectCON (6)
im_gr1 = ImporterGroupSubjectCON_XLS( ...
    'DIRECTORY', [fileparts(which('NNDataPoint_CON_CLA')) filesep 'Example
     data NN CLA CON XLS' filesep 'CON_Group_1_XLS'], ...
    'BA', ba, ...
    'WAITBAR', true ...
    );

gr1 = im_gr1.get('GR');

im_gr2 = ImporterGroupSubjectCON_XLS( ...
    'DIRECTORY', [fileparts(which('NNDataPoint_CON_CLA')) filesep 'Example
     data NN CLA CON XLS' filesep 'CON_Group_2_XLS'], ...
    'BA', ba, ...
    'WAITBAR', true ...
    );

gr2 = im_gr2.get('GR');

% create item lists of NNDataPoint_CON_CLA (7)
[~, group_folder_name] = fileparts(im_gr1.get('DIRECTORY'));
it_list1 = cellfun(@(x) NNDataPoint_CON_CLA( ...
    'ID', x.get('ID'), ...
    'SUB', x, ...
    'TARGET_IDS', {group_folder_name}), ...
    gr1.get('SUB_DICT').get('IT_LIST'), ...
    'UniformOutput', false);

[~, group_folder_name] = fileparts(im_gr2.get('DIRECTORY'));
it_list2 = cellfun(@(x) NNDataPoint_CON_CLA( ...
    'ID', x.get('ID'), ...
    'SUB', x, ...
    'TARGET_IDS', {group_folder_name}), ...
    gr2.get('SUB_DICT').get('IT_LIST'), ...
    'UniformOutput', false);

% create NNDataPoint_CON_CLA DICT items
dp_list1 = IndexedDictionary(...
        'IT_CLASS', 'NNDataPoint_CON_CLA', ...
        'IT_LIST', it_list1 ...
        );

dp_list2 = IndexedDictionary(...
        'IT_CLASS', 'NNDataPoint_CON_CLA', ...
        'IT_LIST', it_list2 ...
        );

% create a NNDataset containing the NNDataPoint_CON_CLA DICT
d1 = NNDataset( ...
```

(6) imports two groups of simulated data.

(7) creates two datasets for the two groups.

```
    'DP_CLASS', 'NNDataPoint_CON_CLA', ...
    'DP_DICT', dp_list1 ...
    );

d2 = NNDataset( ...
    'DP_CLASS', 'NNDataPoint_CON_CLA', ...
    'DP_DICT', dp_list2 ...
    );

% Check whether the number of inputs matches (8)
assert(length(d1.get('INPUTS')) == gr1.get('SUB_DICT').get('LENGTH'), ...
    [BRAPH2.STR ':NNDataPoint_CON_CLA:' BRAPH2.FAIL_TEST], ...
    'NNDataPoint_CON_CLA does not construct the dataset correctly. The
     number of the inputs should be the same as the number of imported
     subjects of group 1.' ...
    )

assert(length(d2.get('INPUTS')) == gr2.get('SUB_DICT').get('LENGTH'), ...
    [BRAPH2.STR ':NNDataPoint_CON_CLA:' BRAPH2.FAIL_TEST], ...
    'NNDataPoint_CON_CLA does not construct the dataset correctly. The
     number of the inputs should be the same as the number of imported
     subjects of group 2.' ...
    )

% Check whether the number of targets matches (9)
assert(length(d1.get('TARGETS')) == gr1.get('SUB_DICT').get('LENGTH'), ...
    [BRAPH2.STR ':NNDataPoint_CON_CLA:' BRAPH2.FAIL_TEST], ...
    'NNDataPoint_CON_CLA does not construct the dataset correctly. The
     number of the targets should be the same as the number of imported
     subjects of group 1.' ...
    )

assert(length(d2.get('TARGETS')) == gr2.get('SUB_DICT').get('LENGTH'), ...
    [BRAPH2.STR ':NNDataPoint_CON_CLA:' BRAPH2.FAIL_TEST], ...
    'NNDataPoint_CON_CLA does not construct the dataset correctly. The
     number of the targets should be the same as the number of imported
     subjects of group 2.' ...
    )

% Check whether the content of input for a single datapoint matches (10)
for index = 1:1:gr1.get('SUB_DICT').get('LENGTH')
    individual_input = d1.get('DP_DICT').get('IT', index).get('INPUT');
    known_input = {gr1.get('SUB_DICT').get('IT', index).get('CON')};

    assert(isequal(individual_input, known_input), ...
        [BRAPH2.STR ':NNDataPoint_CON_CLA:' BRAPH2.FAIL_TEST], ...
        'NNDataPoint_CON_CLA does not construct the dataset correctly. The
         input value is not derived correctly.' ...
        )
end

for index = 1:1:gr2.get('SUB_DICT').get('LENGTH')
    individual_input = d2.get('DP_DICT').get('IT', index).get('INPUT');
    known_input = {gr2.get('SUB_DICT').get('IT', index).get('CON')};

    assert(isequal(individual_input, known_input), ...
        [BRAPH2.STR ':NNDataPoint_CON_CLA:' BRAPH2.FAIL_TEST], ...
        'NNDataPoint_CON_CLA does not construct the dataset correctly. The
         input value is not derived correctly.' ...
        )
```

(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.

```
%% iheader!
NNDataPoint_Graph_REG < NNDataPoint (dp, measure regressioni data point) is
     a data point for regression with a graph.

%%% idescription!
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.
```

Code 10: **NNDataPoint_Graph_REG element prop update.** 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.

```
%% iprops_update!

%%% iprop!
NAME (constant, string) is the name of a data point for regression with a
     graph.
%%%% idefault!
'NNDataPoint_Graph_REG'

%%% iprop!
DESCRIPTION (constant, string) is the description of a data point for
     regression with a graph.
%%%% idefault!
'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.'

%%% iprop!
TEMPLATE (parameter, item) is the template of a data point for regression
     with a graph.
%%%% isettings!
'NNDataPoint_Graph_REG'

%%% iprop!
```

```
ID (data, string) is a few-letter code for a data point for regression with a graph.
%%%% idefault!
'NNDataPoint_Graph_REG ID'

%%% iprop!
LABEL (metadata, string) is an extended label of a data point for regression with a graph.
%%%% idefault!
'NNDataPoint_Graph_REG label'

%%% iprop!
NOTES (metadata, string) are some specific notes about a data point for regression with a graph.
%%%% idefault!
'NNDataPoint_Graph_REG notes'

%%% iprop!
INPUT (result, cell) is the input value for this data point.
%%%% icalculate!
value = dp.get('G').get('A');  (1)

%%% iprop!
TARGET (result, cell) is the target value for this data point.
%%%% icalculate!
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`.

```
%% iprops!

%%% iprop!  (1)
G (data, item) is a graph.
%%%% isettings!
'Graph'

%%% iprop!
SUB (data, item) is a subject.
%%%% isettings!
'Subject'

%%% iprop!
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.

```
%% itests!

%%% iexcluded_props!
[NNDataPoint_Graph_REG.G NNDataPoint_Graph_REG.SUB]

%%% itest!
%%%% iname! (1)
Construct the data point with the adjacency matrix derived from its weighted
      undirected graph (GraphWU)
%%%% icode!
% ensure the example data is generated
if ~isfile([fileparts(which('NNDataPoint_CON_REG')) filesep 'Example data NN
      REG CON XLS' filesep 'atlas.xlsx'])
    test_NNDataPoint_CON_REG % create example files
end

% Load BrainAtlas
im_ba = ImporterBrainAtlasXLS( ...
    'FILE', [fileparts(which('NNDataPoint_CON_REG')) filesep 'Example data
     NN REG CON XLS' filesep 'atlas.xlsx'], ...
    'WAITBAR', true ...
    );

ba = im_ba.get('BA');

% Load Groups of SubjectCON
im_gr = ImporterGroupSubjectCON_XLS( ...
    'DIRECTORY', [fileparts(which('NNDataPoint_CON_REG')) filesep 'Example
     data NN REG CON XLS' filesep 'CON_Group_XLS'], ...
    'BA', ba, ...
    'WAITBAR', true ...
    );

gr = im_gr.get('GR');

% Analysis CON WU (2)
a_WU = AnalyzeEnsemble_CON_WU( ...
    'GR', gr ...
    );

a_WU.memorize('G_DICT'); (3)

% create item lists of NNDataPoint_Graph_REG (4)
it_list = cellfun(@(g, sub) NNDataPoint_Graph_REG( ...
    'ID', sub.get('ID'), ...
    'G', g, ...
    'SUB', sub, ...
    'TARGET_IDS', sub.get('VOI_DICT').get('KEYS')), ...
    a_WU.get('G_DICT').get('IT_LIST'), gr.get('SUB_DICT').get('IT_LIST'),...
    'UniformOutput', false);

% create NNDataPoint_Graph_REG DICT items
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`.

```
        'IT_CLASS', 'NNDataPoint_Graph_REG', ...
        'IT_LIST', it_list ...
        );

% create a NNDataset containing the NNDataPoint_Graph_REG DICT
d = NNDataset( ...
    'DP_CLASS', 'NNDataPoint_Graph_REG', ...
    'DP_DICT', dp_list ...
    );

% Check whether the content of input for a single datapoint matches (5)
for index = 1:1:gr.get('SUB_DICT').get('LENGTH')
    individual_input = d.get('DP_DICT').get('IT', index).get('INPUT');
    known_input = a_WU.get('G_DICT').get('IT', index).get('A');

    assert(isequal(individual_input, known_input), ...
        [BRAPH2.STR ':NNDataPoint_Graph_REG:' BRAPH2.FAIL_TEST], ...
        'NNDataPoint_Graph_REG does not construct the dataset correctly. The
      input value is not derived correctly.' ...
        )
end

%%% itest!
%%%% iname!(6)
Construct the data point with the adjacency matrix derived from its binary
     undirected multigraph with fixed densities (MultigraphBUD)
%%%% icode!
% ensure the example data is generated
if ~isfile([fileparts(which('NNDataPoint_CON_REG')) filesep 'Example data NN
      REG CON XLS' filesep 'atlas.xlsx'])
    test_NNDataPoint_CON_REG % create example files
end

% Load BrainAtlas
im_ba = ImporterBrainAtlasXLS( ...
    'FILE', [fileparts(which('NNDataPoint_CON_REG')) filesep 'Example data
     NN REG CON XLS' filesep 'atlas.xlsx'], ...
    'WAITBAR', true ...
    );

ba = im_ba.get('BA');

% Load Groups of SubjectCON
im_gr = ImporterGroupSubjectCON_XLS( ...
    'DIRECTORY', [fileparts(which('NNDataPoint_CON_REG')) filesep 'Example
     data NN REG CON XLS' filesep 'CON_Group_XLS'], ...
    'BA', ba, ...
    'WAITBAR', true ...
    );

gr = im_gr.get('GR');

% Analysis CON WU
densities = 0:25:100;

a_BUD = AnalyzeEnsemble_CON_BUD( ...
    'DENSITIES', densities, ...
    'GR', gr ...
    );

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

(7) tests with the `MultiplexWU` element which contains the adjacency matrix of weighted undirected multipex.

```
    );

gr_CON = im_gr.get('GR');

% Load Groups of SubjectFUN
im_gr = ImporterGroupSubjectFUN_XLS( ...
    'DIRECTORY', [fileparts(which('SubjectCON_FUN_MP')) filesep 'Example
     data CON_FUN_MP XLS' filesep 'CON_FUN_MP_Group_1_XLS.FUN'], ...
    'BA', ba, ...
    'WAITBAR', true ...
    );

gr_FUN = im_gr.get('GR');

% Combine Groups of SubjectCON with Groups of SubjectFUN
co_gr = CombineGroups_CON_FUN_MP( ...
    'GR_CON', gr_CON, ...
    'GR_FUN', gr_FUN, ...
    'WAITBAR', true ...
    );

gr = co_gr.get('GR_CON_FUN_MP');

% Analysis CON FUN MP WU
a_WU = AnalyzeEnsemble_CON_FUN_MP_WU( ...
    'GR', gr ...
    );

a_WU.memorize('G_DICT');

% create item lists of NNDataPoint_Graph_REG
it_list = cellfun(@(g, sub) NNDataPoint_Graph_REG( ...
    'ID', sub.get('ID'), ...
    'G', g, ...
    'SUB', sub, ...
    'TARGET_IDS', sub.get('VOI_DICT').get('KEYS')), ...
    a_WU.get('G_DICT').get('IT_LIST'), gr.get('SUB_DICT').get('IT_LIST'),...
    'UniformOutput', false);

% create NNDataPoint_Graph_REG DICT items
dp_list = IndexedDictionary(...
        'IT_CLASS', 'NNDataPoint_Graph_REG', ...
        'IT_LIST', it_list ...
        );

% create a NNDataset containing the NNDataPoint_Graph_REG DICT
d = NNDataset( ...
    'DP_CLASS', 'NNDataPoint_Graph_REG', ...
    'DP_DICT', dp_list ...
    );

% Check whether the content of input for a single datapoint matches
for index = 1:1:gr.get('SUB_DICT').get('LENGTH')
    individual_input = d.get('DP_DICT').get('IT', index).get('INPUT');
    known_input = a_WU.get('G_DICT').get('IT', index).get('A');

    assert(isequal(individual_input, known_input), ...
        [BRAPH2.STR ':NNDataPoint_Graph_REG:' BRAPH2.FAIL_TEST], ...
        'NNDataPoint_Graph_REG does not construct the dataset correctly. The
       input value is not derived correctly.' ...
        )
```

```
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.

```
%% iheader!
NNDataPoint_Graph_CLA < NNDataPoint (dp, graph classification data point) is
      a data point for classification with a graph.

%%% idescription!
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.
```

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.

```
%% iprops_update!

%%% iprop!
NAME (constant, string) is the name of a data point for classification with
     a graph.
%%%% idefault!
'NNDataPoint_Graph_CLA'

%%% iprop!
DESCRIPTION (constant, string) is the description of a data point for
     classification with a graph.
%%%% idefault!
'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.'

%%% iprop!
TEMPLATE (parameter, item) is the template of a data point for
     classification with a graph.
%%%% isettings!
'NNDataPoint_Graph_CLA'

%%% iprop!
ID (data, string) is a few-letter code for a data point for classification
     with a graph.
%%%% idefault!
```

```
'NNDataPoint_Graph_CLA ID'

%%% iprop!
LABEL (metadata, string) is an extended label of a data point for
     classification with a graph.
%%%% idefault!
'NNDataPoint_Graph_CLA label'

%%% iprop!
NOTES (metadata, string) are some specific notes about a data point for
     classification with a graph.
%%%% idefault!
'NNDataPoint_Graph_CLA notes'

%%% iprop!
INPUT (result, cell) is the input value for this data point.
%%%% icalculate!
value = dp.get('G').get('A');  (1)

%%% iprop!
TARGET (result, cell) is the target value for this data point.
%%%% icalculate!
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`.

```
%% iprops!

%%% iprop!  (1)
G (data, item) is a graph.
%%%% isettings!
'Graph'

%%% iprop!
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.

```
%% itests!

%%% iexcluded_props!
[NNDataPoint_Graph_CLA.G]

%%% itest!
%%%% iname! (1)
Construct the data point with the adjacency matrix derived from its weighted
      undirected graph (GraphWU)
%%%% icode!
% ensure the example data is generated
if ~isfile([fileparts(which('NNDataPoint_CON_CLA')) filesep 'Example data NN
      CLA CON XLS' filesep 'atlas.xlsx'])
    test_NNDataPoint_CON_CLA % create example files
end

% Load BrainAtlas
im_ba = ImporterBrainAtlasXLS( ...
    'FILE', [fileparts(which('NNDataPoint_CON_CLA')) filesep 'Example data
     NN CLA CON XLS' filesep 'atlas.xlsx'], ...
    'WAITBAR', true ...
    );

ba = im_ba.get('BA');

% Load Groups of SubjectCON
im_gr1 = ImporterGroupSubjectCON_XLS( ...
    'DIRECTORY', [fileparts(which('NNDataPoint_CON_CLA')) filesep 'Example
     data NN CLA CON XLS' filesep 'CON_Group_1_XLS'], ...
    'BA', ba, ...
    'WAITBAR', true ...
    );

gr1 = im_gr1.get('GR');

im_gr2 = ImporterGroupSubjectCON_XLS( ...
    'DIRECTORY', [fileparts(which('NNDataPoint_CON_CLA')) filesep 'Example
     data NN CLA CON XLS' filesep 'CON_Group_2_XLS'], ...
    'BA', ba, ...
    'WAITBAR', true ...
    );

gr2 = im_gr2.get('GR');

% Analysis CON WU
a_WU1 = AnalyzeEnsemble_CON_WU( ...
    'GR', gr1 ...
    );

a_WU2 = AnalyzeEnsemble_CON_WU( ...
    'TEMPLATE', a_WU1, ...
    'GR', gr2 ...
    );

```

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

```
a_WU1.memorize('G_DICT');
a_WU2.memorize('G_DICT');

% create item lists of NNDataPoint_Graph_CLA
[~, group_folder_name] = fileparts(im_gr1.get('DIRECTORY'));
it_list1 = cellfun(@(x) NNDataPoint_Graph_CLA( ...
    'ID', x.get('ID'), ...
    'G', x, ...
    'TARGET_IDS', {group_folder_name}), ...
    a_WU1.get('G_DICT').get('IT_LIST'), ...
    'UniformOutput', false);

[~, group_folder_name] = fileparts(im_gr2.get('DIRECTORY'));
it_list2 = cellfun(@(x) NNDataPoint_Graph_CLA( ...
    'ID', x.get('ID'), ...
    'G', x, ...
    'TARGET_IDS', {group_folder_name}), ...
    a_WU2.get('G_DICT').get('IT_LIST'), ...
    'UniformOutput', false);

% create NNDataPoint_Graph_CLA DICT items
dp_list1 = IndexedDictionary(...
        'IT_CLASS', 'NNDataPoint_Graph_CLA', ...
        'IT_LIST', it_list1 ...
        );

dp_list2 = IndexedDictionary(...
        'IT_CLASS', 'NNDataPoint_Graph_CLA', ...
        'IT_LIST', it_list2 ...
        );

% create a NNDataset containing the NNDataPoint_Graph_CLA DICT
d1 = NNDataset( ...
    'DP_CLASS', 'NNDataPoint_Graph_CLA', ...
    'DP_DICT', dp_list1 ...
    );

d2 = NNDataset( ...
    'DP_CLASS', 'NNDataPoint_Graph_CLA', ...
    'DP_DICT', dp_list2 ...
    );

% Check whether the content of input for a single datapoint matches
for index = 1:1:gr1.get('SUB_DICT').get('LENGTH')
    individual_input = d1.get('DP_DICT').get('IT', index).get('INPUT');
    known_input = a_WU1.get('G_DICT').get('IT', index).get('A');

    assert(isequal(individual_input, known_input), ...
        [BRAPH2.STR ':NNDataPoint_Graph_CLA:' BRAPH2.FAIL_TEST], ...
        'NNDataPoint_Graph_CLA does not construct the dataset correctly. The
      input value is not derived correctly.' ...
        )
end

for index = 1:1:gr2.get('SUB_DICT').get('LENGTH')
    individual_input = d2.get('DP_DICT').get('IT', index).get('INPUT');
    known_input = a_WU2.get('G_DICT').get('IT', index).get('A');

    assert(isequal(individual_input, known_input), ...
        [BRAPH2.STR ':NNDataPoint_Graph_CLA:' BRAPH2.FAIL_TEST], ...
        'NNDataPoint_Graph_CLA does not construct the dataset correctly. The
```

```
        input value is not derived correctly.' ...
110       )
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.

```
it_list1 = cellfun(@(x) NNDataPoint_Graph_CLA( ...
    'ID', x.get('ID'), ...
    'G', x, ...
    'TARGET_IDS', {group_folder_name}), ...
    a_BUD1.get('G_DICT').get('IT_LIST'), ...
    'UniformOutput', false);

[~, group_folder_name] = fileparts(im_gr2.get('DIRECTORY'));
it_list2 = cellfun(@(x) NNDataPoint_Graph_CLA( ...
    'ID', x.get('ID'), ...
    'G', x, ...
    'TARGET_IDS', {group_folder_name}), ...
    a_BUD2.get('G_DICT').get('IT_LIST'), ...
    'UniformOutput', false);

% create NNDataPoint_Graph_CLA DICT items
dp_list1 = IndexedDictionary(...
        'IT_CLASS', 'NNDataPoint_Graph_CLA', ...
        'IT_LIST', it_list1 ...
        );

dp_list2 = IndexedDictionary(...
        'IT_CLASS', 'NNDataPoint_Graph_CLA', ...
        'IT_LIST', it_list2 ...
        );

% create a NNDataset containing the NNDataPoint_Graph_CLA DICT
d1 = NNDataset( ...
    'DP_CLASS', 'NNDataPoint_Graph_CLA', ...
    'DP_DICT', dp_list1 ...
    );

d2 = NNDataset( ...
    'DP_CLASS', 'NNDataPoint_Graph_CLA', ...
    'DP_DICT', dp_list2 ...
    );

% Check whether the content of input for a single datapoint matches
for index = 1:1:gr1.get('SUB_DICT').get('LENGTH')
    individual_input = d1.get('DP_DICT').get('IT', index).get('INPUT');
    known_input = a_BUD1.get('G_DICT').get('IT', index).get('A');

    assert(isequal(individual_input, known_input), ...
        [BRAPH2.STR ':NNDataPoint_Graph_CLA:' BRAPH2.FAIL_TEST], ...
        'NNDataPoint_Graph_CLA does not construct the dataset correctly. The
      input value is not derived correctly.' ...
        )
end

for index = 1:1:gr2.get('SUB_DICT').get('LENGTH')
    individual_input = d2.get('DP_DICT').get('IT', index).get('INPUT');
    known_input = a_BUD2.get('G_DICT').get('IT', index).get('A');

    assert(isequal(individual_input, known_input), ...
        [BRAPH2.STR ':NNDataPoint_Graph_CLA:' BRAPH2.FAIL_TEST], ...
        'NNDataPoint_Graph_CLA does not construct the dataset correctly. The
      input value is not derived correctly.' ...
        )
end

%%% itest!
```

```
%%%% iname! (3)
Construct the data point with the adjacency matrix derived from its multiplex weighted undirected graph (MultiplexWU)
%%%% icode!
% ensure the example data is generated
if ~isfile([fileparts(which('SubjectCON_FUN_MP')) filesep 'Example data CON_FUN_MP XLS' filesep 'atlas.xlsx'])
    test_SubjectCON_FUN_MP % create example files
end

% Load BrainAtlas
im_ba = ImporterBrainAtlasXLS( ...
    'FILE', [fileparts(which('SubjectCON_FUN_MP')) filesep 'Example data CON_FUN_MP XLS' filesep 'atlas.xlsx'], ...
    'WAITBAR', true ...
    );

ba = im_ba.get('BA');

% Load Groups of SubjectCON
im_gr1 = ImporterGroupSubjectCON_XLS( ...
    'DIRECTORY', [fileparts(which('SubjectCON_FUN_MP')) filesep 'Example data CON_FUN_MP XLS' filesep 'CON_FUN_MP_Group_1_XLS.CON'], ...
    'BA', ba, ...
    'WAITBAR', true ...
    );

gr1_CON = im_gr1.get('GR');

im_gr2 = ImporterGroupSubjectCON_XLS( ...
    'DIRECTORY', [fileparts(which('SubjectCON_FUN_MP')) filesep 'Example data CON_FUN_MP XLS' filesep 'CON_FUN_MP_Group_2_XLS.CON'], ...
    'BA', ba, ...
    'WAITBAR', true ...
    );

gr2_CON = im_gr2.get('GR');

% Load Groups of SubjectFUN
im_gr1 = ImporterGroupSubjectFUN_XLS( ...
    'DIRECTORY', [fileparts(which('SubjectCON_FUN_MP')) filesep 'Example data CON_FUN_MP XLS' filesep 'CON_FUN_MP_Group_1_XLS.FUN'], ...
    'BA', ba, ...
    'WAITBAR', true ...
    );

gr1_FUN = im_gr1.get('GR');

im_gr2 = ImporterGroupSubjectFUN_XLS( ...
    'DIRECTORY', [fileparts(which('SubjectCON_FUN_MP')) filesep 'Example data CON_FUN_MP XLS' filesep 'CON_FUN_MP_Group_2_XLS.FUN'], ...
    'BA', ba, ...
    'WAITBAR', true ...
    );

gr2_FUN = im_gr2.get('GR');

% Combine Groups of SubjectCON with Groups of SubjectFUN
co_gr1 = CombineGroups_CON_FUN_MP( ...
    'GR_CON', gr1_CON, ...
    'GR_FUN', gr1_FUN, ...
```

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

```
    'WAITBAR', true ...
    );

gr1 = co_gr1.get('GR_CON_FUN_MP');

co_gr2 = CombineGroups_CON_FUN_MP( ...
    'GR_CON', gr2_CON, ...
    'GR_FUN', gr2_FUN, ...
    'WAITBAR', true ...
    );

gr2 = co_gr2.get('GR_CON_FUN_MP');

% Analysis CON FUN MP WU
a_WU1 = AnalyzeEnsemble_CON_FUN_MP_WU( ...
    'GR', gr1 ...
    );

a_WU2 = AnalyzeEnsemble_CON_FUN_MP_WU( ...
    'TEMPLATE', a_WU1, ...
    'GR', gr2 ...
    );

a_WU1.memorize('G_DICT');
a_WU2.memorize('G_DICT');

% create item lists of NNDataPoint_Graph_CLA
[~, group_folder_name] = fileparts(im_gr1.get('DIRECTORY'));
it_list1 = cellfun(@(x) NNDataPoint_Graph_CLA( ...
    'ID', x.get('ID'), ...
    'G', x, ...
    'TARGET_IDS', {group_folder_name}), ...
    a_WU1.get('G_DICT').get('IT_LIST'), ...
    'UniformOutput', false);

[~, group_folder_name] = fileparts(im_gr2.get('DIRECTORY'));
it_list2 = cellfun(@(x) NNDataPoint_Graph_CLA( ...
    'ID', x.get('ID'), ...
    'G', x, ...
    'TARGET_IDS', {group_folder_name}), ...
    a_WU2.get('G_DICT').get('IT_LIST'), ...
    'UniformOutput', false);

% create NNDataPoint_Graph_CLA DICT items
dp_list1 = IndexedDictionary(...
        'IT_CLASS', 'NNDataPoint_Graph_CLA', ...
        'IT_LIST', it_list1 ...
        );

dp_list2 = IndexedDictionary(...
        'IT_CLASS', 'NNDataPoint_Graph_CLA', ...
        'IT_LIST', it_list2 ...
        );

% create a NNDataset containing the NNDataPoint_Graph_CLA DICT
d1 = NNDataset( ...
    'DP_CLASS', 'NNDataPoint_Graph_CLA', ...
    'DP_DICT', dp_list1 ...
    );

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
          input value is not derived correctly.' ...
351         )
352 end
353 
354 for index = 1:1:gr2.get('SUB_DICT').get('LENGTH')
355     individual_input = d2.get('DP_DICT').get('IT', index).get('INPUT');
356     known_input = a_WU2.get('G_DICT').get('IT', index).get('A');
357 
358     assert(isequal(individual_input, known_input), ...
359         [BRAPH2.STR ':NNDataPoint_Graph_CLA:' BRAPH2.FAIL_TEST], ...
360         'NNDataPoint_Graph_CLA does not construct the dataset correctly. The
          input value is not derived correctly.' ...
361         )
362 end
363 
364 %%% itest!
365 %%%% iname! (4)
366 Example script for weighted undirected graph (GraphWU) using connectivity
        data
367 %%%% icode!
368 if ~isfile([fileparts(which('NNDataPoint_CON_CLA')) filesep 'Example data NN
         CLA CON XLS' filesep 'atlas.xlsx'])
369     test_NNDataPoint_CON_CLA % create example files
370 end
371 example_NNCV_CON_WU_CLA
372 
373 %%% itest!
374 %%%% iname! (5)
375 Example script for binary undirected graph at fixed densities (MultigraphBUD
        ) using connectivity data
376 %%%% icode!
377 if ~isfile([fileparts(which('NNDataPoint_CON_CLA')) filesep 'Example data NN
         CLA CON XLS' filesep 'atlas.xlsx'])
378     test_NNDataPoint_CON_CLA % create example files
379 end
380 example_NNCV_CON_BUD_CLA
381 
382 %%% itest!
383 %%%% iname! (6)
384 Example script for weighted undirected multiplex (MultiplexWU) using
        connectivity data and functional data
385 %%%% icode!
386 if ~isfile([fileparts(which('NNDataPoint_CON_FUN_MP_CLA')) filesep 'Example
        data NN CLA CON_FUN_MP XLS' filesep 'atlas.xlsx'])
387     test_NNDataPoint_CON_FUN_MP_CLA % create example files
388 end
389 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 obtrained 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.

```
%% iheader!
NNDataPoint_Measure_REG < NNDataPoint (dp, measure regression data point)
     is a data point for regression with graph measures.

%%% idescription!
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.
```

Code 18: **NNDataPoint_Measure_REG element prop update.** 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.

```
%% iprops_update!

%%% iprop!
NAME (constant, string) is the name of a data point for regression with
     graph measures.
%%%% idefault!
'NNDataPoint_Measure_REG'

%%% iprop!
DESCRIPTION (constant, string) is the description of a data point for
     regression with graph measures.
%%%% idefault!
'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.'

%%% iprop!
TEMPLATE (parameter, item) is the template of a data point for regression
     with graph measures.
%%%% isettings!
'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.

```
%% itests!

%%% iexcluded_props!
[NNDataPoint_Measure_REG.G NNDataPoint_Measure_REG.SUB]

%%% itest!
%%%% iname! (1)
Construct the data point with the adjacency matrix derived from its weighted
      undirected graph (GraphWU)
%%%% icode!
% ensure the example data is generated
if ~isfile([fileparts(which('NNDataPoint_CON_REG')) filesep 'Example data NN
      REG CON XLS' filesep 'atlas.xlsx'])
    test_NNDataPoint_CON_REG % create example files
end

% Load BrainAtlas
im_ba = ImporterBrainAtlasXLS( ...
    'FILE', [fileparts(which('NNDataPoint_CON_REG')) filesep 'Example data
     NN REG CON XLS' filesep 'atlas.xlsx'], ...
    'WAITBAR', true ...
    );

ba = im_ba.get('BA');

% Load Groups of SubjectCON
im_gr = ImporterGroupSubjectCON_XLS( ...
    'DIRECTORY', [fileparts(which('NNDataPoint_CON_REG')) filesep 'Example
     data NN REG CON XLS' filesep 'CON_Group_XLS'], ...
    'BA', ba, ...
    'WAITBAR', true ...
    );

gr = im_gr.get('GR');

% Analysis CON WU
a_WU = AnalyzeEnsemble_CON_WU( ...
    'GR', gr ...
    );

a_WU.get('MEASUREENSEMBLE', 'Degree').get('M');  (2)
a_WU.get('MEASUREENSEMBLE', 'DegreeAv').get('M'); (3)
a_WU.get('MEASUREENSEMBLE', 'Distance').get('M'); (4)

% create item lists of NNDataPoint_Measure_REG
it_list = cellfun(@(g, sub) NNDataPoint_Measure_REG( ...
    'ID', sub.get('ID'), ...
    'G', g, ...
    'M_LIST', {'Degree' 'DegreeAv' 'Distance'}, ...
    'SUB', sub, ...
    'TARGET_IDS', sub.get('VOI_DICT').get('KEYS')), ...
    a_WU.get('G_DICT').get('IT_LIST'), gr.get('SUB_DICT').get('IT_LIST'),...
    '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',
       index).get('M_DICT').get('IT_LIST'), 'UniformOutput', false);
67
68     assert(isequal(individual_input, known_input), ...
69         [BRAPH2.STR ':NNDataPoint_Measure_REG:' BRAPH2.FAIL_TEST], ...
70         'NNDataPoint_Measure_REG does not construct the dataset correctly.
       The input value is not derived correctly.' ...
71         )
72 end
73
74 %%% itest!
75 %%%% iname! (5)
76 Construct the data point with the adjacency matrix derived from its binary
       undirected multigraph with fixed densities (MultigraphBUD)
77 %%%% icode!
78 % ensure the example data is generated
79 if ~isfile([fileparts(which('NNDataPoint_CON_REG')) filesep 'Example data NN
       REG CON XLS' filesep 'atlas.xlsx'])
80     test_NNDataPoint_CON_REG % create example files
81 end
82
83 % Load BrainAtlas
84 im_ba = ImporterBrainAtlasXLS( ...
85     'FILE', [fileparts(which('NNDataPoint_CON_REG')) filesep 'Example data
       NN REG CON XLS' filesep 'atlas.xlsx'], ...
86     'WAITBAR', true ...
87     );
88
89 ba = im_ba.get('BA');
90
91 % Load Groups of SubjectCON
92 im_gr = ImporterGroupSubjectCON_XLS( ...
93     'DIRECTORY', [fileparts(which('NNDataPoint_CON_REG')) filesep 'Example
       data NN REG CON XLS' filesep 'CON_Group_XLS'], ...
94     'BA', ba, ...
95     'WAITBAR', true ...
96     );
97
98 gr = im_gr.get('GR');
99
100 % Analysis CON WU
101 densities = 0:25:100;
102
103 a_BUD = AnalyzeEnsemble_CON_BUD( ...
104     'DENSITIES', densities, ...
```

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

```
    'GR', gr ...
    );

a_BUD.get('MEASUREENSEMBLE', 'Degree').get('M');
a_BUD.get('MEASUREENSEMBLE', 'DegreeAv').get('M');
a_BUD.get('MEASUREENSEMBLE', 'Distance').get('M');

% create item lists of NNDataPoint_Measure_REG
it_list = cellfun(@(g, sub) NNDataPoint_Measure_REG( ...
    'ID', sub.get('ID'), ...
    'G', g, ...
    'M_LIST', {'Degree' 'DegreeAv' 'Distance'}, ...
    'SUB', sub, ...
    'TARGET_IDS', sub.get('VOI_DICT').get('KEYS')), ...
    a_BUD.get('G_DICT').get('IT_LIST'), gr.get('SUB_DICT').get('IT_LIST')
     ,...
    'UniformOutput', false);

% create NNDataPoint_CON_CLA DICT items
dp_list = IndexedDictionary(...
        'IT_CLASS', 'NNDataPoint_Measure_REG', ...
        'IT_LIST', it_list ...
        );

% create a NNDataset containing the NNDataPoint_Measure_REG DICT
d = NNDataset( ...
    'DP_CLASS', 'NNDataPoint_Measure_REG', ...
    'DP_DICT', dp_list ...
    );

% Check whether the content of input for a single datapoint matches
for index = 1:1:gr.get('SUB_DICT').get('LENGTH')
    individual_input = d.get('DP_DICT').get('IT', index).get('INPUT');
    known_input = cellfun(@(m) m.get('M'), a_BUD.get('G_DICT').get('IT',
     index).get('M_DICT').get('IT_LIST'), 'UniformOutput', false);

    assert(isequal(individual_input, known_input), ...
        [BRAPH2.STR ':NNDataPoint_Measure_REG:' BRAPH2.FAIL_TEST], ...
        'NNDataPoint_Measure_REG does not construct the dataset correctly.
     The input value is not derived correctly.' ...
        )
end

%%% itest!
%%%% iname! (6)
Construct the data point with the adjacency matrix derived from its
     multiplex weighted undirected graph (MultiplexWU)
%%%% icode!
% ensure the example data is generated
if ~isfile([fileparts(which('SubjectCON_FUN_MP')) filesep 'Example data
     CON_FUN_MP XLS' filesep 'atlas.xlsx'])
    test_SubjectCON_FUN_MP % create example files
end

% Load BrainAtlas
im_ba = ImporterBrainAtlasXLS( ...
    'FILE', [fileparts(which('SubjectCON_FUN_MP')) filesep 'Example data
     CON_FUN_MP XLS' filesep 'atlas.xlsx'], ...
    'WAITBAR', true ...
    );

```

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

```
ba = im_ba.get('BA');

% Load Groups of SubjectCON
im_gr = ImporterGroupSubjectCON_XLS( ...
    'DIRECTORY', [fileparts(which('SubjectCON_FUN_MP')) filesep 'Example
     data CON_FUN_MP XLS' filesep 'CON_FUN_MP_Group_1_XLS.CON'], ...
    'BA', ba, ...
    'WAITBAR', true ...
    );

gr_CON = im_gr.get('GR');

% Load Groups of SubjectFUN
im_gr = ImporterGroupSubjectFUN_XLS( ...
    'DIRECTORY', [fileparts(which('SubjectCON_FUN_MP')) filesep 'Example
     data CON_FUN_MP XLS' filesep 'CON_FUN_MP_Group_1_XLS.FUN'], ...
    'BA', ba, ...
    'WAITBAR', true ...
    );

gr_FUN = im_gr.get('GR');

% Combine Groups of SubjectCON with Groups of SubjectFUN
co_gr = CombineGroups_CON_FUN_MP( ...
    'GR_CON', gr_CON, ...
    'GR_FUN', gr_FUN, ...
    'WAITBAR', true ...
    );

gr = co_gr.get('GR_CON_FUN_MP');

% Analysis CON FUN MP WU
a_WU = AnalyzeEnsemble_CON_FUN_MP_WU( ...
    'GR', gr ...
    );

% To be added the multiplex measures
a_WU.get('MEASUREENSEMBLE', 'Degree').get('M');
a_WU.get('MEASUREENSEMBLE', 'DegreeAv').get('M');
a_WU.get('MEASUREENSEMBLE', 'Distance').get('M');

% create item lists of NNDataPoint_Measure_REG
it_list = cellfun(@(g, sub) NNDataPoint_Measure_REG( ...
    'ID', sub.get('ID'), ...
    'G', g, ...
    'M_LIST', {'Degree' 'DegreeAv' 'Distance'}, ...
    'SUB', sub, ...
    'TARGET_IDS', sub.get('VOI_DICT').get('KEYS')), ...
    a_WU.get('G_DICT').get('IT_LIST'), gr.get('SUB_DICT').get('IT_LIST'),...
    'UniformOutput', false);

% create NNDataPoint_Measure_REG DICT items
dp_list = IndexedDictionary(...
        'IT_CLASS', 'NNDataPoint_Measure_REG', ...
        'IT_LIST', it_list ...
        );

% create a NNDataset containing the NNDataPoint_Measure_REG DICT
d = NNDataset( ...
    'DP_CLASS', 'NNDataPoint_Measure_REG', ...
    'DP_DICT', dp_list ...
```

```
    );

% Check whether the content of input for a single datapoint matches
for index = 1:1:gr.get('SUB_DICT').get('LENGTH')
    individual_input = d.get('DP_DICT').get('IT', index).get('INPUT');
    known_input = cellfun(@(m) m.get('M'), a_WU.get('G_DICT').get('IT',
     index).get('M_DICT').get('IT_LIST'), 'UniformOutput', false);

    assert(isequal(individual_input, known_input), ...
        [BRAPH2.STR ':NNDataPoint_Measure_REG:' BRAPH2.FAIL_TEST], ...
        'NNDataPoint_Measure_REG does not construct the dataset correctly.
     The input value is not derived correctly.' ...
        )
end

%%% itest!
%%%% iname! (7)
Example script for weighted undirected graph (GraphWU) using connectivity
     data
%%%% icode!
if ~isfile([fileparts(which('NNDataPoint_CON_REG')) filesep 'Example data NN
      REG CON XLS' filesep 'atlas.xlsx'])
    test_NNDataPoint_CON_REG % create example files
end
example_NNCV_CON_WU_M_REG

%%% itest!
%%%% iname! (8)
Example script for weighted undirected multiplex (MultiplexWU) using
     connectivity data and functional data
%%%% icode!
if ~isfile([fileparts(which('NNDataPoint_CON_FUN_MP_REG')) filesep 'Example
     data NN REG CON_FUN_MP XLS' filesep 'atlas.xlsx'])
    test_NNDataPoint_CON_FUN_MP_REG % create example files
end
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 obtrained 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.

```
%% iheader!
NNDataPoint_Measure_CLA < NNDataPoint (dp, measure classification data point
    ) is a data point for classification with graph measures.

%%% idescription!
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.
```

Code 22: **NNDataPoint_Measure_CLA element prop update.** 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.

```
%% iprops_update!

%%% iprop!
NAME (constant, string) is the name of a data point for classification with
    graph measures.
%%%% idefault!
'NNDataPoint_Measure_CLA'

%%% iprop!
DESCRIPTION (constant, string) is the description of a data point for
    classification with graph measures.
%%%% idefault!
'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.'

%%% iprop!
TEMPLATE (parameter, item) is the template of a data point for
    classification with graph measures.
%%%% isettings!
'NNDataPoint_Measure_CLA'

```

```
%%% iprop!
ID (data, string) is a few-letter code for a data point for classification
    with graph measures.
%%%% idefault!
'NNDataPoint_Measure_CLA ID'

%%% iprop!
LABEL (metadata, string) is an extended label of a data point for
    classification with graph measures.
%%%% idefault!
'NNDataPoint_Measure_CLA label'

%%% iprop!
NOTES (metadata, string) are some specific notes about a data point for
    classification with graph measures.
%%%% idefault!
'NNDataPoint_Measure_CLA notes'

%%% iprop!
INPUT (result, cell) is the input value for this data point.
%%%% icalculate!
value = cellfun(@(m_class) dp.get('G').get('MEASURE', m_class).get('M'), dp.
    get('M_LIST'), 'UniformOutput', false);  (1)

%%% iprop!
TARGET (result, cell) is the target value for this data point.
%%%% icalculate!
value = dp.get('TARGET_IDS');
```

(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 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`.

```
%% iprops!

%%% iprop!
G (data, item) is a graph.
%%%% isettings!
'Graph'

%%% iprop!  (1)
M_LIST (parameter, classlist) is a list of graph measure to be used as the
    input

%%% iprop!
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 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.

```
%% itests!

%%% iexcluded_props!
[NNDataPoint_Measure_CLA.G]

%%% itest!
%%%% iname! (1)
Construct the data point with the graph measures derived from its weighted
     undirected graph (GraphWU)
%%%% icode!
% ensure the example data is generated
if ~isfile([fileparts(which('NNDataPoint_CON_CLA')) filesep 'Example data NN
      CLA CON XLS' filesep 'atlas.xlsx'])
    test_NNDataPoint_CON_CLA % create example files
end

% Load BrainAtlas
im_ba = ImporterBrainAtlasXLS( ...
    'FILE', [fileparts(which('NNDataPoint_CON_CLA')) filesep 'Example data
     NN CLA CON XLS' filesep 'atlas.xlsx'], ...
    'WAITBAR', true ...
    );

ba = im_ba.get('BA');

% Load Groups of SubjectCON
im_gr1 = ImporterGroupSubjectCON_XLS( ...
    'DIRECTORY', [fileparts(which('NNDataPoint_CON_CLA')) filesep 'Example
     data NN CLA CON XLS' filesep 'CON_Group_1_XLS'], ...
    'BA', ba, ...
    'WAITBAR', true ...
    );

gr1 = im_gr1.get('GR');

im_gr2 = ImporterGroupSubjectCON_XLS( ...
    'DIRECTORY', [fileparts(which('NNDataPoint_CON_CLA')) filesep 'Example
     data NN CLA CON XLS' filesep 'CON_Group_2_XLS'], ...
    'BA', ba, ...
    'WAITBAR', true ...
    );

gr2 = im_gr2.get('GR');

% Analysis CON WU
a_WU1 = AnalyzeEnsemble_CON_WU( ...
    'GR', gr1 ...
    );

a_WU2 = AnalyzeEnsemble_CON_WU( ...
    'TEMPLATE', a_WU1, ...
    'GR', gr2 ...
    );

```

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

```
a_WU1.get('MEASUREENSEMBLE', 'Degree').get('M');  (2)
a_WU1.get('MEASUREENSEMBLE', 'DegreeAv').get('M'); (3)
a_WU1.get('MEASUREENSEMBLE', 'Distance').get('M'); (4)

a_WU2.get('MEASUREENSEMBLE', 'Degree').get('M');
a_WU2.get('MEASUREENSEMBLE', 'DegreeAv').get('M');
a_WU2.get('MEASUREENSEMBLE', 'Distance').get('M');

% create item lists of NNDataPoint_Graph_CLA
[~, group_folder_name] = fileparts(im_gr1.get('DIRECTORY'));
it_list1 = cellfun(@(x) NNDataPoint_Measure_CLA( ...
    'ID', x.get('ID'), ...
    'G', x, ...
    'M_LIST', {'Degree' 'DegreeAv' 'Distance'}, ...
    'TARGET_IDS', {group_folder_name}), ...
    a_WU1.get('G_DICT').get('IT_LIST'), ...
    'UniformOutput', false);

[~, group_folder_name] = fileparts(im_gr2.get('DIRECTORY'));
it_list2 = cellfun(@(x) NNDataPoint_Measure_CLA( ...
    'ID', x.get('ID'), ...
    'G', x, ...
    'M_LIST', {'Degree' 'DegreeAv' 'Distance'}, ...
    'TARGET_IDS', {group_folder_name}), ...
    a_WU2.get('G_DICT').get('IT_LIST'), ...
    'UniformOutput', false);

% create NNDataPoint_Graph_CLA DICT items
dp_list1 = IndexedDictionary(...
        'IT_CLASS', 'NNDataPoint_Measure_CLA', ...
        'IT_LIST', it_list1 ...
        );

dp_list2 = IndexedDictionary(...
        'IT_CLASS', 'NNDataPoint_Measure_CLA', ...
        'IT_LIST', it_list2 ...
        );

% create a NNDataset containing the NNDataPoint_Measure_CLA DICT
d1 = NNDataset( ...
    'DP_CLASS', 'NNDataPoint_Measure_CLA', ...
    'DP_DICT', dp_list1 ...
    );

d2 = NNDataset( ...
    'DP_CLASS', 'NNDataPoint_Measure_CLA', ...
    'DP_DICT', dp_list2 ...
    );

% Check whether the content of input for a single datapoint matches
for index = 1:1:gr1.get('SUB_DICT').get('LENGTH')
    individual_input = d1.get('DP_DICT').get('IT', index).get('INPUT');
    known_input = cellfun(@(m) m.get('M'), a_WU1.get('G_DICT').get('IT',
     index).get('M_DICT').get('IT_LIST'), 'UniformOutput', false);

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

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

```
    'TEMPLATE', a_BUD1, ...
    'GR', gr2 ...
    );

a_BUD1.get('MEASUREENSEMBLE', 'Degree').get('M');
a_BUD1.get('MEASUREENSEMBLE', 'DegreeAv').get('M');
a_BUD1.get('MEASUREENSEMBLE', 'Distance').get('M');

a_BUD2.get('MEASUREENSEMBLE', 'Degree').get('M');
a_BUD2.get('MEASUREENSEMBLE', 'DegreeAv').get('M');
a_BUD2.get('MEASUREENSEMBLE', 'Distance').get('M');

% create item lists of NNDataPoint_Graph_CLA
[~, group_folder_name] = fileparts(im_gr1.get('DIRECTORY'));
it_list1 = cellfun(@(x) NNDataPoint_Measure_CLA( ...
    'ID', x.get('ID'), ...
    'G', x, ...
    'M_LIST', {'Degree' 'DegreeAv' 'Distance'}, ...
    'TARGET_IDS', {group_folder_name}), ...
    a_BUD1.get('G_DICT').get('IT_LIST'), ...
    'UniformOutput', false);

[~, group_folder_name] = fileparts(im_gr2.get('DIRECTORY'));
it_list2 = cellfun(@(x) NNDataPoint_Measure_CLA( ...
    'ID', x.get('ID'), ...
    'G', x, ...
    'M_LIST', {'Degree' 'DegreeAv' 'Distance'}, ...
    'TARGET_IDS', {group_folder_name}), ...
    a_BUD2.get('G_DICT').get('IT_LIST'), ...
    'UniformOutput', false);

% create NNDataPoint_Graph_CLA DICT items
dp_list1 = IndexedDictionary(...
        'IT_CLASS', 'NNDataPoint_Measure_CLA', ...
        'IT_LIST', it_list1 ...
        );

dp_list2 = IndexedDictionary(...
        'IT_CLASS', 'NNDataPoint_Measure_CLA', ...
        'IT_LIST', it_list2 ...
        );

% create a NNDataset containing the NNDataPoint_Measure_CLA DICT
d1 = NNDataset( ...
    'DP_CLASS', 'NNDataPoint_Measure_CLA', ...
    'DP_DICT', dp_list1 ...
    );

d2 = NNDataset( ...
    'DP_CLASS', 'NNDataPoint_Measure_CLA', ...
    'DP_DICT', dp_list2 ...
    );

% Check whether the content of input for a single datapoint matches
for index = 1:1:gr1.get('SUB_DICT').get('LENGTH')
    individual_input = d1.get('DP_DICT').get('IT', index).get('INPUT');
    known_input = cellfun(@(m) m.get('M'), a_BUD1.get('G_DICT').get('IT',
     index).get('M_DICT').get('IT_LIST'), 'UniformOutput', false);

    assert(isequal(individual_input, known_input), ...
        [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, ...
```

```
    'WAITBAR', true ...
    );

gr1_FUN = im_gr1.get('GR');

im_gr2 = ImporterGroupSubjectFUN_XLS( ...
    'DIRECTORY', [fileparts(which('SubjectCON_FUN_MP')) filesep 'Example
     data CON_FUN_MP XLS' filesep 'CON_FUN_MP_Group_2_XLS.FUN'], ...
    'BA', ba, ...
    'WAITBAR', true ...
    );

gr2_FUN = im_gr2.get('GR');

% Combine Groups of SubjectCON with Groups of SubjectFUN
co_gr1 = CombineGroups_CON_FUN_MP( ...
    'GR_CON', gr1_CON, ...
    'GR_FUN', gr1_FUN, ...
    'WAITBAR', true ...
    );

gr1 = co_gr1.get('GR_CON_FUN_MP');

co_gr2 = CombineGroups_CON_FUN_MP( ...
    'GR_CON', gr2_CON, ...
    'GR_FUN', gr2_FUN, ...
    'WAITBAR', true ...
    );

gr2 = co_gr2.get('GR_CON_FUN_MP');

% Analysis CON FUN MP WU
a_WU1 = AnalyzeEnsemble_CON_FUN_MP_WU( ...
    'GR', gr1 ...
    );

a_WU2 = AnalyzeEnsemble_CON_FUN_MP_WU( ...
    'TEMPLATE', a_WU1, ...
    'GR', gr2 ...
    );

% To be added the multiplex measures
a_WU1.get('MEASUREENSEMBLE', 'Degree').get('M');
a_WU1.get('MEASUREENSEMBLE', 'DegreeAv').get('M');
a_WU1.get('MEASUREENSEMBLE', 'Distance').get('M');

a_WU2.get('MEASUREENSEMBLE', 'Degree').get('M');
a_WU2.get('MEASUREENSEMBLE', 'DegreeAv').get('M');
a_WU2.get('MEASUREENSEMBLE', 'Distance').get('M');

% create item lists of NNDataPoint_Graph_CLA
[~, group_folder_name] = fileparts(im_gr1.get('DIRECTORY'));
it_list1 = cellfun(@(x) NNDataPoint_Measure_CLA( ...
    'ID', x.get('ID'), ...
    'G', x, ...
    'M_LIST', {'Degree' 'DegreeAv' 'Distance'}, ...
    'TARGET_IDS', {group_folder_name}), ...
    a_WU1.get('G_DICT').get('IT_LIST'), ...
    'UniformOutput', false);

[~, group_folder_name] = fileparts(im_gr2.get('DIRECTORY'));
```

```
it_list2 = cellfun(@(x) NNDataPoint_Measure_CLA( ...
    'ID', x.get('ID'), ...
    'G', x, ...
    'M_LIST', {'Degree' 'DegreeAv' 'Distance'}, ...
    'TARGET_IDS', {group_folder_name}), ...
    a_WU2.get('G_DICT').get('IT_LIST'), ...
    'UniformOutput', false);

% create NNDataPoint_Graph_CLA DICT items
dp_list1 = IndexedDictionary(...
        'IT_CLASS', 'NNDataPoint_Measure_CLA', ...
        'IT_LIST', it_list1 ...
        );

dp_list2 = IndexedDictionary(...
        'IT_CLASS', 'NNDataPoint_Measure_CLA', ...
        'IT_LIST', it_list2 ...
        );

% create a NNDataset containing the NNDataPoint_Measure_CLA DICT
d1 = NNDataset( ...
    'DP_CLASS', 'NNDataPoint_Measure_CLA', ...
    'DP_DICT', dp_list1 ...
    );

d2 = NNDataset( ...
    'DP_CLASS', 'NNDataPoint_Measure_CLA', ...
    'DP_DICT', dp_list2 ...
    );

% Check whether the content of input for a single datapoint matches
for index = 1:1:gr1.get('SUB_DICT').get('LENGTH')
    individual_input = d1.get('DP_DICT').get('IT', index).get('INPUT');
    known_input = cellfun(@(m) m.get('M'), a_WU1.get('G_DICT').get('IT',
     index).get('M_DICT').get('IT_LIST'), 'UniformOutput', false);

    assert(isequal(individual_input, known_input), ...
        [BRAPH2.STR ':NNDataPoint_Measure_CLA:' BRAPH2.FAIL_TEST], ...
        'NNDataPoint_Measure_CLA does not construct the dataset correctly.
     The input value is not derived correctly.' ...
        )
end

for index = 1:1:gr2.get('SUB_DICT').get('LENGTH')
    individual_input = d2.get('DP_DICT').get('IT', index).get('INPUT');
    known_input = cellfun(@(m) m.get('M'), a_WU2.get('G_DICT').get('IT',
     index).get('M_DICT').get('IT_LIST'), 'UniformOutput', false);

    assert(isequal(individual_input, known_input), ...
        [BRAPH2.STR ':NNDataPoint_Measure_CLA:' BRAPH2.FAIL_TEST], ...
        'NNDataPoint_Measure_CLA does not construct the dataset correctly.
     The input value is not derived correctly.' ...
        )
end

%%% itest!
%%%% iname! (6)
Example script for binary undirected graph at fixed densities (GraphBUD)
     using connectivity data
%%%% icode!
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.