Implement a new Neural Network Cross Validation

The BRAPH 2 Developers

September 19, 2023

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

Contents

Implementation of a Cross Validation for Regressors

We will start by implementing in detail `NNRegressorMLP_CrossValidation`, which is a direct extension of `NNCrossValidation`. A cross validation for multi-layer perceptron regressors (`NNRegressorMLP_CrossValidation`) is a procedure designed for evaluating multi-layer perceptron regressors using cross-validation.

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

```
%% iheader!
NNRegressorMLP_CrossValidation < NNCrossValidation (nncv, neural network cross-validation) is a process for evaluating multi-layer perceptron regressors using cross-validation. (1)

%%% idescription!
A cross validation for multi-layer perceptron regressors (NNRegressorMLP_CrossValidation) is a process that facilitates the evaluation of multi-layer perceptron regressors using cross-validation.
 It involves splitting a dataset into multiple subsets (folds), training the model on some folds while validating on others, and then repeating the process for all combinations of folds.
 This helps in assessing the generalization performance of the model and detecting overfitting.

To train all the neural networks for all folds, use: nncv.get('TRAIN')
```

(1) defines `NNRegressorMLP_CrossValidation.gen` as a subclass of `NNCrossValidation`. The moniker will be `nncv`.

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

```
%% iprops_update!

%%% iprop!
NAME (constant, string) is the name of the cross-validation.
%%%% idefault!
'NNRegressorMLP_CrossValidation'

%%% iprop!
DESCRIPTION (constant, string) is the description of the cross-validation.
%%%% idefault!
'A cross validation for multi-layer perceptron regressors (NNRegressorMLP_CrossValidation) is a process that facilitates the evaluation of multi-layer perceptron regressors using cross-validation.
 It involves splitting a dataset into multiple subsets (folds), training the model on some folds while validating on others, and then repeating the process for all combinations of folds. This helps in assessing the generalization performance of the model and detecting overfitting.'

%%% iprop!
```

```
TEMPLATE (parameter, item) is the template of the nerual cross-validation.
%%%% isettings!
'NNRegressorMLP_CrossValidation'

%%% iprop!
ID (data, string) is a few-letter code for the cross-validation.
%%%% idefault!
'NNRegressorMLP_CrossValidation ID'

%%% iprop!
LABEL (metadata, string) is an extended label of the cross-validation.
%%%% idefault!
'NNRegressorMLP_CrossValidation label'

%%% iprop!
NOTES (metadata, string) are some specific notes about the cross-validation.
%%%% idefault!
'NNRegressorMLP_CrossValidation notes'

%%% iprop!
NN_TEMPLATE (parameter, item) is the neural network template to set all
     neural network parameters.
%%%% isettings!
'NNRegressorMLP' (1)

%%% iprop!
NNEVALUATOR_TEMPLATE (parameter, item) is the neural network evaluator
     template to set all evalutor parameters.
%%%% isettings!
'NNRegressorMLP_Evaluator' (2)

%%% iprop!
NN_LIST (result, itemlist) contains the neural network models corresponding
     to k folds.
%%%% icalculate!
d_list = nncv.get('D_LIST');

if isempty(d_list)
    value = {};
else
    for i = 1:length(d_list) (3)
        d_training_set{i} = d_list;
        d_training_set{i}(i) = [];  % Exclude the i-th element
        d_training_set{i} = NNDatasetCombine('D_LIST', d_training_set{i}).
     get('D');
    end

    d_training_set = d_training_set';

    if ~isa(nncv.getr('NN_TEMPLATE'), 'NoValue')
        nn_template = nncv.get('NN_TEMPLATE'); (4)
    else
        nn_template = NNRegressorMLP( ... (5)
            'EPOCHS', nncv.get('EPOCHS'), ...
            'BATCH', nncv.get('BATCH'), ...
            'SHUFFLE', nncv.get('SHUFFLE'), ...
            'SOLVER', nncv.get('SOLVER'), ...
            'VERBOSE', nncv.get('VERBOSE'), ...
            'PLOT_TRAINING', nncv.get('PLOT_TRAINING'));
    end
```

(1) defines the neural network template to be `NNRegressorMLP`, which will be used to set the parameters for all the neural network regressors for the cross validation.

(2) defines the neural network evaluator template to be `NNRegressorMLP_Evaluator`, which will be used to set the parameters for all the neural network evaluators for the cross validation.

(3) constructs the training sets iteratively by concatenating all of the remaining folds after exlcuding the one as the validation set.

(4) and (5) set the parameters to all the `NNRegressorMLP`s based on either the `NN_TEMPLATE` or the parameters from this cross validation.

```
70
71      value = cellfun(@(d) NNRegressorMLP( ... (6)
72          'TEMPLATE', nn_template, 'D', d), ...
73          d_training_set, 'UniformOutput', false);
74  end
75
76  %%% iprop!
77  EVALUATOR_LIST (result, itemlist) contains the evaluators corresponding to k
        folds.
78  %%%% icalculate!
79  %%% iprop!
80  EVALUATOR_LIST (result, itemlist) contains the evaluators corresponding to k
        folds.
81  %%%% icalculate!
82  d_list = nncv.get('D_LIST');
83  nn_list = nncv.get('NN_LIST');
84
85  if ~isa(nncv.getr('NNEVALUATOR_TEMPLATE'), 'NoValue')
86      nne_template = nncv.get('NNEVALUATOR_TEMPLATE');(7)
87  else
88      nne_template = NNRegressorMLP_Evaluator( ...  (8)
89          'P', nncv.get('P'));
90  end
91
92  value = cellfun(@(d, nn) NNRegressorMLP_Evaluator('TEMPLATE', nne_template,
        'D', d, 'NN', nn), ... (9)
93      d_list, nn_list, 'UniformOutput', false);
```

(6) initializes all of the `NNRegressorMLPs`.

(7) and (8) sets the parameters to all the `NNRegressorMLP_Evaluators` based on either the `NNEVALUATOR_TEMPLATE` or the parameters from this cross validation.

(9) initializes all of the `NNRegressorMLP_Evaluators`.

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

```
1   %% iprops!
2
3   %%% iprop!
4   P (parameter, scalar) is the permutation number.(1)
5   %%%% idefault!
6   1e+2
7   %%%% icheck_prop!
8   check = value > 0 && value == round(value);
9
10  %%% iprop!
11  AV_CORR (result, rvector) provides the metric of the correlation of
        coefficients. (2)
12  %%%% icalculate!
13  e_list = nncv.get('EVALUATOR_LIST');
14
15  value = cellfun(@(e) e.get('CORR'), ...
16      e_list, 'UniformOutput', false);
17
18  if isempty(value)
19      value = [];
20  else
21      value = mean(cell2mat(value), 1);
22  end
23
24  %%% iprop!
```

(1) defines the number for permuation feature importance.

(2), (3), (4), and (5) calculate the average metrics, including the correlation of coefficients, the coefficient of determination, the mean absolute error, the mean squared error, and the root mean squared error.

```
AV_DET (result, rvector) provides the coefficient of determination, a
     measure showing how well the predictions are replicated by the model.
     (3)
%%%% icalculate!
e_list = nncv.get('EVALUATOR_LIST');

value = cellfun(@(e) e.get('DET'), ...
    e_list, 'UniformOutput', false);

if isempty(value)
    value = [];
else
    value = mean(cell2mat(value), 1);
end

%%% iprop!
AV_MAE (result, rvector) provides the metric of the mean absolute error. (4)
%%%% icalculate!
e_list = nncv.get('EVALUATOR_LIST');

value = cellfun(@(e) e.get('MAE'), ...
    e_list, 'UniformOutput', false);

if isempty(value)
    value = [];
else
    value = mean(cell2mat(value), 1);
end

%%% iprop!
AV_MSE (result, rvector) provides the metric of the mean squared error. (5)
%%%% icalculate!
e_list = nncv.get('EVALUATOR_LIST');

value = cellfun(@(e) e.get('MSE'), ...
    e_list, 'UniformOutput', false);

if isempty(value)
    value = [];
else
    value = mean(cell2mat(value), 1);
end

%%% iprop!
AV_RMSE (result, rvector) provides the metric of the root mean squared error
     .
%%%% icalculate!
e_list = nncv.get('EVALUATOR_LIST');

value = cellfun(@(e) e.get('RMSE'), ...
    e_list, 'UniformOutput', false);

if isempty(value)
    value = [];
else
    value = mean(cell2mat(value), 1);
end

%%% iprop!
AV_FEATURE_IMPORTANCE (result, cell) averages the feature importances across
```

k folds. (6)

```
%%%% icalculate!
e_list = nncv.get('EVALUATOR_LIST');

all_fi = cellfun(@(e) cell2mat(e.get('FEATURE_IMPORTANCE')), ...
    e_list, 'UniformOutput', false);

if isempty(cell2mat(all_fi))
    value = {};
else
    average_fi = zeros(size(all_fi{1}));
    for i = 1:numel(all_fi)
        % Add the current cell contents to the averageCell
        average_fi = average_fi + all_fi{i};
    end
    average_fi = average_fi / numel(all_fi);
    value = {average_fi};
end
```

(6) calculate the average feature importance.

Code 4: **NNRegressorMLP_CrossValidation element tests.** The `tests` section from the element generator `_NNRegressorMLP_CrossValidation.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!

%%% itest!
%%%% iname!
evaluate a regressor cross-validation with the example data
%%%% 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');

% create a item list of NNDataPoint_CON_REG
it_list = cellfun(@(x) NNDataPoint_CON_REG( ...
    'ID', x.get('ID'), ...
    'SUB', x, ...
    'TARGET_IDS', x.get('VOI_DICT').get('KEYS')), ...
    gr.get('SUB_DICT').get('IT_LIST'), ...
    'UniformOutput', false);

% create a NNDataPoint_CON_REG DICT
dp_list = IndexedDictionary(...
        'IT_CLASS', 'NNDataPoint_CON_REG', ...
        'IT_LIST', it_list ...
        );

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

kfolds = 3;
nncv = NNRegressorMLP_CrossValidation('KFOLDS', kfolds, 'D', d);
```

```
51
52 nn_list = nncv.get('NN_LIST');
53 assert(length(nn_list) == kfolds, ... (1)
54     [BRAPH2.STR ':NNRegressorMLP_CrossValidation:' BRAPH2.FAIL_TEST], ...
55     'NNRegressorMLP_CrossValidation does not calculate the neural network
        list correctly.' ...
56     )
57 e_list = nncv.get('EVALUATOR_LIST');
58 assert(length(e_list) == kfolds, ... (2)
59     [BRAPH2.STR ':NNRegressorMLP_CrossValidation:' BRAPH2.FAIL_TEST], ...
60     'NNRegressorMLP_CrossValidation does not calculate the evaluator list
        correctly.' ...
61     )
```

(1) and (2) check whether the data, regressors, and evaluators are initialized according to the user-specified number of folds.

Implementation of a Cross Validation for Classifiers

We can now use `NNRegressorMLP_CrossValidation` as the basis to implement the`NNClassifierMLP_CrossValidation`. The parts of the code that are modified are highlighted. A cross validation for multi-layer perceptron classifier (`NNClassifierMLP_CrossValidation`) is a procedure designed for evaluating multi-layer perceptron classifiers using cross-validation.

Code 5: **NNClassifierMLP_CrossValidation element header.** The `header` section of the generator code for `_NNClassifierMLP_CrossValidation.gen.m` provides the general information about the `NNClassifierMLP_CrossValidation` element.

```
%% !header!
NNClassifierMLP_CrossValidation < NNCrossValidation (nncv, neural network
    cross-validation) is a process for evaluating multi-layer perceptron
    classifiers using cross-validation.

%%% !description!
A cross validation for multi-layer perceptron classifiers (
    NNClassifierMLP_CrossValidation) is a process that facilitates the
    evaluation of multi-layer perceptron classifiers using cross-validation
    .
 It involves splitting a dataset into multiple subsets (folds), training the
     model on some folds while validating on others, and then repeating the
     process for all combinations of folds.
 This helps in assessing the generalization performance of the model and
    detecting overfitting.

To train all the neural networks for all folds, use: nncv.get('TRAIN')
```

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

```
%% !props_update!

%%% !prop!
NAME (constant, string) is the name of the cross-validation.
%%%% !default!
'NNClassifierMLP_CrossValidation'

%%% !prop!
DESCRIPTION (constant, string) is the description of the cross-validation.
%%%% !default!
'A cross validation for multi-layer perceptron classifiers (
    NNClassifierMLP_CrossValidation) is a process that facilitates the
    evaluation of multi-layer perceptron classifiers using cross-validation
    . It involves splitting a dataset into multiple subsets (folds),
    training the model on some folds while validating on others, and then
    repeating the process for all combinations of folds. This helps in
    assessing the generalization performance of the model and detecting
    overfitting.'
```

```
12
13 %%% iprop!
14 TEMPLATE (parameter, item) is the template of the cross-validation.
15 %%%% isettings!
16 'NNClassifierMLP_CrossValidation'
17
18 %%% iprop!
19 ID (data, string) is a few-letter code for the cross-validation.
20 %%%% idefault!
21 'NNClassifierMLP_CrossValidation ID'
22
23 %%% iprop!
24 LABEL (metadata, string) is an extended label of the cross-validation.
25 %%%% idefault!
26 'NNClassifierMLP_CrossValidation label'
27
28 %%% iprop!
29 NOTES (metadata, string) are some specific notes about the cross-validation.
30 %%%% idefault!
31 'NNClassifierMLP_CrossValidation notes'
32
33 %%% iprop!
34 NN_TEMPLATE (parameter, item) is the neural network template to set all
       neural network parameters.
35 %%%% isettings!
36 'NNClassifierMLP'(1)
37
38 %%% iprop!
39 NNEVALUATOR_TEMPLATE (parameter, item) is the neural network evaluator
       template to set all evalutor parameters.
40 %%%% isettings!
41 'NNClassifierMLP_Evaluator'(2)
42
43 %%% iprop!
44 NN_LIST (result, itemlist) contains the neural network models corresponding
       to k folds.
45 %%%% icalculate!
46 d_list = nncv.get('D_LIST');
47
48 if isempty(d_list)
49     value = {};
50 else
51     for i = 1:length(d_list)
52         d_training_set{i} = d_list;
53         d_training_set{i}(i) = [];  % Exclude the i-th element
54         d_training_set{i} = NNDatasetCombine('D_LIST', d_training_set{i}).
       get('D');
55     end
56
57     d_training_set = d_training_set';
58
59     if ~isa(nncv.getr('NN_TEMPLATE'), 'NoValue')
60         nn_template = nncv.get('NN_TEMPLATE');
61     else
62         nn_template = NNClassifierMLP( ...
63             'EPOCHS', nncv.get('EPOCHS'), ...
64             'BATCH', nncv.get('BATCH'), ...
65             'SHUFFLE', nncv.get('SHUFFLE'), ...
66             'SOLVER', nncv.get('SOLVER'), ...
67             'VERBOSE', nncv.get('VERBOSE'), ...
68             'PLOT_TRAINING', nncv.get('PLOT_TRAINING'));
```

(1) and (2) define the `NN_TEMPLATE` as `NNClassifierMLP` and the `NNEVALUATOR_TEMPLATE` as `NNClassifierMLP_Evaluator`.

```
    end

    value = cellfun(@(d) NNClassifierMLP( ...
        'TEMPLATE', nn_template, 'D', d), ...
        d_training_set, 'UniformOutput', false);
end

%%% iprop!
EVALUATOR_LIST (result, itemlist) contains the evaluators corresponding to k folds.
%%%% icalculate!
d_list = nncv.get('D_LIST');
nn_list = nncv.get('NN_LIST');

if ~isa(nncv.getr('NNEVALUATOR_TEMPLATE'), 'NoValue')
    nne_template = nncv.get('NNEVALUATOR_TEMPLATE');
else
    nne_template = NNClassifierMLP_Evaluator( ...
        'P', nncv.get('P'));
end

value = cellfun(@(d, nn) NNClassifierMLP_Evaluator('TEMPLATE', nne_template, 'D', d, 'NN', nn), ...
    d_list, nn_list, 'UniformOutput', false);
```

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

```
%% iprops!

%%% iprop!
P (parameter, scalar) is the permutation number.
%%%% idefault!
1e+2
%%%% icheck_prop!
check = value > 0 && value == round(value);

%%% iprop! (1)
AV_AUC (result, rvector) provides the average value of the area under the receiver operating characteristic curve across k folds.
%%%% icalculate!
e_list = nncv.get('EVALUATOR_LIST');

aucs = cellfun(@(e) e.get('AUC'), ...
    e_list, 'UniformOutput', false);

if isempty(aucs)
    value = [];
else
    value = mean(cell2mat(aucs), 1);
end

%%% iprop! (2)
AV_MACRO_AUC (result, scalar) provides the metric of the average macro AUC value across k folds.
%%%% icalculate!
e_list = nncv.get('EVALUATOR_LIST');
```

(1) and (2) calculate the average value of the area under the receiver operating characteristic curve across k folds.

```
28
29 macro_aucs = cellfun(@(e) e.get('MACRO_AUC'), ...
30     e_list, 'UniformOutput', false);
31
32 if isempty(macro_aucs)
33     value = 0;
34 else
35     value = mean(cell2mat(macro_aucs), 1);
36 end
37
38 %%% !prop! (3)
39 C_MATRIX (result, matrix) provides the confusion matrix across k folds.
40 %%%% !calculate!
41 e_list = nncv.get('EVALUATOR_LIST');
42
43 c_matrices = cellfun(@(e) e.get('C_MATRIX'), ...
44     e_list, 'UniformOutput', false);
45
46 combined_c_matrix = cellfun(@(x) double(x), c_matrices, 'UniformOutput',
       false);
47 value = sum(cat(3, combined_c_matrix{:}), 3);
48
49 %%% !prop!
50 AV_FEATURE_IMPORTANCE (result, cell) averages the feature importances across
       k folds.
51 %%%% !calculate!
52 e_list = nncv.get('EVALUATOR_LIST');
53
54 all_fi = cellfun(@(e) cell2mat(e.get('FEATURE_IMPORTANCE')), ...
55     e_list, 'UniformOutput', false);
56
57 if isempty(cell2mat(all_fi))
58     value = {};
59 else
60     average_fi = zeros(size(all_fi{1}));
61     for i = 1:numel(all_fi)
62         % Add the current cell contents to the averageCell
63         average_fi = average_fi + all_fi{i};
64     end
65     average_fi = average_fi / numel(all_fi);
66     value = {average_fi};
67 end
```

(3) aggregates the confusion matrix across k folds.

Code 8: **NNClassifierMLP_CrossValidation element tests.** The `tests` section from the element generator `_NNClassifierMLP_CrossValidation.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!

%%% itest!
%%%% iname!
evaluate a classifier cross-validation with the example data
%%%% 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');

% create item lists of NNDataPoint_CON_CLA
[~, 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( ...
    'DP_CLASS', 'NNDataPoint_CON_CLA', ...
    'DP_DICT', dp_list1 ...
    );

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

% combine the two datasets
d = NNDatasetCombine('D_LIST', {d1, d2}).get('D');

kfolds = 7;
nncv = NNClassifierMLP_CrossValidation('KFOLDS', kfolds, 'D', d);

nn_list = nncv.get('NN_LIST');
assert(length(nn_list) == kfolds, ... (1)
    [BRAPH2.STR ':NNClassifierMLP_CrossValidation:' BRAPH2.FAIL_TEST], ...
    'NNClassifierMLP_CrossValidation does not calculate the neural network
     list correctly.' ...
    )
e_list = nncv.get('EVALUATOR_LIST');
assert(length(e_list) == kfolds, ... (2)
    [BRAPH2.STR ':NNClassifierMLP_CrossValidation:' BRAPH2.FAIL_TEST], ...
    'NNClassifierMLP_CrossValidation does not calculate the evaluator list
     correctly.' ...
    )
```

(1) and (2) check whether the data, classifiers, and evaluators are initialized according to the user-specified number of folds.