Implement a new Neural Network Data Point

The BRAPH 2 Developers

September 20, 2023

This is the developer tutorial for implementing a new neural network data point. In this Tutorial, we will explain how to create the generator file `*.gen.m` for a new neural network data point, which can then be compiled by `braph2genesis`. All kinds of neural network data point are (direct or indirect) extensions of the base element `NNDataPoint`. Here, we will use as examples the neural network data point `NNDataPoint_CON_REG` (connectivity data for regression), `NNDataPoint_CON_CLA` (connectivity data for classification), `NNDataPoint_Graph_REG` (adjacency matrix for regression), `NNDataPoint_Graph_CLA` (adjacency matrix for classification), `NNDataPoint_Measure_REG` (graph measure for regression), and `NNDataPoint_Measure_CLA` (graph measure for classification).

Contents

Implementation of a Data Point with Connectivity Data

*Connectivity Data Point for Regression (*NNDataPoint_CON_REG*)*

We will start by implementing in detail NNDataPoint_CON_REG, which is a direct extension of NNDataPoint. A data point for regression with connectivity data NNDataPoint_CON_REG contains the input and target for neural network analysis with a subject with connectivity data (SubjectCON), where the input is the subject's connectivity data and the target is the subject's variables of interest.

Code 1: **NNDataPoint_CON_REG element header.** The header section of the generator code for _NNDataPoint_CON_REG.gen.m provides the general information about the NNDataPoint_CON_REG element.

```
%% iheader!
NNDataPoint_CON_REG < NNDataPoint (dp, connectivity regression data point)
     is a data point for regression with connectivity data.
 (1)

%%% idescription!
A data point for regression with connectivity data (NNDataPoint_CON_REG)
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.
```

(1) defines NNDataPoint_CON_REG as a subclass of NNDataPoint. The moniker will be dp.

Code 2: **NNDataPoint_CON_REG element prop update.** The props_update section of the generator code for _NNDataPoint_CON_REG.gen.m updates the properties of the NNDataPoint_CON_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
     connectivity data.
%%%% idefault!
'NNDataPoint_CON_REG'

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

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

```
17
18 %%% iprop!
19 ID (data, string) is a few-letter code for a data point for regression with
       connectivity data.
20 %%%% idefault!
21 'NNDataPoint_CON_REG ID'
22
23 %%% iprop!
24 LABEL (metadata, string) is an extended label of a data point for regression
        with connectivity data.
25 %%%% idefault!
26 'NNDataPoint_CON_REG label'
27
28 %%% iprop!
29 NOTES (metadata, string) are some specific notes about a data point for
       regression with connectivity data.
30 %%%% idefault!
31 'NNDataPoint_CON_REG notes'
32
33 %%% iprop! (1)
34 INPUT (result, cell) is the input value for this data point.
35 %%%% icalculate!
36 value = {dp.get('SUB').get('CON')};
37
38 %%% iprop! (2)
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) The property `INPUT` is the input value for this data point, which is obtained directly from the connectivity data of `Subject_CON` by the code under `icalculate!`.

(2) The property `TARGET` is the target value for this data point, which is obtained directly from the variables of interest of `VOI_DICT` by the code under `icalculate!`.

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

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

(1) The property `SUB` is a subject with connectivity data (`Subject_CON`), which is used to calculated the mentioned properties `INPUT` and `TARGET`.

(2) The property `TARGET_IDS` defines the IDs of target, where the target IDs should be from the subject's variable-of-interest IDs.

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 data point 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.

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

%%% idescription!
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.

```
%% iprops_update!

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

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

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

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

%%% iprop!
```

```
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! (1)
TARGET (result, stringlist) is the target values for this data point.
%%%% icalculate!
value = dp.get('TARGET_IDS');
```

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

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!
TARGET_IDS (parameter, stringlist) is a list of variable-of-interest IDs to
    be used as the class targets.
```

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);

    ...

    % 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
        r = 0 + (0.5 - 0)*rand(size(A1)); diffA = A1 - r; A1(A1 ~= 0) =
     diffA(A1 ~= 0); % make the adjacency matrix weighted
        A1 = max(A1, transpose(A1)); % make the adjacency matrix symmetric

        writetable(array2table(A1), [gr1_dir filesep() sub_id '.xlsx'], '
     WriteVariableNames', false)

        % variables of interest
        vois1 = [vois1; {sub_id, randi(90), sex_options(randi(2))}];
    end
    writetable(table(vois1), [data_dir filesep() gr1_name '.vois.xlsx'], '
     WriteVariableNames', false)

```

(1) creates the first group of simulated data.

```
% Group 2 (2)
K2 = 2; % degree (mean node degree is 2K) - group 2
beta2 = 0.85; % Rewiring probability - group 2 (3)
gr2_name = 'CON_Group_2_XLS';
gr2_dir = [data_dir filesep() gr2_name];
mkdir(gr2_dir);
vois2 = [
    {{'Subject ID'} {'Age'} {'Sex'}}
    {{} {} cell2str(sex_options)}
    ];
for i = 51:1:100
    sub_id = ['SubjectCON_' num2str(i)];

    h2 = WattsStrogatz(N, K2, beta2);
    % 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

    A2 = full(adjacency(h2)); A2(1:length(A2)+1:numel(A2)) = 0;
    r = 0 + (0.5 - 0)*rand(size(A2)); diffA = A2 - r; A2(A2 ~= 0) =
 diffA(A2 ~= 0);
    A2 = max(A2, transpose(A2));

    writetable(array2table(A2), [gr2_dir filesep() sub_id '.xlsx'], '
 WriteVariableNames', false)

    % variables of interest
    vois2 = [vois2; {sub_id, randi(90), sex_options(randi(2))}];
end
writetable(table(vois2), [data_dir filesep() gr2_name '.vois.xlsx'], '
 WriteVariableNames', false)

% Group 3 (4)
K3 = 2; % degree (mean node degree is 2K) - group 3
beta3 = 0.55; % Rewiring probability - group 3  (5)
gr3_name = 'CON_Group_3_XLS';
gr3_dir = [data_dir filesep() gr3_name];
mkdir(gr3_dir);
vois3 = [
    {{'Subject ID'} {'Age'} {'Sex'}}
    {{} {} cell2str(sex_options)}
    ];
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

```

(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.

```
        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)
...

%%% 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'), ...
```

(6) imports two groups of simulated data.

(7) creates two datasets for the two groups.

```
    '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( ...
    '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.' ...
    )
```

(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.

```
209
210 % Check whether the content of input for a single data point matches (10)
211 for index = 1:1:gr1.get('SUB_DICT').get('LENGTH')
212     individual_input = d1.get('DP_DICT').get('IT', index).get('INPUT');
213     known_input = {gr1.get('SUB_DICT').get('IT', index).get('CON')};
214
215     assert(isequal(individual_input, known_input), ...
216         [BRAPH2.STR ':NNDataPoint_CON_CLA:' BRAPH2.FAIL_TEST], ...
217         'NNDataPoint_CON_CLA does not construct the dataset correctly. The
      input value is not derived correctly.' ...
218         )
219 end
220
221 for index = 1:1:gr2.get('SUB_DICT').get('LENGTH')
222     individual_input = d2.get('DP_DICT').get('IT', index).get('INPUT');
223     known_input = {gr2.get('SUB_DICT').get('IT', index).get('CON')};
224
225     assert(isequal(individual_input, known_input), ...
226         [BRAPH2.STR ':NNDataPoint_CON_CLA:' BRAPH2.FAIL_TEST], ...
227         'NNDataPoint_CON_CLA does not construct the dataset correctly. The
      input value is not derived correctly.' ...
228         )
229 end
230
231 %%% itest!
232 %%%% iname!
233 Example training-test classification (11)
234 %%%% icode!
235 % ensure the example data is generated
236 if ~isfile([fileparts(which('NNDataPoint_CON_CLA')) filesep 'Example data NN
       CLA CON XLS' filesep 'atlas.xlsx'])
237     test_NNDataPoint_CON_CLA % create example files
238 end
239
240 example_NN_CON_CLA
```

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

(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 data point 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

...

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

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

% create item lists of NNDataPoint_Graph_REG
it_list = cellfun(@(g, sub)  (4)NNDataPoint_Graph_REG( ...
    'ID', sub.get('ID'), ...
    'G', g, ...
    'SUB', sub, ...
    'TARGET_IDS', sub.get('VOI_DICT').get('KEYS')), ...
    (5)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 data point matches (6)
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');
```

(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) and (5) creates the `NNDataPoint_Graph_REG` element and use the `Graph` from `G_DICT`.

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

```
51
52      assert(isequal(individual_input, known_input), ...
53          [BRAPH2.STR ':NNDataPoint_Graph_REG:' BRAPH2.FAIL_TEST], ...
54          'NNDataPoint_Graph_REG does not construct the dataset correctly. The
          input value is not derived correctly.' ...
55          )
56  end
57
58  %%% itest!
59  %%%% iname! (7)
60  Construct the data point with the adjacency matrix derived from its binary
        undirected multigraph with fixed densities (MultigraphBUD)
61  %%%% icode!
62  % ensure the example data is generated
63  if ~isfile([fileparts(which('NNDataPoint_CON_REG')) filesep 'Example data NN
         REG CON XLS' filesep 'atlas.xlsx'])
64      test_NNDataPoint_CON_REG % create example files
65  end
66
67  % Load BrainAtlas
68
69  ...
70
71  % Analysis CON WU
72  densities = 0:25:100;
73
74  a_BUD = (8) AnalyzeEnsemble_CON_BUD( ...
75      'DENSITIES', densities, ...
76      'GR', gr ...
77      );
78
79  a_BUD.memorize('G_DICT');
80
81  % create item lists of NNDataPoint_Graph_REG
82  it_list = cellfun(@(g, sub) (9) NNDataPoint_Graph_REG( ...
83      'ID', sub.get('ID'), ...
84      'G', g, ...
85      'SUB', sub, ...
86      'TARGET_IDS', sub.get('VOI_DICT').get('KEYS')), ...
87      (10) a_BUD.get('G_DICT').get('IT_LIST'), gr.get('SUB_DICT').get('IT_LIST'
         ),...
88      'UniformOutput', false);
89
90  % create NNDataPoint_Graph_REG DICT items
91  dp_list = IndexedDictionary(...
92          'IT_CLASS', 'NNDataPoint_Graph_REG', ...
93          'IT_LIST', it_list ...
94          );
95
96  ...
97
98  %%% itest!
99  %%%% iname! (11)
100 Construct the data point with the adjacency matrix derived from its
        multiplex weighted undirected graph (MultiplexWU)
101 %%%% icode!
102 % ensure the example data is generated
103 if ~isfile([fileparts(which('SubjectCON_FUN_MP')) filesep 'Example data
        CON_FUN_MP XLS' filesep 'atlas.xlsx'])
```

(7) tests with the `MultigraphBUD` element which contains the adjacency matrix of binary undirected graph at fixed densities.

(8), (9), and (10) creates the `NNDataPoint_Graph_REG` element and use the `Graph` from `AnalyzeEnsemble_CON_BUD`.

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

```
    test_SubjectCON_FUN_MP % create example files
end

% Load BrainAtlas

...

% Analysis CON FUN MP WU
a_WU = (12)AnalyzeEnsemble_CON_FUN_MP_WU( ...
    'GR', gr ...
    );

a_WU.memorize('G_DICT');

% create item lists of NNDataPoint_Graph_REG
it_list = cellfun(@(g, sub) (13)NNDataPoint_Graph_REG( ...
    'ID', sub.get('ID'), ...
    'G', g, ...
    'SUB', sub, ...
    'TARGET_IDS', sub.get('VOI_DICT').get('KEYS')), ...
    (14)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 ...
        );

...

%%% itest!
%%%% iname! (15)
Example script for binary undirected graph (MultigraphBUT) 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_BUT_REG

%%% itest!
%%%% iname! (16)
Example script for binary undirected multiplex at fixed densities (
     MultiplexBUD) 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_BUD_REG

%%% itest!
%%%% iname! (17)
Example script for binary undirected multiplex at fixed thresholds (
     MultiplexBUT) using connectivity data and functional data
```

(12), (13), and (14) creates the `NNDataPoint_Graph_REG` element and use the `Graph` from `AnalyzeEnsemble_CON_BUD`.

(15) tests with the `MultigraphBUT` element with the simulated connectivity data.

(16) tests with the `MultiplexBUD` element with the simulated connectivity and functional data.

(17) tests with the `MultiplexBUT` element with the simulated connectivity 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_BUT_REG
```

*Graph Data Point for Classification (*NNDataPoint_Graph_CLA*)*

Now we implement NNDataPoint_Graph_CLA based on previous codes NNDataPoint_CON_CLA. This neural network data point 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

...

% Analysis CON WU
a_WU1 = (2)AnalyzeEnsemble_CON_WU( ...
    'GR', gr1 ...
    );

a_WU2 = AnalyzeEnsemble_CON_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) (3)NNDataPoint_Graph_CLA( ...
    'ID', x.get('ID'), ...
    'G', x, ...
    'TARGET_IDS', {group_folder_name}), ...
    (4)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

...
```

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

(2), (3), and (4) create `Analyzensemble_CON_WU` and use its `G_DICT` to initialize `NNDataPoint_Graph_CLA`.

```
52
53 %%% itest!
54 %%%% iname! (5)
55 Construct the data point with the adjacency matrix derived from its binary
       undirected multigraph with fixed densities (MultigraphBUD)
56 %%%% icode!
57 % ensure the example data is generated
58 if ~isfile([fileparts(which('NNDataPoint_CON_CLA')) filesep 'Example data NN
       CLA CON XLS' filesep 'atlas.xlsx'])
59     test_NNDataPoint_CON_CLA % create example files
60 end
61
62 % Load BrainAtlas
63
64 ...
65
66 % Analysis CON WU
67 densities = 0:25:100;
68
69 a_BUD1 =  (6)AnalyzeEnsemble_CON_BUD( ...
70     'DENSITIES', densities, ...
71     'GR', gr1 ...
72     );
73
74 a_BUD2 = AnalyzeEnsemble_CON_BUD( ...
75     'TEMPLATE', a_BUD1, ...
76     'GR', gr2 ...
77     );
78
79 a_BUD1.memorize('G_DICT');
80 a_BUD2.memorize('G_DICT');
81
82 % create item lists of NNDataPoint_Graph_CLA
83 [~, group_folder_name] = fileparts(im_gr1.get('DIRECTORY'));
84 it_list1 = cellfun(@(x) (7)NNDataPoint_Graph_CLA( ...
85     'ID', x.get('ID'), ...
86     'G', x, ...
87     'TARGET_IDS', {group_folder_name}), ...
88     (8)a_BUD1.get('G_DICT').get('IT_LIST'), ...
89     'UniformOutput', false);
90
91 [~, group_folder_name] = fileparts(im_gr2.get('DIRECTORY'));
92 it_list2 = cellfun(@(x) NNDataPoint_Graph_CLA( ...
93     'ID', x.get('ID'), ...
94     'G', x, ...
95     'TARGET_IDS', {group_folder_name}), ...
96     a_BUD2.get('G_DICT').get('IT_LIST'), ...
97     'UniformOutput', false);
98
99 % create NNDataPoint_Graph_CLA DICT items
100 dp_list1 = IndexedDictionary(...
101         'IT_CLASS', 'NNDataPoint_Graph_CLA', ...
102         'IT_LIST', it_list1 ...
103         );
104
105 dp_list2 = IndexedDictionary(...
106         'IT_CLASS', 'NNDataPoint_Graph_CLA', ...
107         'IT_LIST', it_list2 ...
108         );
109
```

(5) tests with the `MultigraphBUD` element which contains binary undirected adjacency matrix at fixed densities.

(6), (7), and (8) create `Analyzensemble_CON_BUD` and use its `G_DICT` to initialize `NNDataPoint_Graph_CLA`.

```
% create a NNDataset containing the NNDataPoint_Graph_CLA DICT

...

%%% itest!
%%%% iname! (9)
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

...

% Analysis CON FUN MP WU
a_WU1 = (10)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) (11)NNDataPoint_Graph_CLA( ...
    'ID', x.get('ID'), ...
    'G', x, ...
    'TARGET_IDS', {group_folder_name}), ...
    (12)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

...

%%% itest!
%%%% iname! (13)
Example script for weighted undirected graph (GraphWU) using connectivity
     data
%%%% icode!
if ~isfile([fileparts(which('NNDataPoint_CON_CLA')) filesep 'Example data NN
```

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

(10), (11), and (12) create `Analyzensemble_CON_FUN_MP_WU` and use its `G_DICT` to initialize `NNDataPoint_Graph_CLA`.

(13) tests with the `GraphWU` element with simulated data.

```
      CLA CON XLS' filesep 'atlas.xlsx'])
166     test_NNDataPoint_CON_CLA % create example files
167 end
168 example_NNCV_CON_WU_CLA
169
170 %%% itest!
171 %%%% iname! (14)
172 Example script for binary undirected graph at fixed densities (MultigraphBUD
      ) using connectivity data
173 %%%% icode!
174 if ~isfile([fileparts(which('NNDataPoint_CON_CLA')) filesep 'Example data NN
      CLA CON XLS' filesep 'atlas.xlsx'])
175     test_NNDataPoint_CON_CLA % create example files
176 end
177 example_NNCV_CON_BUD_CLA
178
179 %%% itest!
180 %%%% iname! (15)
181 Example script for weighted undirected multiplex (MultiplexWU) using
      connectivity data and functional data
182 %%%% icode!
183 if ~isfile([fileparts(which('NNDataPoint_CON_FUN_MP_CLA')) filesep 'Example
      data NN CLA CON_FUN_MP XLS' filesep 'atlas.xlsx'])
184     test_NNDataPoint_CON_FUN_MP_CLA % create example files
185 end
186 example_NNCV_CON_FUN_MP_WU_CLA
```

(14) tests with the `MultigraphBUD` element with simulated data.

(15) 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 data point 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 the graph measures, 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!
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

...

% Analysis CON WU
a_WU = (1)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)  (5)NNDataPoint_Measure_REG( ...
    'ID', sub.get('ID'), ...
    'G', g, ...
    'M_LIST', {'Degree' 'DegreeAv' 'Distance'}, ...
    'SUB', sub, ...
    'TARGET_IDS', sub.get('VOI_DICT').get('KEYS')), ...
     (6)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 ...
        );

...

%%% itest!
%%%% iname! (7)
Construct the data point with the adjacency matrix derived from its binary
      undirected multigraph with fixed densities (MultigraphBUD)
```

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

(5) and (6) use `AnalyzeEnsemble_CON_WU`'s `G_DICT` to set up a `NNDataPoint_Measure_REG`.

(7), (8), (9), and (10) tests various kinds of graph measure with the `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

...

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

a_BUD = (8)AnalyzeEnsemble_CON_BUD( ...
    'DENSITIES', densities, ...
    '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) (9)NNDataPoint_Measure_REG( ...
    'ID', sub.get('ID'), ...
    'G', g, ...
    'M_LIST', {'Degree' 'DegreeAv' 'Distance'}, ...
    'SUB', sub, ...
    'TARGET_IDS', sub.get('VOI_DICT').get('KEYS')), ...
    (10)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 ...
        );

...

%%% itest!
%%%% iname! (11)
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

...

% Analysis CON FUN MP WU
a_WU = (12)AnalyzeEnsemble_CON_FUN_MP_WU( ...
```

(11), (12), (13), and (14) tests various kinds of graph measure with the `MultiplexWU`.

```
104     'GR', gr ...
105     );
106
107 % To be added the multiplex measures
108 a_WU.get('MEASUREENSEMBLE', 'Degree').get('M');
109 a_WU.get('MEASUREENSEMBLE', 'DegreeAv').get('M');
110 a_WU.get('MEASUREENSEMBLE', 'Distance').get('M');
111
112 % create item lists of NNDataPoint_Measure_REG
113 it_list = cellfun(@(g, sub) (13)NNDataPoint_Measure_REG( ...
114     'ID', sub.get('ID'), ...
115     'G', g, ...
116     'M_LIST', {'Degree' 'DegreeAv' 'Distance'}, ...
117     'SUB', sub, ...
118     'TARGET_IDS', sub.get('VOI_DICT').get('KEYS')), ...
119     (14)a_WU.get('G_DICT').get('IT_LIST'), gr.get('SUB_DICT').get('IT_LIST')
        ,...
120     'UniformOutput', false);
121
122 % create NNDataPoint_Measure_REG DICT items
123
124 ...
125
126 %%% itest!
127 %%%% iname! (15)
128 Example script for weighted undirected graph (GraphWU) using connectivity
        data
129 %%%% icode!
130 if ~isfile([fileparts(which('NNDataPoint_CON_REG')) filesep 'Example data NN
         REG CON XLS' filesep 'atlas.xlsx'])
131     test_NNDataPoint_CON_REG % create example files
132 end
133 example_NNCV_CON_WU_M_REG
134
135 %%% itest!
136 %%%% iname! (16)
137 Example script for weighted undirected multiplex (MultiplexWU) using
        connectivity data and functional data
138 %%%% icode!
139 if ~isfile([fileparts(which('NNDataPoint_CON_FUN_MP_REG')) filesep 'Example
        data NN REG CON_FUN_MP XLS' filesep 'atlas.xlsx'])
140     test_NNDataPoint_CON_FUN_MP_REG % create example files
141 end
142 example_NNCV_CON_FUN_MP_WU_M_REG
```

(15) tests various kinds of graph measure with the `GraphWU` using example data.

(16) tests 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 data point 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!
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

...

% Analysis CON WU
a_WU1 = (1)AnalyzeEnsemble_CON_WU( ...
    'GR', gr1 ...
    );

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

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_Measure_CLA
[~, group_folder_name] = fileparts(im_gr1.get('DIRECTORY'));
it_list1 = cellfun(@(x) (5)NNDataPoint_Measure_CLA( ...
    'ID', x.get('ID'), ...
    'G', x, ...
    'M_LIST', {'Degree' 'DegreeAv' 'Distance'}, ...
    'TARGET_IDS', {group_folder_name}), ...
    (6)a_WU1.get('G_DICT').get('IT_LIST'), ...
    'UniformOutput', false);

% create NNDataPoint_Measure_CLA DICT items

...

%%% itest!
```

(1), (2), (3), and (4) create a `AnalyzeEnsemble_CON_WU` and add various kinds of graph measure with the `GraphWU` element which contains weighted undirected adjacency matrix. (5) and (6) use `AnalyzeEnsemble_CON_WU`'s `G_DICT` to set up a `NNDataPoint_Measure_CLA`.

```
%%%% iname! (7)
Construct the data point with the graph measures derived from its binary
     undirected multigraph with fixed densities (MultigraphBUD)
%%%% 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

...

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

a_BUD1 = (8)AnalyzeEnsemble_CON_BUD( ...
    'DENSITIES', densities, ...
    'GR', gr1 ...
    );

a_BUD2 = AnalyzeEnsemble_CON_BUD( ...
    'TEMPLATE', a_BUD1, ...
    'GR', gr2 ...
    );

a_BUD1.get('MEASUREENSEMBLE', 'Degree').get('M'); (9)
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_Measure_CLA
[~, group_folder_name] = fileparts(im_gr1.get('DIRECTORY'));
it_list1 = cellfun(@(x) (10)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_Measure_CLA items
dp_list1 = IndexedDictionary(...
        'IT_CLASS', 'NNDataPoint_Measure_CLA', ...
        'IT_LIST', it_list1 ...
        );

```

(7), (8), (9), and (10) tests various kinds of graph measure with the `MultigraphBUD`.

```
110 ...
111
112 %%% itest!
113 %%%% iname!  (11)
114 Construct the data point with the graph measures derived from its multiplex
      weighted undirected graph (MultiplexWU)
115 %%%% icode!
116 % ensure the example data is generated
117 if ~isfile([fileparts(which('SubjectCON_FUN_MP')) filesep 'Example data
      CON_FUN_MP XLS' filesep 'atlas.xlsx'])
118     test_SubjectCON_FUN_MP % create example files
119 end
120
121 % Load BrainAtlas
122
123 ...
124
125 % Analysis CON FUN MP WU
126 a_WU1 =  (12)AnalyzeEnsemble_CON_FUN_MP_WU( ...
127     'GR', gr1 ...
128     );
129
130 a_WU2 = AnalyzeEnsemble_CON_FUN_MP_WU( ...
131     'TEMPLATE', a_WU1, ...
132     'GR', gr2 ...
133     );
134
135 % To be added the multiplex measures
136 a_WU1.get('MEASUREENSEMBLE', 'Degree').get('M'); (13)
137 a_WU1.get('MEASUREENSEMBLE', 'DegreeAv').get('M');
138 a_WU1.get('MEASUREENSEMBLE', 'Distance').get('M');
139
140 a_WU2.get('MEASUREENSEMBLE', 'Degree').get('M');
141 a_WU2.get('MEASUREENSEMBLE', 'DegreeAv').get('M');
142 a_WU2.get('MEASUREENSEMBLE', 'Distance').get('M');
143
144 % create item lists of NNDataPoint_Graph_CLA
145 [~, group_folder_name] = fileparts(im_gr1.get('DIRECTORY'));
146 it_list1 = cellfun(@(x) (14)NNDataPoint_Measure_CLA( ...
147     'ID', x.get('ID'), ...
148     'G', x, ...
149     'M_LIST', {'Degree' 'DegreeAv' 'Distance'}, ...
150     'TARGET_IDS', {group_folder_name}), ...
151     a_WU1.get('G_DICT').get('IT_LIST'), ...
152     'UniformOutput', false);
153
154 [~, group_folder_name] = fileparts(im_gr2.get('DIRECTORY'));
155 it_list2 = cellfun(@(x) NNDataPoint_Measure_CLA( ...
156     'ID', x.get('ID'), ...
157     'G', x, ...
158     'M_LIST', {'Degree' 'DegreeAv' 'Distance'}, ...
159     'TARGET_IDS', {group_folder_name}), ...
160     a_WU2.get('G_DICT').get('IT_LIST'), ...
161     'UniformOutput', false);
162
163 % create NNDataPoint_Measure_CLA DICT items
164 dp_list1 = IndexedDictionary(...
165         'IT_CLASS', 'NNDataPoint_Measure_CLA', ...
166         'IT_LIST', it_list1 ...
```

(11), (12), (13), and (14) tests various kinds of graph measure with the `MultigraphBUD`.

```
        );

...

%%% itest!
%%%% iname! (15)
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
      CLA CON XLS' filesep 'atlas.xlsx'])
    test_NNDataPoint_CON_CLA % create example files
end
example_NNCV_CON_BUD_M_CLA

%%% itest!
%%%% iname! (16)
Example script for binary undirected graph at fixed thresholds (
     MultigraphBUT) using connectivity data
%%%% icode!
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
example_NNCV_CON_BUT_M_CLA

%%% itest!
%%%% iname! (17)
Example script for binary undirected graph at fixed densities (MultigraphBUD
     ) using connectivity data
%%%% icode!
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
example_NNCV_CON_BUD_M_CLA

%%% itest!
%%%% iname! (18)
Example script for binary undirected multiplex at fixed densities (
     MultiplexBUD) using connectivity data and functional data
%%%% icode!
if ~isfile([fileparts(which('NNDataPoint_CON_FUN_MP_CLA')) filesep 'Example
     data NN CLA CON_FUN_MP XLS' filesep 'atlas.xlsx'])
    test_NNDataPoint_CON_FUN_MP_CLA % create example files
end
example_NNCV_CON_FUN_MP_BUD_M_CLA

%%% itest!
%%%% iname! (19)
Example script for binary undirected multiplex at fixed thresholds (
     MultiplexBUT) using connectivity data and functional data
%%%% icode!
if ~isfile([fileparts(which('NNDataPoint_CON_FUN_MP_CLA')) filesep 'Example
     data NN CLA CON_FUN_MP XLS' filesep 'atlas.xlsx'])
    test_NNDataPoint_CON_FUN_MP_CLA % create example files
end
example_NNCV_CON_FUN_MP_BUT_M_CLA
```

(15) tests various kinds of graph measure with the `MultigraphBUD` using example connectivity data.

(16) tests various kinds of graph measure with the `MultigraphBUT` using example data.

(17) tests various kinds of graph measure with the `MultigraphBUD` using example connectivity data.

(18) tests various kinds of graph measure with the `MultiplexBUD` using example functional data.

(19) tests various kinds of graph measure with the `MultiplexBUT` using example functional data.