

# Package ‘bnstruct’

May 19, 2021

**Description** Bayesian Network Structure Learning from Data with Missing Values.

The package implements the Silander-Myllymaki complete search, the Max-Min Parents-and-Children, the Hill-Climbing, the Max-Min Hill-climbing heuristic searches, and the Structural Expectation-Maximization algorithm. Available scoring functions are BDeu, AIC, BIC. The package also implements methods for generating and using bootstrap samples, imputed data, inference.

**Type** Package

**Title** Bayesian Network Structure Learning from Data with Missing Values

**Version** 1.0.11

**Date** 2021-05-19

**Depends** R (>= 3.5.0), bitops, igraph, methods

**Suggests** graph, Rgraphviz, qgraph, knitr, testthat

**License** GPL (>= 2) | file LICENSE

**Encoding** UTF-8

**RoxygenNote** 7.1.0

**VignetteBuilder** knitr

**NeedsCompilation** yes

**Author** Francesco Sambo [aut],  
Alberto Franzin [aut, cre]

**Maintainer** Alberto Franzin <afranzin@ulb.ac.be>

**Repository** CRAN

**Date/Publication** 2021-05-19 08:10:02 UTC

## R topics documented:

add.observations<-	4
asia	5
asia_10000	5
asia_2_layers	6

belief.propagation . . . . .	7
bn . . . . .	8
BN-class . . . . .	8
bn<- . . . . .	10
BNDataset-class . . . . .	10
boot . . . . .	13
boots . . . . .	14
boots<- . . . . .	14
bootstrap . . . . .	15
build.junction.tree . . . . .	15
child . . . . .	16
child_NA_5000 . . . . .	17
complete . . . . .	17
cpts . . . . .	18
cpts<- . . . . .	19
dag . . . . .	19
dag.to.cpdag . . . . .	20
dag<- . . . . .	21
data.file . . . . .	21
data.file<- . . . . .	22
discreteness . . . . .	22
discreteness<- . . . . .	23
edge.dir.wpdag . . . . .	23
em . . . . .	24
get.most.probable.values . . . . .	25
has.boots . . . . .	26
has.imputed.boots . . . . .	26
has.imputed.data . . . . .	27
has.raw.data . . . . .	28
header.file . . . . .	28
header.file<- . . . . .	29
imp.boots . . . . .	30
imp.boots<- . . . . .	30
impute . . . . .	31
imputed.data . . . . .	31
imputed.data<- . . . . .	32
InferenceEngine-class . . . . .	32
interventions . . . . .	34
interventions<- . . . . .	35
jpts . . . . .	35
jpts<- . . . . .	36
jt.cliques . . . . .	37
jt.cliques<- . . . . .	37
junction.tree . . . . .	38
junction.tree<- . . . . .	38
knn.impute . . . . .	39
layering . . . . .	40
learn.dynamic.network . . . . .	40

learn.network	43
learn.params	46
learn.structure	47
marginals	51
name	52
name<-	52
node.sizes	53
node.sizes<-	53
num.boots	54
num.boots<-	54
num.items	55
num.items<-	55
num.nodes	56
num.nodes<-	56
num.time.steps	57
num.time.steps<-	57
num.variables	58
num.variables<-	58
observations	59
observations<-	59
plot	60
print	61
quantiles	62
quantiles<-	62
raw.data	63
raw.data<-	64
read.bif	64
read.dataset	65
read.dsc	67
read.net	67
sample.dataset	68
sample.row	68
save.to.eps	69
scoring.func	70
scoring.func<-	70
shd	71
show	71
struct.algo	72
struct.algo<-	72
test.updated.bn	73
tune.knn.impute	74
updated.bn	74
updated.bn<-	75
variables	75
variables<-	76
wpdag	76
wpdag.from.dag	77
wpdag<-	78

<code>write.dsc</code> . . . . .	78
<code>write_xgmml</code> . . . . .	79

<b>Index</b>	<b>80</b>
--------------	-----------

---

`add.observations<-`     *add further evidence to an existing list of observations of an [InferenceEngine](#).*

---

### Description

Add a list of observations to an `InferenceEngine` that already has observations, using a list composed by the two following vectors:

- `observed.vars` vector of observed variables;
- `observed.vals` vector of values observed for the variables in `observed.vars` in the corresponding position.

### Usage

```
add.observations(x) <- value
```

```
## S4 replacement method for signature 'InferenceEngine'
add.observations(x) <- value
```

### Arguments

<code>x</code>	an <a href="#">InferenceEngine</a> .
<code>value</code>	the list of observations of the <a href="#">InferenceEngine</a> .

### Details

In case of multiple observations of the same variable, the last observation is the one used, as the most recent.

### See Also

[observations<-](#)

---

asia	<i>load Asia dataset.</i>
------	---------------------------

---

**Description**

Wrapper for a loader for the Asia dataset, with only raw data.

**Usage**

```
asia()
```

**Details**

The dataset has 10000 items, no missing data, so no imputation needs to be performed.

**Value**

a `BNDataset` containing the Child dataset.

**See Also**

[asia\\_10000](#)

**Examples**

```
dataset <- asia()
print(dataset)
```

---

asia_10000	<i>Asia dataset.</i>
------------	----------------------

---

**Description**

The Asia dataset contains 10000 complete (no missing data, no latent variables) randomly generated items of the Asia Bayesian Network. No imputation needs to be performed, so only raw data is present.

**Format**

a `BNDataset` with raw data slow filled.

**Details**

The data the `BNDataset` object is built from is located in files `pkg_folder/extdata/asia_10000.header` and `pkg_folder/extdata/asia_10000.data`.

## References

S. Lauritzen, D. Spiegelhalter. Local Computation with Probabilities on Graphical Structures and their Application to Expert Systems (with discussion). *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 50(2):157-224, 1988.

## See Also

[asia](#)

---

asia\_2\_layers

*load a two-layers dataset derived from the Asia dataset.*

---

## Description

Wrapper for a loader for a 2-layers dataset derived from the `Asia` dataset, with only raw data.

## Usage

```
asia_2_layers()
```

## Details

The dataset has 100 items, no missing data, so no imputation needs to be performed.

## Value

a `BNDataset` containing the `Child` dataset.

## See Also

[asia\\_10000](#)

## Examples

```
dataset <- asia_2_layers()
print(dataset)
```

---

belief.propagation     *perform belief propagation.*

---

## Description

Perform belief propagation for the network of an `InferenceEngine`, given a set of observations. In the current version of `bnstruct`, belief propagation can be computed only over a junction tree.

## Usage

```
belief.propagation(ie, observations = NULL, return.potentials = FALSE)
```

```
## S4 method for signature 'InferenceEngine'
```

```
belief.propagation(ie, observations = NULL, return.potentials = FALSE)
```

## Arguments

`ie`                    an `InferenceEngine` object.

`observations`        list of observations, consisting in two vector, `observed.vars` for the observed variables, and `observed.vals` for the values taken by the variables listed in `observed.vars`. If no observations are provided, the `InferenceEngine` will use the ones it already contains.

`return.potentials`    if TRUE only the potentials are returned, instead of the default `BN`.

## Value

updated `InferenceEngine` object.

## Examples

```
## Not run:
dataset <- BNdataset("file.header", "file.data")
bn <- BN(dataset)
ie <- InferenceEngine(bn)
ie <- belief.propagation(ie)

observations(ie) <- list("observed.vars"=("A","G","X"), "observed.vals"=c(1,2,1))
belief.propagation(ie)

## End(Not run)
```

---

bn	<i>get the <a href="#">BN</a> object contained in an <a href="#">InferenceEngine</a>.</i>
----	---

---

**Description**

Return a network contained in an [InferenceEngine](#).

**Usage**

```
bn(x)

## S4 method for signature 'InferenceEngine'
bn(x)
```

**Arguments**

x                    an [InferenceEngine](#).

**Value**

the [BN](#) object contained in an [InferenceEngine](#).

---

BN-class	<i>BN class definition.</i>
----------	-----------------------------

---

**Description**

Instantiate a [BN](#) object.

**Usage**

```
## S4 method for signature 'BN'
initialize(.Object, dataset = NULL, ...)

BN(dataset = NULL, ...)
```

**Arguments**

.Object	a <a href="#">BN</a>
dataset	a <a href="#">BNDataset</a> object containing the dataset the network is built upon, if any. The remaining parameters are considered only if a starting dataset is provided.
...	potential further arguments of methods.



## Details

The constructor may be invoked without parameters – in this case an empty network will be created, and its slots will be filled manually by the user. This is usually viable only if the user already has knowledge about the network structure.

## Value

BN object.

## Slots

`name`: name of the network

`num.nodes`: number of nodes in the network

`variables`: names of the variables in the network

`discreteness`: TRUE if variable is discrete, FALSE if variable is continue

`node.sizes`: if variable `i` is discrete, `node.sizes[i]` contains the cardinality of `i`, if `i` is instead discrete the value is the number of states variable `i` takes when discretized

`cpts`: list of conditional probability tables of the network

`dag`: adjacency matrix of the network

`wpdag`: weighted partially dag

`scoring.func`: scoring function used in structure learning (when performed)

`struct.algo`: algorithm used in structure learning (when performed)

`num.time.steps`: number of instants in which the network is observed (1, unless it is a Dynamic Bayesian Network)

`discreteness`: TRUE if variable is discrete, FALSE if variable is continue

## Examples

```
## Not run:
net.1 <- BN()

dataset <- BNdataset()
dataset <- read.dataset(dataset, "file.header", "file.data")
net.2 <- BN(dataset)

## End(Not run)
```

---

```
bn<-                set the original BN object contained in an InferenceEngine.
```

---

### Description

Add an original network to an InferenceEngine.

### Usage

```
bn(x) <- value

## S4 replacement method for signature 'InferenceEngine'
bn(x) <- value
```

### Arguments

x	an <a href="#">InferenceEngine</a> .
value	the BN object contained in an <a href="#">InferenceEngine</a> .

---

```
BNDataset-class    BNDataset class.
```

---

### Description

Contains the all of the data that can be extracted from a given dataset: raw data, imputed data, raw and imputed data with bootstrap.

### Usage

```
BNDataset(data, discreteness, variables = NULL, node.sizes = NULL, ...)

## S4 method for signature 'BNDataset'
initialize(.Object)
```

### Arguments

.Object	an empty BNDataset.
data	raw data.frame or path/name of the file containing the raw dataset (see 'Details').
discreteness	a vector of booleans indicating if the variables are discrete or continuous (TRUE and FALSE, respectively), or path/name of the file containing header information for the dataset (discreteness, variable names, cardinality - see 'Details').
variables	vector of variable names.
node.sizes	vector of variable cardinalities (for discrete variables) or quantization ranges (for continuous variables).
...	further arguments for reading a dataset from files (see documentation for read.dataset).

## Details

There are two ways to build a BNDataset: using two files containing respectively header informations and data, and manually providing the data table and the related header informations (variable names, cardinality and discreteness).

The key informations needed are: 1. the data; 2. the state of variables (discrete or continuous); 3. the names of the variables; 4. the cardinalities of the variables (if discrete), or the number of levels they have to be quantized into (if continuous). Names and cardinalities/leves can be guessed by looking at the data, but it is strongly advised to provide `_all_` of the informations, in order to avoid problems later on during the execution.

Data can be provided in form of `data.frame` or matrix. It can contain NAs. By default, NAs are indicated with `'?'`; to specify a different character for NAs, it is possible to provide also the `na.string.symbol` parameter. The values contained in the data have to be numeric (real for continuous variables, integer for discrete ones). The default range of values for a discrete variable  $X$  is  $[1, |X|]$ , with  $|X|$  being the cardinality of  $X$ . The same applies for the levels of quantization for continuous variables. If the value ranges for the data are different from the expected ones, it is possible to specify a different starting value (for the whole dataset) with the `starts.from` parameter. E.g. by `starts.from=0` we assume that the values of the variables in the dataset have range  $[0, |X|-1]$ . Please keep in mind that the internal representation of `bnstruct` starts from 1, and the original starting values are then lost.

It is possible to use two files, one for the data and one for the metadata, instead of providing manually all of the info. `bnstruct` requires the data files to be in a format subsequently described. The actual data has to be in (a text file containing data in) tabular format, one tuple per row, with the values for each variable separated by a space or a tab. Values for each variable have to be numbers, starting from 1 in case of discrete variables. Data files can have a first row containing the names of the corresponding variables.

In addition to the data file, a header file containing additional informations can also be provided. An header file has to be composed by three rows of tab-delimited values: 1. list of names of the variables, in the same order of the data file; 2. a list of integers representing the cardinality of the variables, in case of discrete variables, or the number of levels each variable has to be quantized in, in case of continuous variables; 3. a list that indicates, for each variable, if the variable is continuous (c or C), and thus has to be quantized before learning, or discrete (d or D). In case of need of more advanced options when reading a dataset from files, please refer to the documentation of the `read.dataset` method. Imputation and bootstrap are also available as separate routines (`impute` and `bootstrap`, respectively).

In case of an evolving system to be modeled as a Dynamic Bayesian Network, it is possible to specify only the description of the variables of a single instant; the information will be replicated for all the `num.time.steps` instants that compose the dataset, where `num.time.steps` needs to be set as parameter. In this case, it is assumed that the  $N$  variables  $v_1, v_2, \dots, v_N$  of a single instant appear in the dataset as  $v_1_{t1}, v_2_{t1}, \dots, v_N_{t1}, v_1_{t2}, v_2_{t2}, \dots$ , in this exact order. The user can however provide information for all the variables in all the instants; if it is not the case, the name of the variables will be edited to include the instant. In case of an evolving system, the `num.variables` slots refers anyway to the total number of variables observed in all the instants (the number of columns in the dataset), and not to a single instant.

## Value

BNDataset object.

a BNDataset object.

### Slots

**name:** name of the dataset  
**header.file:** name and location of the header file  
**data.file:** name and location of the data file  
**variables:** names of the variables in the network  
**node.sizes:** cardinality of each variable of the network  
**num.variables:** number of variables (columns) in the dataset  
**discreteness:** TRUE if variable is discrete, FALSE if variable is continue  
**quantiles:** list of vectors containing the quantiles, one vector per variable. Each vector is NULL if the variable is discrete, and contains the quantiles if it is continuous  
**num.items:** number of observations (rows) in the dataset  
**has.raw.data:** TRUE if the dataset contains data read from a file  
**has.imputed.data:** TRUE if the dataset contains imputed data (computed from raw data)  
**raw.data:** matrix containing raw data  
**imputed.data:** matrix containing imputed data  
**has.boots:** dataset has bootstrap samples  
**boots:** list of bootstrap samples  
**has.imputed.boots:** dataset has imputed bootstrap samples  
**imp.boots:** list of imputed bootstrap samples  
**num.boots:** number of bootstrap samples  
**num.time.steps:** number of instants in which the network is observed (1, unless it is a dynamic system)

### See Also

read.dataset, impute, bootstrap

### Examples

```

## Not run:
# create from files
dataset <- BNDataset("file.data", "file.header")

# other way: create from raw dataset and metadata
data <- matrix(c(1:16), nrow = 4, ncol = 4)
dataset <- BNDataset(data = data,
                     discreteness = rep('d',4),
                     variables = c("a", "b", "c", "d"),
                     node.sizes = c(4,8,12,16))

## End(Not run)

```

---

boot	<i>get selected element of bootstrap list.</i>
------	--

---

## Description

Given a [BNDataset](#), return the sample corresponding to given index.

## Usage

```
boot(dataset, index, use.imputed.data = FALSE)
```

```
## S4 method for signature 'BNDataset,numeric'  
boot(dataset, index, use.imputed.data = FALSE)
```

## Arguments

dataset	a <a href="#">BNDataset</a> object.
index	the index of the requested sample.
use.imputed.data	TRUE if samples from imputed dataset are to be used. Default if FALSE.

## See Also

bootstrap  
[bootstrap](#)

## Examples

```
## Not run:  
dataset <- BNDataset("file.data", "file.header")  
dataset <- bootstrap(dataset, num.boots = 1000)  
  
for (i in 1:num.boots(dataset))  
  print(boot(dataset, i))  
  
## End(Not run)
```

---

boots	<i>get list of bootstrap samples of a <a href="#">BNDataset</a>.</i>
-------	--

---

**Description**

Return the list of samples computed from raw data of a dataset.

**Usage**

```
boots(x)

## S4 method for signature 'BNDataset'
boots(x)
```

**Arguments**

x                    a [BNDataset](#) object.

**Value**

the list of bootstrap samples.

**See Also**

[has.boots](#), [has.imputed.boots](#), [imp.boots](#)

---

boots<-	<i>set list of bootstrap samples of a <a href="#">BNDataset</a>.</i>
---------	--

---

**Description**

Add to a dataset a list of samples from raw data computed using bootstrap.

**Usage**

```
boots(x) <- value

## S4 replacement method for signature 'BNDataset'
boots(x) <- value
```

**Arguments**

x                    a [BNDataset](#) object.  
value                the list of bootstrap samples.

---

bootstrap	<i>Perform bootstrap.</i>
-----------	---------------------------

---

**Description**

Create a list of num.boots samples of the original dataset.

**Usage**

```
bootstrap(object, num.boots = 100, seed = 0, imputation = FALSE, k.impute = 10)

## S4 method for signature 'BNDataset'
bootstrap(object, num.boots = 100, seed = 0, imputation = FALSE, k.impute = 10)
```

**Arguments**

object	the <a href="#">BNDataset</a> object.
num.boots	number of sampled datasets for bootstrap.
seed	random seed.
imputation	TRUE if imputation has to be performed. Default is FALSE.
k.impute	number of neighbours to be used; for discrete variables we use mode, for continuous variables the median value is instead taken (useful only if imputation == TRUE).

**Examples**

```
## Not run:
dataset <- BNDataset("file.data", "file.header")
dataset <- bootstrap(dataset, num.boots = 1000)

## End(Not run)
```

---

build.junction.tree	<i>build a JunctionTree.</i>
---------------------	------------------------------

---

**Description**

Starting from the adjacency matrix of the directed acyclic graph of the network contained in an InferenceEngine, build a JunctionTree for the network and store it into an InferenceEngine.

**Usage**

```
build.junction.tree(object, ...)

## S4 method for signature 'InferenceEngine'
build.junction.tree(object, ...)
```

**Arguments**

object            an [InferenceEngine](#) object.  
...               potential further arguments for methods.

**See Also**

[InferenceEngine](#)

**Examples**

```
## Not run:  
dataset <- BNdataset("file.header", "file.data")  
net <- BN(dataset)  
eng <- InferenceEngine()  
eng <- build.junction.tree(eng)  
  
## End(Not run)
```

---

child	<i>load Child dataset.</i>
-------	----------------------------

---

**Description**

Wrapper for a loader for the Child raw dataset; also perform imputation.

**Usage**

```
child()
```

**Details**

The dataset has 5000 items, with random missing values (no latent variables). `BNdataset` object contains the raw dataset and imputed dataset, with  $k=10$  (see [impute](#) for related explanation).

**Value**

a `BNdataset` containing the Child dataset.

**See Also**

[child\\_NA\\_5000](#)

**Examples**

```
dataset <- child()  
print(dataset)
```



---

child_NA_5000	Child <i>dataset</i> .
---------------	------------------------

---

### Description

The Child dataset contains 5000 randomly generated items with missing data (no latent variables) of the Child Bayesian Network. Imputation is performed, so both raw and imputed data is present.

### Format

a [BNDataset](#) with a raw and imputed data slow filled with 5000 items.

### Details

The data the BNDataset object is built from is located in files `pkg_folder/extdata/extdata/Child_data_na_5000.head` and `pkg_folder/extdata/extdata/Child_data_na_5000.data`.

### References

D. J. Spiegelhalter, R. G. Cowell (1992). Learning in probabilistic expert systems. In Bayesian Statistics 4 (J. M. Bernardo, J. O. Berger, A. P. Dawid and A. F. M. Smith, eds.) 447-466. Clarendon Press, Oxford.

### See Also

[child](#)

---

complete	<i>Subset a <a href="#">BNDataset</a> to get only complete cases.</i>
----------	---

---

### Description

Given a [BNDataset](#), return a copy of the original object where the `raw.data` consists only in the observations that do not contain missing values.

### Usage

```
complete(x, complete.vars = seq_len(num.variables(x)))

## S4 method for signature 'BNDataset'
complete(x, complete.vars = seq_len(num.variables(x)))
```

### Arguments

<code>x</code>	a <a href="#">BNDataset</a> .
<code>complete.vars</code>	vector containing the indices of the variables to be considered for the subsetting; variables not included in the vector can still contain NAs.

**Details**

Non-missingness can be required on a subset of variables (by default, on all variables).

If present, imputed data and bootstrap samples are eliminated from the new `BNDataset`, as using this method *after* using `impute` or `bootstrap`, there may likely be a loss of correspondence between the subsetting `raw.data` and the previously generated `imputed.data` and bootstrap samples.

**Value**

a copy of the original `BNDataset` containing only complete observations.

---

cpts	<i>get the list of conditional probability tables of a <code>BN</code>.</i>
------	---

---

**Description**

Return the list of conditional probability tables of the variables of a `BN` object. Each probability table is associated to the corresponding variable, and its dimensions are named according to the variable they represent.

**Usage**

```
cpts(x)

## S4 method for signature 'BN'
cpts(x)
```

**Arguments**

`x` an object.

**Details**

Each conditional probability table is represented as a multidimensional array. The ordering of the dimensions of each variable is not guaranteed to follow the actual conditional distribution. E.g. dimensions for conditional probability  $P(C|A,B)$  can be either  $(C,A,B)$  or  $(A,B,C)$ , depending on if some operations have been performed, or how the probability table has been computed. Users should not rely on dimension numbers, but should instead select the dimensions using their names.

**Value**

list of the conditional probability tables of the desired object.

---

cpts<- *set the list of conditional probability tables of a network.*

---

**Description**

Set the list of conditional probability tables of a [BN](#) object.

**Usage**

```
cpts(x) <- value

## S4 replacement method for signature 'BN'
cpts(x) <- value
```

**Arguments**

x                    an object.  
value                list of the conditional probability tables of the object.

**Details**

Each conditional probability table is represented as a multidimensional array. To retrieve single dimensions (e.g. to compute marginals), users should provide dimensions names.

---

dag *get adjacency matrix of a network.*

---

**Description**

Return the adjacency matrix of the directed acyclic graph representing the structure of a network.

**Usage**

```
dag(x)

## S4 method for signature 'BN'
dag(x)
```

**Arguments**

x                    an object.

**Value**

matrix containing the adjacency matrix of the directed acyclic graph representing the structure of the object.

---

dag.to.cpdag	<i>convert a DAG to a CPDAG</i>
--------------	---------------------------------

---

### Description

Convert the adjacency matrix representing the DAG of a [BN](#) into the adjacency matrix representing a CPDAG for the network.

### Usage

```
dag.to.cpdag(dag.adj.matrix, layering = NULL, layer.struct = NULL)
```

### Arguments

`dag.adj.matrix` the adjacency matrix representing the DAG of a [BN](#).

`layering` vector containing the layers where each node belongs.

`layer.struct` layer.struct 0/1 matrix for indicating which layers can contain parent nodes for nodes in a layer (only for `mmhc`, `mmpc`).

### Value

the adjacency matrix representing a CPDAG for the network.

### See Also

[wpdag.from.dag](#)

### Examples

```
## Not run:  
net <- learn.network(dataset, layering=layering, layer.struct=layer.struct)  
pdag <- dag.to.cpdag(dag(net), layering, layer.struct)  
wpdag(net) <- pdag  
  
## End(Not run)
```

---

dag<-                    *set adjacency matrix of an object.*

---

### Description

Set the adjacency matrix of the directed acyclic graph representing the structure of a network.

### Usage

```
dag(x) <- value

## S4 replacement method for signature 'BN'
dag(x) <- value
```

### Arguments

x	an object.
value	matrix containing the adjacency matrix of the directed acyclic graph representing the structure of the object.

---

data.file                *get data file of a [BNDataset](#).*

---

### Description

Return the data filename of a dataset (with the path to its position, as given by the user). The data filename may contain a header in the first row, containing the list of names of the variables, in the same order as in the header file. After the header, if present, the file contains a data.frame with the observations, one item per row.

### Usage

```
data.file(x)

## S4 method for signature 'BNDataset'
data.file(x)
```

### Arguments

x	a <a href="#">BNDataset</a> .
---	-------------------------------

### Value

data filename of the dataset.

### See Also

[data.file](#)

---

<code>data.file&lt;-</code>	<i>set data file of a <a href="#">BNDataset</a>.</i>
-----------------------------	--

---

**Description**

Set the data filename of a dataset (with the path to its position, as given by the user). The data filename may contain a header in the first row, containing the list of names of the variables, in the same order as in the header file. After the header, if present, the file contains a `data.frame` with the observations, one item per row.

**Usage**

```
data.file(x) <- value
```

```
## S4 replacement method for signature 'BNDataset'
data.file(x) <- value
```

**Arguments**

<code>x</code>	a <a href="#">BNDataset</a> .
<code>value</code>	data filename.

**See Also**

[header.file<-](#)

---

<code>discreteness</code>	<i>get status (discrete or continuous) of the variables of an object.</i>
---------------------------	---

---

**Description**

Get a vector representing the status of the variables (with their names) of a [BN](#) or [BNDataset](#). Elements of the vector are `c` if the variable is continue, and `d` if the variable is discrete.

**Usage**

```
discreteness(x)
```

```
## S4 method for signature 'BN'
discreteness(x)
```

```
## S4 method for signature 'BNDataset'
discreteness(x)
```

**Arguments**

x                    an object.

**Value**

vector containing, for each variable of the desired object, c if the variable is continue, and d if the variable is discrete.

---

discreteness<-                    *set status (discrete or continuous) of the variables of an object.*

---

**Description**

Set the list of variable status for the variables in a network or a dataset.

**Usage**

```
discreteness(x) <- value

## S4 replacement method for signature 'BN'
discreteness(x) <- value

## S4 replacement method for signature 'BNdataset'
discreteness(x) <- value
```

**Arguments**

x                    an object.  
value                a vector of elements in {c,d} for continuous and discrete variables (respectively).

---

edge.dir.wpdag                    *counts the edges in a WPDAG with their directionality*

---

**Description**

Given a BN with a WPDAG, it counts the edges, with their directionality.

**Usage**

```
edge.dir.wpdag(x, use.node.names = TRUE)
```

**Arguments**

x                    the BN  
use.node.names    use node names rather than number (TRUE by default).

**Value**

a matrix containing the node pairs with the count of the edges between them in the WPDAG.

---

em	<i>expectation-maximization algorithm.</i>
----	--

---

**Description**

Learn parameters of a network using the Expectation-Maximization algorithm.

**Usage**

```
em(x, dataset, threshold = 0.001, max.em.iterations = 10, ess = 1)

## S4 method for signature 'InferenceEngine,BNDataset'
em(x, dataset, threshold = 0.001, max.em.iterations = 10, ess = 1)
```

**Arguments**

x	an <a href="#">InferenceEngine</a> .
dataset	observed dataset with missing values for the Bayesian Network of x.
threshold	threshold for convergence, used as stopping criterion.
max.em.iterations	maximum number of iterations to run in case of no convergence.
ess	Equivalent Sample Size value.

**Value**

a list containing: an [InferenceEngine](#) with a new updated network ("InferenceEngine"), and the imputed dataset ("BNDataset").

**Examples**

```
## Not run:
em(x, dataset)

## End(Not run)
```



---

`get.most.probable.values`*compute the most probable values to be observed.*

---

## Description

Return an array containing the values that each variable of the network is more likely to take, according to the CPTS. In case of ties take the first value.

## Usage

```
get.most.probable.values(x, prev.values = NULL)
```

```
## S4 method for signature 'BN'  
get.most.probable.values(x, prev.values = NULL)
```

```
## S4 method for signature 'InferenceEngine'  
get.most.probable.values(x, prev.values = NULL)
```

## Arguments

<code>x</code>	a <a href="#">BN</a> or <a href="#">InferenceEngine</a> object.
<code>prev.values</code>	vector of size <code>num.nodes</code> containing an existing observation to complete (NULL if none).

## Value

array containing, in each position, the most probable value for the corresponding variable.

## Examples

```
## Not run:  
# try with a BN object x  
get.most.probable.values(x)  
  
# now build an InferenceEngine object  
eng <- InferenceEngine(x)  
get.most.probable.values(eng)  
  
## End(Not run)
```

---

has.boots	<i>check whether a <a href="#">BNDataset</a> has bootstrap samples or not.</i>
-----------	--

---

**Description**

Return TRUE if the given dataset contains samples for bootstrap, FALSE otherwise.

**Usage**

```
has.boots(x)
```

```
## S4 method for signature 'BNDataset'
has.boots(x)
```

**Arguments**

x                    a [BNDataset](#) object.

**Value**

TRUE if dataset has bootstrap samples.

**See Also**

[has.imputed.boots](#), [boots](#), [imp.boots](#)

---

has.imputed.boots	<i>check whether a <a href="#">BNDataset</a> has bootstrap samples from imputed data or not.</i>
-------------------	--

---

**Description**

Return TRUE if the given dataset contains samples for bootstrap from imputed dataset, FALSE otherwise.

**Usage**

```
has.imputed.boots(x)
```

```
## S4 method for signature 'BNDataset'
has.imputed.boots(x)
```

**Arguments**

x                    a [BNDataset](#) object.

**Value**

TRUE if dataset has bootstrap samples from imputed data.

**See Also**

[has.boots](#), [boots](#), [imp.boots](#)

---

has.imputed.data	<i>check if a BNDataset contains imputed data.</i>
------------------	--

---

**Description**

Check whether a [BNDataset](#) object actually contains imputed data.

**Usage**

```
has.imputed.data(x)
```

```
## S4 method for signature 'BNDataset'  
has.imputed.data(x)
```

**Arguments**

x                    a [BNDataset](#).

**See Also**

[has.raw.data](#), [raw.data](#), [imputed.data](#)

**Examples**

```
## Not run:  
x <- BNDataset()  
has.imputed.data(x) # FALSE  
  
x <- read.dataset(x, "file.header", "file.data")  
has.imputed.data(x) # FALSE, since read.dataset() actually reads raw data.  
  
x <- impute(x)  
has.imputed.data(x) # TRUE  
  
## End(Not run)
```

---

has.raw.data	<i>check if a <code>BNDataset</code> contains raw data.</i>
--------------	---

---

### Description

Check whether a `BNDataset` object actually contains raw data.

### Usage

```
has.raw.data(x)

## S4 method for signature 'BNDataset'
has.raw.data(x)
```

### Arguments

`x` a `BNDataset`.

### See Also

[has.imputed.data](#), [raw.data](#), [imputed.data](#)

### Examples

```
## Not run:
x <- BNDataset()
has.raw.data(x) # FALSE

x <- read.dataset(x, "file.header", "file.data")
has.raw.data(x) # TRUE, since read.dataset() actually reads raw data.

## End(Not run)
```

---

header.file	<i>get header file of a <code>BNDataset</code>.</i>
-------------	---

---

### Description

Return the header filename of a dataset (with the path to its position, as given by the user), present if the dataset has been read from a file and not manually inserted. The header file contains three rows:

1. list of names of the variables, in the same order as in the data file;
2. list of cardinalities of the variables, if discrete, or levels for quantization if continuous;
3. list of status of the variables: c for continuous variables, d for discrete ones.

**Usage**

```
header.file(x)

## S4 method for signature 'BNDataset'
header.file(x)
```

**Arguments**

x                    a [BNDataset](#).

**Value**

header filename of the dataset.

**See Also**

[data.file](#)

---

`header.file<-`                    *set header file of a [BNDataset](#).*

---

**Description**

Set the header filename of a dataset (with the path to its position, as given by the user). The header file has to contain three rows:

1. list of names of the variables, in the same order as in the data file;
2. list of cardinalities of the variables, if discrete, or levels for quantization if continuous;
3. list of status of the variables: c for continuous variables, d for discrete ones.

Further rows are ignored.

**Usage**

```
header.file(x) <- value

## S4 replacement method for signature 'BNDataset'
header.file(x) <- value
```

**Arguments**

x                    a [BNDataset](#).  
value                header filename.

**See Also**

[data.file<-](#)

---

imp.boots	<i>get list of bootstrap samples from imputed data of a <a href="#">BNDataset</a>.</i>
-----------	--

---

**Description**

Return the list of samples computed from raw data of a dataset.

**Usage**

```
imp.boots(x)

## S4 method for signature 'BNDataset'
imp.boots(x)
```

**Arguments**

x                    a [BNDataset](#) object.

**Value**

the list of bootstrap samples from imputed data.

**See Also**

[has.boots](#), [has.imputed.boots](#), [boots](#)

---

imp.boots<-	<i>set list of bootstrap samples from imputed data of a <a href="#">BNDataset</a>.</i>
-------------	--

---

**Description**

Add to a dataset a list of samples from imputed data computed using bootstrap.

**Usage**

```
imp.boots(x) <- value

## S4 replacement method for signature 'BNDataset'
imp.boots(x) <- value
```

**Arguments**

x                    a [BNDataset](#) object.  
value                the list of bootstrap samples from imputed data.

---

impute	<i>Impute a <a href="#">BNDataset</a> raw data with missing values.</i>
--------	---

---

### Description

Impute a [BNDataset](#) raw data with missing values.

### Usage

```
impute(object, k.impute = 10)

## S4 method for signature 'BNDataset'
impute(object, k.impute = 10)
```

### Arguments

object	the <a href="#">BNDataset</a> object.
k.impute	number of neighbours to be used; for discrete variables we use mode, for continuous variables the median value is instead taken.

### Examples

```
## Not run:
dataset <- BNDataset("file.data", "file.header")
dataset <- impute(dataset)

## End(Not run)
```

---

imputed.data	<i>get imputed data of a <a href="#">BNDataset</a>.</i>
--------------	---

---

### Description

Return imputed data contained in a [BNDataset](#) object, if any.

### Usage

```
imputed.data(x)

## S4 method for signature 'BNDataset'
imputed.data(x)
```

### Arguments

x	a <a href="#">BNDataset</a> .
---	-------------------------------

**See Also**

[has.raw.data](#), [has.imputed.data](#), [raw.data](#)

---

`imputed.data<-`            *add imputed data.*

---

**Description**

Insert imputed data in a [BNDataset](#) object.

**Usage**

```
imputed.data(x) <- value
```

```
## S4 replacement method for signature 'BNDataset'
imputed.data(x) <- value
```

**Arguments**

`x`                            a [BNDataset](#).  
`value`                        a matrix of integers containing a dataset.

**See Also**

[has.imputed.data](#), [imputed.data](#), [read.dataset](#)

---

`InferenceEngine-class`    *InferenceEngine class.*

---

**Description**

`InferenceEngine` class.

Constructor method of [InferenceEngine](#) class.

constructor for [InferenceEngine](#) object

**Usage**

```
## S4 method for signature 'InferenceEngine'
initialize(.Object, ...)
```

```
InferenceEngine(bn = NULL, observations = NULL, interventions = NULL, ...)
```



**Arguments**

<code>.Object</code>	an empty InferenceEngine object.
<code>...</code>	potential further arguments of methods.
<code>bn</code>	a <a href="#">BN</a> object.
<code>observations</code>	a list of observations composed by the two following vectors: <ul style="list-style-type: none"> <li>• <code>observed.vars</code>:vector of observed variables;</li> <li>• <code>observed.vals</code>:vector of values observed for the variables in <code>observed.vars</code> in the corresponding position.</li> </ul>
<code>interventions</code>	a list of interventions composed of the following two vectors: <ul style="list-style-type: none"> <li>• <code>intervention.vars</code>:vector of variables for which an intervention has been performed;</li> <li>• <code>intervention.vals</code>:vector of values chosen for the variables in <code>intervention.vals</code> in the corresponding position.</li> </ul>

**Value**

an InferenceEngine object.  
InferenceEngine object.

**Slots**

`junction.tree`: junction tree adjacency matrix.  
`num.nodes`: number of nodes in the junction tree.  
`cliques`: list of cliques composing the nodes of the junction tree.  
`triangulated.graph`: adjacency matrix of the original triangulated graph.  
`jpts`: inferred joint probability tables.  
`bn`: original Bayesian Network (as object of class [BN](#)) as provided by the user, or learnt from a dataset. NULL if missing.  
`updated.bn`: Bayesian Network (as object of class [BN](#)) as modified by a belief propagation computation. In particular, it will have different conditional probability tables with respect to its original version. NULL if missing.  
`observed.vars`: list of observed variables, by name or number.  
`observed.vals`: list of observed values for the corresponding variables in `observed.vars`.  
`intervention.vars`: list of manipulated variables, by name or number.  
`intervention.vals`: list of specified values for the corresponding variables in `intervention.vars`.

**Examples**

```
## Not run:
dataset <- BNdataset()
dataset <- read.dataset(dataset, "file.header", "file.data")
bn <- BN(dataset)
eng <- InferenceEngine(bn)
```

```
obs <- list(c("A", "G", "X"), c(1, 2, 1))
eng.2 <- InferenceEngine(bn, obs)

## End(Not run)
```

---

interventions      *get the list of interventions of an [InferenceEngine](#).*

---

### Description

Return the list of interventions added to an [InferenceEngine](#).

### Usage

```
interventions(x)

## S4 method for signature 'InferenceEngine'
interventions(x)
```

### Arguments

x                    an [InferenceEngine](#).

### Details

Output is a list in the following format:

- `intervention.vars` vector of manipulated variables;
- `intervention.vals` vector of values for the variables in `observed.vars` in the corresponding position.

### Value

the list of interventions of the [InferenceEngine](#).

---

```
interventions<-      set the list of interventions for an InferenceEngine.
```

---

### Description

Add a list of interventions to an `InferenceEngine`, using a list composed by the two following vectors:

- `intervention.vars` vector of the variables we manipulate;
- `intervention.vals` vector of values for the variables in `observed.vars` in the corresponding position.

### Usage

```
interventions(x) <- value

## S4 replacement method for signature 'InferenceEngine'
interventions(x) <- value
```

### Arguments

`x`                    an `InferenceEngine`.

`value`                the list of interventions of the `InferenceEngine`.

### Details

An intervention can be applied only when building an `InferenceEngine`.

In case of multiple interventions of the same variable, the last intervention is the one used.

---

```
jpts                    get the list of joint probability tables compiled by an  
                          InferenceEngine.
```

---

### Description

Return the list of joint probability tables for the cliques of the junction tree obtained after belief propagation has been performed.

### Usage

```
jpts(x)

## S4 method for signature 'InferenceEngine'
jpts(x)
```

**Arguments**

x                    an [InferenceEngine](#).

**Details**

Each joint probability table is represented as a multidimensional array. To retrieve single dimensions (e.g. to compute marginals), users should not rely on dimension numbers, but should instead select the dimensions using their names.

**Value**

the list of joint probability tables compiled by the [InferenceEngine](#).

---

```
jpts<-                    set the list of joint probability tables compiled by an
                          InferenceEngine.
```

---

**Description**

Add a list of joint probability tables for the cliques of the junction tree.

**Usage**

```
jpts(x) <- value

## S4 replacement method for signature 'InferenceEngine'
jpts(x) <- value
```

**Arguments**

x                    an [InferenceEngine](#).

value                the list of joint probability tables compiled by the [InferenceEngine](#).

**Details**

Each joint probability table is represented as a multidimensional array. To retrieve single dimensions (e.g. to compute marginals), users should provide dimension names.

---

jt.cliques	<i>get the list of cliques of the junction tree of an <a href="#">InferenceEngine</a>.</i>
------------	--

---

**Description**

Return the list of cliques containing the variables associated to each node of a junction tree.

**Usage**

```
jt.cliques(x)

## S4 method for signature 'InferenceEngine'
jt.cliques(x)
```

**Arguments**

x                    an [InferenceEngine](#).

**Value**

the list of cliques of the junction tree contained in the [InferenceEngine](#).

---

jt.cliques<-	<i>set the list of cliques of the junction tree of an <a href="#">InferenceEngine</a>.</i>
--------------	--

---

**Description**

Add to the [InferenceEngine](#) a list containing the cliques of variables composing the nodes of the junction tree.

**Usage**

```
jt.cliques(x) <- value

## S4 replacement method for signature 'InferenceEngine'
jt.cliques(x) <- value
```

**Arguments**

x                    an [InferenceEngine](#).  
value                the list of cliques of the junction tree contained in the [InferenceEngine](#).

---

junction.tree	<i>get the junction tree of an <a href="#">InferenceEngine</a>.</i>
---------------	---

---

**Description**

Return the adjacency matrix representing the junction tree computed for a network.

**Usage**

```
junction.tree(x)

## S4 method for signature 'InferenceEngine'
junction.tree(x)
```

**Arguments**

x                    an [InferenceEngine](#).

**Details**

Rows and columns are named after the (variables in the) cliques that each node of the junction tree represent.

**Value**

the junction tree contained in the [InferenceEngine](#).

**See Also**

[build.junction.tree](#)

---

junction.tree<-	<i>set the junction tree of an <a href="#">InferenceEngine</a>.</i>
-----------------	---

---

**Description**

Set the adjacency matrix of the junction tree computed for a network.

**Usage**

```
junction.tree(x) <- value

## S4 replacement method for signature 'InferenceEngine'
junction.tree(x) <- value
```

**Arguments**

x	an <a href="#">InferenceEngine</a> .
value	the junction tree to be inserted in the <a href="#">InferenceEngine</a> .

---

knn.impute	<i>Perform imputation of a data frame using k-NN.</i>
------------	---

---

**Description**

Perform imputation of missing data in a data frame using the k-Nearest Neighbour algorithm. For discrete variables we use the mode, for continuous variables the median value is instead taken.

**Usage**

```
knn.impute(
  data,
  k = 10,
  cat.var = 1:ncol(data),
  to.impute = 1:nrow(data),
  using = 1:nrow(data)
)
```

**Arguments**

data	a numerical matrix.
k	number of neighbours to be used; for categorical variables the mode of the neighbours is used, for continuous variables the median value is used instead. Default: 10.
cat.var	vector containing the indices of the variables to be considered as categorical. Default: all variables.
to.impute	vector indicating which rows of the dataset are to be imputed. Default: impute all rows.
using	vector indicating which rows of the dataset are to be used to search for neighbours. Default: use all rows.

**Value**

imputed matrix.

---

layering	<i>return the layering of the nodes.</i>
----------	--

---

### Description

Compute the topological ordering of the nodes of a network, in order to divide the network in layers.

### Usage

```
layering(x)

## S4 method for signature 'BN'
layering(x)
```

### Arguments

x a [BN](#) object.

### Value

a vector containing layers the nodes can be divided into.

### Examples

```
## Not run:
dataset <- BNdataset("file.header", "file.data")
x <- BN(dataset)
x <- learn.network(x, dataset)
layering(x)

## End(Not run)
```

---

learn.dynamic.network *learn a dynamic network (structure and parameters) of a [BN](#) from a [BNdataset](#).*

---

### Description

Learn a dynamic network (structure and parameters) of a [BN](#) from a [BNdataset](#) (see the [Details](#) section). This method is a wrapper for [learn.network](#) to simplify the learning of a dynamic network. It provides an automated generation of the `layering` required to represent the set of time constraints encoded in a dynamic network. In this function, it is assumed that the dataset contains the observations for each variable in all the time steps:  $V_1^{t_1}, V_2^{t_1}, V_n^{t_1}, V_1^{t_2}, \dots, V_n^{t_k}$ . Variables in time step  $j$  can be parents for any variable in time steps  $k \geq j$ , but not for variables  $i < j$ . If a `layering` is provided for a time step, it is valid in each time step, and not throughout the whole dynamic network; a global `layering` can however be provided.



**Usage**

```
learn.dynamic.network(x, ...)

## S4 method for signature 'BN'
learn.dynamic.network(
  x,
  y = NULL,
  num.time.steps = num.time.steps(y),
  algo = "mmhc",
  scoring.func = "BDeu",
  initial.network = NULL,
  alpha = 0.05,
  ess = 1,
  bootstrap = FALSE,
  layering = c(),
  max.fanin = num.variables(y) - 1,
  max.fanin.layers = NULL,
  max.parents = num.variables(y) - 1,
  max.parents.layers = NULL,
  layer.struct = NULL,
  cont.nodes = c(),
  use.imputed.data = FALSE,
  use.cpc = TRUE,
  mandatory.edges = NULL,
  ...
)

## S4 method for signature 'BNdataset'
learn.dynamic.network(
  x,
  num.time.steps = num.time.steps(x),
  algo = "mmhc",
  scoring.func = "BDeu",
  initial.network = NULL,
  alpha = 0.05,
  ess = 1,
  bootstrap = FALSE,
  layering = c(),
  max.fanin = num.variables(x) - 1,
  max.fanin.layers = NULL,
  max.parents = num.variables(x) - 1,
  max.parents.layers = NULL,
  layer.struct = NULL,
  cont.nodes = c(),
  use.imputed.data = FALSE,
  use.cpc = TRUE,
  mandatory.edges = NULL,
  ...
)
```

)

**Arguments**

<code>x</code>	can be a <a href="#">BN</a> or a <a href="#">BNDataset</a> . If <code>x</code> is a <a href="#">BN</a> , then also the dataset parameter must be given.
<code>...</code>	potential further arguments for methods.
<code>y</code>	a <a href="#">BNDataset</a> object, to be provided only if <code>x</code> is a <a href="#">BN</a> .
<code>num.time.steps</code>	the number of time steps to be represented in the dynamic BN.
<code>algo</code>	the algorithm to use. Currently, one among <code>sm</code> (Silander-Myllymaki), <code>mmpc</code> (Max-Min Parent-and-Children), <code>mmhc</code> (Max-Min Hill Climbing, default), <code>hc</code> (Hill Climbing) and <code>sem</code> (Structural Expectation Maximization).
<code>scoring.func</code>	the scoring function to use. Currently, one among <code>BDeu</code> , <code>AIC</code> , <code>BIC</code> .
<code>initial.network</code>	network structure to be used as starting point for structure search. Can take different values: a <a href="#">BN</a> object, a matrix containing the adjacency matrix of the structure of the network, or the string <code>random.chain</code> to sample a random chain as starting point.
<code>alpha</code>	confidence threshold (only for <code>mmhc</code> ).
<code>ess</code>	Equivalent Sample Size value.
<code>bootstrap</code>	TRUE to use bootstrap samples.
<code>layering</code>	vector containing the layers each node belongs to.
<code>max.fanin</code>	maximum number of parents for each node (only for <code>hc</code> , <code>mmhc</code> ).
<code>max.fanin.layers</code>	matrix of available parents in each layer (only for <code>sm</code> – DEPRECATED, use <code>max.parents.layers</code> instead).
<code>max.parents</code>	maximum number of parents for each node (for <code>sm</code> , <code>hc</code> , <code>mmhc</code> ).
<code>max.parents.layers</code>	matrix of available parents in each layer (only for <code>sm</code> ).
<code>layer.struct</code>	0/1 matrix for indicating which layers can contain parent nodes for nodes in a layer (only for <code>mmhc</code> , <code>mmpc</code> ).
<code>cont.nodes</code>	vector containing the index of continuous variables.
<code>use.imputed.data</code>	TRUE to learn the structure from the imputed dataset (if available, a check is performed). Default is to use raw dataset
<code>use.cpc</code>	(when using <code>mmhc</code> ) compute Candidate Parent-and-Children sets instead of starting the Hill Climbing from an empty graph.
<code>mandatory.edges</code>	binary matrix, where a 1 in cell <code>[i, j]</code> indicates that an edge from node <code>i</code> to node <code>j</code> must be present in the final network.

**Details**

The other parameters available are the ones of [learn.network](#), refer to the documentation of that function for more details. See also the documentation for [learn.structure](#) and [learn.params](#) for more informations.

**Value**

new [BN](#) object with structure (DAG) and conditional probabilities as learnt from the given dataset.

**See Also**

`learn.network` `learn.structure` `learn.params`

**Examples**

```
## Not run:
mydataset <- BNdataset("data.file", "header.file")

net <- learn.dynamic.network(mydataset, num.time.steps=2)

## End(Not run)
```

---

<code>learn.network</code>	<i>learn a network (structure and parameters) of a <a href="#">BN</a> from a <a href="#">BNdataset</a>.</i>
----------------------------	---

---

**Description**

Learn a network (structure and parameters) of a [BN](#) from a [BNdataset](#) (see the `Details` section).

**Usage**

```
learn.network(x, ...)
```

```
## S4 method for signature 'BN'
learn.network(
  x,
  y = NULL,
  algo = "mmhc",
  scoring.func = "BDeu",
  initial.network = NULL,
  alpha = 0.05,
  ess = 1,
  bootstrap = FALSE,
  layering = c(),
  max.fanin = num.variables(y) - 1,
  max.fanin.layers = NULL,
  max.parents = num.variables(y) - 1,
  max.parents.layers = NULL,
  layer.struct = NULL,
  cont.nodes = c(),
  use.imputed.data = FALSE,
  use.cpc = TRUE,
```

```

    mandatory.edges = NULL,
    ...
)

## S4 method for signature 'BNdataset'
learn.network(
  x,
  algo = "mmhc",
  scoring.func = "BDeu",
  initial.network = NULL,
  alpha = 0.05,
  ess = 1,
  bootstrap = FALSE,
  layering = c(),
  max.fanin = num.variables(x) - 1,
  max.fanin.layers = NULL,
  max.parents = num.variables(x) - 1,
  max.parents.layers = NULL,
  layer.struct = NULL,
  cont.nodes = c(),
  use.imputed.data = FALSE,
  use.cpc = TRUE,
  mandatory.edges = NULL,
  ...
)

```

### Arguments

<code>x</code>	can be a <a href="#">BN</a> or a <a href="#">BNdataset</a> . If <code>x</code> is a <a href="#">BN</a> , then also the dataset parameter must be given.
<code>...</code>	potential further arguments for methods.
<code>y</code>	a <a href="#">BNdataset</a> object, to be provided only if <code>x</code> is a <a href="#">BN</a> .
<code>algo</code>	the algorithm to use. Currently, one among <code>sm</code> (Silander-Myllymaki), <code>mmpc</code> (Max-Min Parent-and-Children), <code>mmhc</code> (Max-Min Hill Climbing, default), <code>hc</code> (Hill Climbing) and <code>sem</code> (Structural Expectation Maximization).
<code>scoring.func</code>	the scoring function to use. Currently, one among <code>BDeu</code> , <code>AIC</code> , <code>BIC</code> .
<code>initial.network</code>	network structure to be used as starting point for structure search. Can take different values: a <a href="#">BN</a> object, a matrix containing the adjacency matrix of the structure of the network, or the string <code>random.chain</code> to sample a random chain as starting point.
<code>alpha</code>	confidence threshold (only for <code>mmhc</code> ).
<code>ess</code>	Equivalent Sample Size value.
<code>bootstrap</code>	<code>TRUE</code> to use bootstrap samples.
<code>layering</code>	vector containing the layers each node belongs to.
<code>max.fanin</code>	maximum number of parents for each node (only for <code>hc</code> , <code>mmhc</code> ).

<code>max.fanin.layers</code>	matrix of available parents in each layer (only for <code>sm</code> – DEPRECATED, use <code>max.parents.layers</code> instead).
<code>max.parents</code>	maximum number of parents for each node (for <code>sm</code> , <code>hc</code> , <code>mmhc</code> ).
<code>max.parents.layers</code>	matrix of available parents in each layer (only for <code>sm</code> ).
<code>layer.struct</code>	0/1 matrix for indicating which layers can contain parent nodes for nodes in a layer (only for <code>mmhc</code> , <code>mmpc</code> ).
<code>cont.nodes</code>	vector containing the index of continuous variables.
<code>use.imputed.data</code>	TRUE to learn the structure from the imputed dataset (if available, a check is performed). Default is to use raw dataset
<code>use.cpc</code>	(when using <code>mmhc</code> ) compute Candidate Parent-and-Children sets instead of starting the Hill Climbing from an empty graph.
<code>mandatory.edges</code>	binary matrix, where a 1 in cell $[i, j]$ indicates that an edge from node $i$ to node $j$ must be present in the final network.

## Details

Learn the structure (the directed acyclic graph) of a `BN` object according to a `BNdataset`. We provide five algorithms for learning the structure of the network, that can be chosen with the `algo` parameter. The first one is the Silander-Myllymäki (`sm`) exact search-and-score algorithm, that performs a complete evaluation of the search space in order to discover the best network; this algorithm may take a very long time, and can be inapplicable when discovering networks with more than 25–30 nodes. Even for small networks, users are strongly encouraged to provide meaningful parameters such as the layering of the nodes, or the maximum number of parents – refer to the documentation in package manual for more details on the method parameters.

The second method is the constraint-based Max-Min Parents-and-Children (`mmpc`), that returns the skeleton of the network. Given the possible presence of loops, due to the non-directionality of the edges discovered, no parameter learning is possible using this algorithm. Also note that in the case of a very dense network and lots of observations, the statistical evaluation of the search space may take a long time. Also for this algorithm there are parameters that may need to be tuned, mainly the confidence threshold of the statistical pruning. Please refer to the rest of this documentation for their explanation.

The third algorithm is another heuristic, the Hill-Climbing (`hc`). It can start from the complete space of possibilities (default) or from a reduced subset of possible edges, using the `cpc` argument.

The fourth algorithm (and the default one) is the Max-Min Hill-Climbing heuristic (`mmhc`), that performs a statistical sieving of the search space followed by a greedy evaluation, by combining the `MMPC` and the `HC` algorithms. It is considerably faster than the complete method, at the cost of a (likely) lower quality. As for `MMPC`, the computational time depends on the density of the network, the number of observations and the tuning of the parameters.

The fifth method is the Structural Expectation-Maximization (`sem`) algorithm, for learning a network from a dataset with missing values. It iterates a sequence of Expectation-Maximization (in order to “fill in” the holes in the dataset) and structure learning from the guessed dataset, until convergence. The structure learning used inside `SEM`, due to computational reasons, is `MMHC`. Conver-

gence of SEM can be controlled with the parameters `struct.threshold` and `param.threshold`, for the structure and the parameter convergence, respectively.

Search-and-score methods also need a scoring function to compute an estimated measure of each configuration of nodes. We provide three of the most popular scoring functions, BDeu (Bayesian-Dirichlet equivalent uniform, default), AIC (Akaike Information Criterion) and BIC (Bayesian Information Criterion). The scoring function can be chosen using the `scoring.func` parameter.

Structure learning sets the `dag` field of the BN under study, unless bootstrap or the `mmpc` algorithm are employed. In these cases, given the possible presence of loops, the `wpdag` field is set.

In case of missing data, the default behaviour (with no other indication from the user) is to learn the structure using `mmhc` starting from the raw dataset, using only the available cases with no imputation.

In case of learning from a dataset containing observations of a dynamic system, [learn.dynamic.network](#) will be employed.

Then, the parameters of the network are learnt using MAP (Maximum A Posteriori) estimation (when not using bootstrap or `mmpc`).

See documentation for [learn.structure](#) and [learn.params](#) for more informations.

### Value

new [BN](#) object with structure (DAG) and conditional probabilities as learnt from the given dataset.

### See Also

[learn.structure](#) [learn.params](#) [learn.dynamic.network](#)

### Examples

```
## Not run:
mydataset <- BNdataset("data.file", "header.file")

# starting from a BN
net <- BN(mydataset)
net <- learn.network(net, mydataset)

# start directly from the dataset
net <- learn.network(mydataset)

## End(Not run)
```

---

learn.params

*learn the parameters of a [BN](#).*

---

### Description

Learn the parameters of a [BN](#) object according to a [BNdataset](#) using MAP (Maximum A Posteriori) estimation.

**Usage**

```
learn.params(bn, dataset, ess = 1, use.imputed.data = F)

## S4 method for signature 'BN,BNDataset'
learn.params(bn, dataset, ess = 1, use.imputed.data = FALSE)
```

**Arguments**

bn	a <a href="#">BN</a> object.
dataset	a <a href="#">BNDataset</a> object.
ess	Equivalent Sample Size value.
use.imputed.data	use imputed data.

**Details**

Parameter learning is not possible in case of networks learnt using the mmpc algorithm, or from bootstrap samples, as there may be loops.

**Value**

new [BN](#) object with conditional probabilities.

**See Also**

[learn.network](#)

**Examples**

```
## Not run:
## first create a BN and learn its structure from a dataset
dataset <- BNdataset("file.header", "file.data")
bn <- BN(dataset)
bn <- learn.structure(bn, dataset)
bn <- learn.params(bn, dataset, ess=1)

## End(Not run)
```

---

learn.structure	<i>learn the structure of a network.</i>
-----------------	--

---

**Description**

Learn the structure (the directed acyclic graph) of a [BN](#) object according to a [BNDataset](#).

**Usage**

```
learn.structure(  
  bn,  
  dataset,  
  algo = "mmhc",  
  scoring.func = "BDeu",  
  initial.network = NULL,  
  alpha = 0.05,  
  ess = 1,  
  bootstrap = FALSE,  
  layering = c(),  
  max.fanin = num.variables(dataset),  
  max.fanin.layers = NULL,  
  max.parents = num.variables(dataset),  
  max.parents.layers = NULL,  
  layer.struct = NULL,  
  cont.nodes = c(),  
  use.imputed.data = FALSE,  
  use.cpc = TRUE,  
  mandatory.edges = NULL,  
  ...  
)  
  
## S4 method for signature 'BN,BNDataset'  
learn.structure(  
  bn,  
  dataset,  
  algo = "mmhc",  
  scoring.func = "BDeu",  
  initial.network = NULL,  
  alpha = 0.05,  
  ess = 1,  
  bootstrap = FALSE,  
  layering = c(),  
  max.fanin = num.variables(dataset) - 1,  
  max.fanin.layers = NULL,  
  max.parents = num.variables(dataset) - 1,  
  max.parents.layers = NULL,  
  layer.struct = NULL,  
  cont.nodes = c(),  
  use.imputed.data = FALSE,  
  use.cpc = TRUE,  
  mandatory.edges = NULL,  
  ...  
)
```



**Arguments**

<code>bn</code>	a BN object.
<code>dataset</code>	a <code>BNDataset</code> .
<code>algo</code>	the algorithm to use. Currently, one among <code>sm</code> (Silander-Myllymaki), <code>mmpc</code> (Max-Min Parent-and-Children), <code>mmhc</code> (Max-Min Hill Climbing, default), <code>hc</code> (Hill Climbing) and <code>sem</code> (Structural Expectation Maximization).
<code>scoring.func</code>	the scoring function to use. Currently, one among <code>BDeu</code> , <code>AIC</code> , <code>BIC</code> .
<code>initial.network</code>	network structure to be used as starting point for structure search. Can take different values: a BN object, a matrix containing the adjacency matrix of the structure of the network, or the string <code>random.chain</code> to sample a random chain as starting point.
<code>alpha</code>	confidence threshold (only for <code>mmhc</code> ).
<code>ess</code>	Equivalent Sample Size value.
<code>bootstrap</code>	<code>TRUE</code> to use bootstrap samples.
<code>layering</code>	vector containing the layers each node belongs to (only for <code>sm</code> ).
<code>max.fanin</code>	maximum number of parents for each node (only for <code>hc</code> , <code>mmhc</code> ).
<code>max.fanin.layers</code>	matrix of available parents in each layer (only for <code>sm</code> – DEPRECATED, use <code>max.parents.layers</code> instead).
<code>max.parents</code>	maximum number of parents for each node (for <code>sm</code> , <code>hc</code> , <code>mmhc</code> ).
<code>max.parents.layers</code>	matrix of available parents in each layer (only for <code>sm</code> ).
<code>layer.struct</code>	$\{0/1\}$ matrix for indicating which layers can contain parent nodes for nodes in a layer (only for <code>mmhc</code> , <code>mmpc</code> ).
<code>cont.nodes</code>	vector containing the index of continuous variables.
<code>use.imputed.data</code>	<code>TRUE</code> to learn the structure from the imputed dataset (if available, a check is performed). Default is to use raw dataset
<code>use.cpc</code>	(when using <code>mmhc</code> ) compute Candidate Parent-and-Children sets instead of starting the Hill Climbing from an empty graph.
<code>mandatory.edges</code>	binary matrix, where a 1 in cell $[i, j]$ indicates that an edge from node $i$ to node $j$ must be present in the final network.
<code>...</code>	potential further arguments for method.

**Details**

We provide three algorithms in order to learn the structure of the network, that can be chosen with the `algo` parameter. The first is the Silander-Myllymaki (`sm`) exact search-and-score algorithm, that performs a complete evaluation of the search space in order to discover the best network; this algorithm may take a very long time, and can be inapplicable when discovering networks with more than 25–30 nodes. Even for small networks, users are strongly encouraged to provide meaningful

parameters such as the layering of the nodes, or the maximum number of parents – refer to the documentation in package manual for more details on the method parameters.

The second method is the constraint-based Max-Min Parents-and-Children (`mmpc`), that returns the skeleton of the network. Given the possible presence of loops, due to the non-directionality of the edges discovered, no parameter learning is possible using this algorithm. Also note that in the case of a very dense network and lots of observations, the statistical evaluation of the search space may take a long time. Also for this algorithm there are parameters that may need to be tuned, mainly the confidence threshold of the statistical pruning. Please refer to the rest of this documentation for their explanation.

The third algorithm is another heuristic, the Hill-Climbing (`hc`). It can start from the complete space of possibilities (default) or from a reduced subset of possible edges, using the `cpc` argument.

The fourth algorithm (and the default one) is the Max-Min Hill-Climbing heuristic (`mmhc`), that performs a statistical sieving of the search space followed by a greedy evaluation, by combining the MMPC and the HC algorithms. It is considerably faster than the complete method, at the cost of a (likely) lower quality. As for MMPC, the computational time depends on the density of the network, the number of observations and the tuning of the parameters.

The fifth method is the Structural Expectation-Maximization (`sem`) algorithm, for learning a network from a dataset with missing values. It iterates a sequence of Expectation-Maximization (in order to “fill in” the holes in the dataset) and structure learning from the guessed dataset, until convergence. The structure learning used inside SEM, due to computational reasons, is MMHC. Convergence of SEM can be controlled with the parameters `struct.threshold` and `param.threshold`, for the structure and the parameter convergence, respectively. for learning a network from a dataset with missing values. It iterates a sequence of Expectation-Maximization (in order to “fill in” the holes in the dataset) and structure learning from the guessed dataset, until convergence. The structure learning used inside SEM, due to computational reasons, is MMHC. Convergence of SEM can be controlled with the parameters `struct.threshold` and `param.threshold`, for the structure and the parameter convergence, respectively.

Search-and-score methods also need a scoring function to compute an estimated measure of each configuration of nodes. We provide three of the most popular scoring functions, `BDeu` (Bayesian-Dirichlet equivalent uniform, default), `AIC` (Akaike Information Criterion) and `BIC` (Bayesian Information Criterion). The scoring function can be chosen using the `scoring.func` parameter.

Structure learning sets the `dag` field of the BN under study, unless bootstrap or the `mmpc` algorithm are employed. In these cases, given the possible presence of loops, the `wpdag` field is set.

In case of missing data, the default behaviour (with no other indication from the user) is to learn the structure using `mmhc` starting from the raw dataset.

### Value

new [BN](#) object with DAG.

### See Also

`learn.network` `learn.dynamic.network`

### Examples

```
## Not run:
```

```

dataset <- BNdataset("file.header", "file.data")
bn <- BN(dataset)
# use MMHC
bn <- learn.structure(bn, dataset, alpha=0.05, ess=1, bootstrap=FALSE)

# now use Silander-Myllymaki
layers <- layering(bn)
mfl <- as.matrix(read.table(header=F,
text='0 1 1 1 1 0 1 1 1 1 0 0 8 7 7 0 0 0 14 6 0 0 0 0 19'))
bn <- learn.structure(bn, dataset, algo='sm', max.fanin=3, cont.nodes=c(),
                    layering=layers, max.fanin.layers=mfl, use.imputed.data=FALSE)

## End(Not run)

```

---

marginals

*compute the list of inferred marginals of a BN.*


---

### Description

Given an [InferenceEngine](#), it returns a list containing the marginals for the variables in the network, according to the propagated beliefs.

### Usage

```

marginals(x, ...)

## S4 method for signature 'InferenceEngine'
marginals(x, ...)

```

### Arguments

x                    an [InferenceEngine](#)

...                   potential further arguments of methods.

### Value

a list containing the marginals of each variable, as probability tables.

### Examples

```

## Not run:
eng <- InferenceEngine(net)
marginals(eng)

## End(Not run)

```

---

name	<i>get name of an object.</i>
------	-------------------------------

---

**Description**

Return the name of an object, of class [BN](#) or [BNDataset](#).

**Usage**

```
name(x)

## S4 method for signature 'BN'
name(x)

## S4 method for signature 'BNDataset'
name(x)
```

**Arguments**

x                    an object.

**Value**

name of the object.

---

name<-	<i>set name of an object.</i>
--------	-------------------------------

---

**Description**

Set the name slot of an object of type [BN](#) or [BNDataset](#).

**Usage**

```
name(x) <- value

## S4 replacement method for signature 'BN'
name(x) <- value

## S4 replacement method for signature 'BNDataset'
name(x) <- value
```

**Arguments**

x                    an object.  
value                the new name of the object.

---

node.sizes	<i>get size of the variables of an object.</i>
------------	--

---

**Description**

Return a list containing the size of the variables of an object. It is the actual cardinality of discrete variables, and the cardinality of the discretized variable for continuous variables.

**Usage**

```
node.sizes(x)

## S4 method for signature 'BN'
node.sizes(x)

## S4 method for signature 'BNDataset'
node.sizes(x)
```

**Arguments**

x                    an object.

**Value**

vector containing the size of each variable of the desired object.

---

node.sizes<-	<i>set the size of variables of an object.</i>
--------------	--

---

**Description**

Set the size of the variables of a BN or BNDataset object. It represents the actual cardinality of discrete variables, and the cardinality of the discretized variable for continuous variables.

**Usage**

```
node.sizes(x) <- value

## S4 replacement method for signature 'BN'
node.sizes(x) <- value

## S4 replacement method for signature 'BNDataset'
node.sizes(x) <- value
```

**Arguments**

x                    an object.  
 value                vector containing the size of each variable of the object.

---

num.boots            *get number of bootstrap samples of a [BNDataset](#).*

---

**Description**

Return the number of bootstrap samples computed from a dataset.

**Usage**

```
num.boots(x)

## S4 method for signature 'BNDataset'
num.boots(x)
```

**Arguments**

x                    a [BNDataset](#) object.

**Value**

the number of bootstrap samples.

---

num.boots<-         *set number of bootstrap samples of a [BNDataset](#).*

---

**Description**

Set the length of the list of samples of a dataset computed using bootstrap.

**Usage**

```
num.boots(x) <- value

## S4 replacement method for signature 'BNDataset'
num.boots(x) <- value
```

**Arguments**

x                    a [BNDataset](#) object.  
 value                the number of bootstrap samples.

---

num.items	<i>get number of items of a <a href="#">BNDataset</a>.</i>
-----------	--

---

**Description**

Return the number of items in a dataset, that is, the number of rows in its data slot.

**Usage**

```
num.items(x)

## S4 method for signature 'BNDataset'
num.items(x)
```

**Arguments**

x                    a [BNDataset](#) object.

**Value**

number of items of the desired dataset.

---

num.items<-	<i>set number of items of a <a href="#">BNDataset</a>.</i>
-------------	--

---

**Description**

Set the number of observed items (rows) in a dataset.

**Usage**

```
num.items(x) <- value

## S4 replacement method for signature 'BNDataset'
num.items(x) <- value
```

**Arguments**

x                    a [BNDataset](#) object.  
value                number of items of the desired dataset.

---

num.nodes	<i>get number of nodes of an object.</i>
-----------	--

---

**Description**

Return the name of an object, of class [BN](#) or [InferenceEngine](#).

**Usage**

```
num.nodes(x)

## S4 method for signature 'BN'
num.nodes(x)

## S4 method for signature 'InferenceEngine'
num.nodes(x)
```

**Arguments**

x                    an object.

**Value**

number of nodes of the desired object.

---

num.nodes<-	<i>set number of nodes of an object.</i>
-------------	--

---

**Description**

Set the number of nodes of an object of type [BN](#) (number of nodes of the network) or [InferenceEngine](#) (where parameter contains the number of nodes of the junction tree).

**Usage**

```
num.nodes(x) <- value

## S4 replacement method for signature 'BN'
num.nodes(x) <- value

## S4 replacement method for signature 'InferenceEngine'
num.nodes(x) <- value
```

**Arguments**

x                    an object.  
value                the number of nodes in the object.



---

num.time.steps            *get number of time steps observed in a [BN](#) or a [BNDataset](#).*

---

### Description

Return the number of time steps observed in a dataset.

### Usage

```
num.time.steps(x)

## S4 method for signature 'BN'
num.time.steps(x)

## S4 method for signature 'BNDataset'
num.time.steps(x)
```

### Arguments

x                    a [BN](#) or a [BNDataset](#) object.

### Value

the number of time steps.

---

num.time.steps<-            *set number of time steps of a [BN](#) or a [BNDataset](#).*

---

### Description

Set the number of time steps of a dataset.

### Usage

```
num.time.steps(x) <- value

## S4 replacement method for signature 'BN'
num.time.steps(x) <- value

## S4 replacement method for signature 'BNDataset'
num.time.steps(x) <- value
```

### Arguments

x                    a [BN](#) or a [BNDataset](#) object.  
value                the number of time steps.

---

num.variables	<i>get number of variables of a <a href="#">BNDataset</a>.</i>
---------------	--

---

**Description**

Return the number of the variables contained in a dataset. This value corresponds to the value of [num.nodes](#) of a network built upon the same dataset.

**Usage**

```
num.variables(x)

## S4 method for signature 'BNDataset'
num.variables(x)

## S4 method for signature 'BNDataset'
num.variables(x)
```

**Arguments**

x                    a [BNDataset](#) object.

**Value**

number of variables of the desired dataset.

**See Also**

[num.nodes](#)

---

num.variables<-	<i>set number of variables of a <a href="#">BNDataset</a>.</i>
-----------------	--

---

**Description**

Set the number of variables observed in a dataset.

**Usage**

```
num.variables(x) <- value

## S4 replacement method for signature 'BNDataset'
num.variables(x) <- value
```

**Arguments**

x                    a [BNDataset](#) object.  
value                number of variables of the dataset.

---

observations            *get the list of observations of an [InferenceEngine](#).*

---

### Description

Return the list of observations added to an [InferenceEngine](#).

### Usage

```
observations(x)

## S4 method for signature 'InferenceEngine'
observations(x)
```

### Arguments

x                    an [InferenceEngine](#).

### Details

Output is a list in the following format:

- `observed.vars` vector of observed variables;
- `observed.vals` vector of values observed for the variables in `observed.vars` in the corresponding position.

### Value

the list of observations of the [InferenceEngine](#).

---

observations<-            *set the list of observations of an [InferenceEngine](#).*

---

### Description

Add a list of observations to an [InferenceEngine](#), using a list of observations composed by the two following vectors:

- `observed.vars` vector of observed variables;
- `observed.vals` vector of values observed for the variables in `observed.vars` in the corresponding position.

**Usage**

```
observations(x) <- value

## S4 replacement method for signature 'InferenceEngine'
observations(x) <- value
```

**Arguments**

x                    an [InferenceEngine](#).

value                the list of observations of the [InferenceEngine](#).

**Details**

Replace previous list of observations, if present. In order to add evidence, and not just replace it, one must use the [add.observations<-](#) method.

In case of multiple observations of the same variable, the last observation is the one used, as the most recent.

**See Also**

[add.observations<-](#)

---

plot                    *plot a [BN](#) as a picture.*

---

**Description**

plot a [BN](#) as a picture.

**Usage**

```
## S3 method for class 'BN'
plot(
  x,
  method = "default",
  use.node.names = TRUE,
  frac = 0.2,
  max.weight = max(dag(x)),
  node.size.lab = 14,
  node.col = rep("white", num.nodes(x)),
  plot.wpdag = FALSE,
  ...
)
```

**Arguments**

x	a <a href="#">BN</a> object.
method	either default of <code>qgraph</code> . The default method requires the <code>Rgraphviz</code> package, while <code>qgraph</code> requires the <code>qgraph</code> package and allows for a greater customization.
use.node.names	TRUE if node names have to be printed. If FALSE, numbers are used instead.
frac	minimum fraction [0,1] of presence of an edge to be plotted (used in case of <code>plot.wpdag=TRUE</code> ).
max.weight	maximum possible weight of an edge (used in case of <code>plot.wpdag=TRUE</code> ).
node.size.lab	font size for the node labels in the default mode.
node.col	list of (R) colors for the nodes.
plot.wpdag	if TRUE plot the network according to the WPDAG computed using bootstrap instead of the DAG.
...	potential further arguments when using <code>method="qgraph"</code> . Please refer to the <code>qgraph</code> documentation for the parameters available for the <code>qgraph()</code> method.

---

print                                    *print a [BN](#), [BNDataset](#) or [InferenceEngine](#) to stdout.*

---

**Description**

print a [BN](#), [BNDataset](#) or [InferenceEngine](#) to stdout.

**Usage**

```
## S3 method for class 'BN'
print(x, ...)

## S3 method for class 'BNDataset'
print(x, show.raw.data = FALSE, show.imputed.data = FALSE, ...)

## S3 method for class 'InferenceEngine'
print(x, engine = "jt", ...)
```

**Arguments**

x	a <a href="#">BN</a> , <a href="#">BNDataset</a> or <a href="#">InferenceEngine</a> .
...	potential other arguments.
show.raw.data	if x is a <a href="#">BNDataset</a> , print also raw dataset, if available.
show.imputed.data	if x is a <a href="#">BNDataset</a> , print also imputed dataset, if available.
engine	if x is an <a href="#">InferenceEngine</a> , specify the inference engine to be shown. Currently only <code>engine = 'jt'</code> is supported.

---

quantiles                    *get the list of quantiles of an object.*

---

### Description

Return the list of quantiles of a [BN](#) or a [BNDataset](#). It is set when a discretization needs to be performed.

### Usage

```
quantiles(x)

## S4 method for signature 'BN'
quantiles(x)

## S4 method for signature 'BNDataset'
quantiles(x)
```

### Arguments

x                    a list of vectors.

### Details

Output is a list of [num.nodes](#) vectors, one per variable. Each vector is NULL if the corresponding variable is discrete in the original dataset, and contains the cut points for the quantiles if the corresponding variable is continuous.

### Value

the list of quantiles of the [BN](#) or [BNDataset](#).

---

quantiles<-                    *set the list of quantiles of an object.*

---

### Description

Set the list of quantiles of a [BN](#) or a [BNDataset](#).

### Usage

```
quantiles(x) <- value

## S4 replacement method for signature 'BN'
quantiles(x) <- value

## S4 replacement method for signature 'BNDataset'
quantiles(x) <- value
```

**Arguments**

x                    a [BN](#) or [BNDataset](#).  
value                a list of vectors.

**Details**

It is used when a discretization needs to be performed.

**See Also**

[quantiles](#)

---

raw.data	<i>get raw data of a BNDataset.</i>
----------	-------------------------------------

---

**Description**

Return raw data contained in a [BNDataset](#) object, if any.

**Usage**

```
raw.data(x)  
  
## S4 method for signature 'BNDataset'  
raw.data(x)
```

**Arguments**

x                    a [BNDataset](#).

**See Also**

[has.raw.data](#), [has.imputed.data](#)

---

```
raw.data<-          add raw data.
```

---

**Description**

Insert raw data in a [BNDataset](#) object.

**Usage**

```
raw.data(x) <- value
```

```
## S4 replacement method for signature 'BNDataset'  
raw.data(x) <- value
```

**Arguments**

x                    a [BNDataset](#).  
value                a matrix of integers containing a dataset.

**See Also**

[has.raw.data](#), [raw.data](#), [read.dataset](#)

---

```
read.bif          Read a network from a .bif file.
```

---

**Description**

Read a network described in a .bif-formatted file, and build a [BN](#) object.

**Usage**

```
read.bif(x)
```

```
## S4 method for signature 'character'  
read.bif(x)
```

**Arguments**

x                    the .bif file, with absolute/relative position.

**Details**

The method relies on a coherent ordering of variable values and parameters in the file.

**Value**

a [BN](#) object.



---

read.dataset	<i>Read a dataset from file.</i>
--------------	----------------------------------

---

### Description

There are two ways to build a `BNDataset`: using two files containing respectively header informations and data, and manually providing the data table and the related header informations (variable names, cardinality and discreteness).

### Usage

```
read.dataset(
  object,
  data.file,
  header.file,
  data.with.header = FALSE,
  na.string.symbol = "?",
  sep.symbol = "",
  starts.from = 1,
  num.time.steps = 1
)

## S4 method for signature 'BNDataset,character,character'
read.dataset(
  object,
  data.file,
  header.file,
  data.with.header = FALSE,
  na.string.symbol = "?",
  sep.symbol = "",
  starts.from = 1,
  num.time.steps = 1
)
```

### Arguments

<code>object</code>	the <code>BNDataset</code> object.
<code>data.file</code>	the data file.
<code>header.file</code>	the header file.
<code>data.with.header</code>	TRUE if the first row of dataset file is an header (e.g. it contains the variable names).
<code>na.string.symbol</code>	character that denotes NA in the dataset.
<code>sep.symbol</code>	separator among values in the dataset.
<code>starts.from</code>	starting value for entries in the dataset (observed values, default is 1).

`num.time.steps` number of instants composing the observations (1, unless it is a dynamic system).

### Details

The key informations needed are: 1. the data; 2. the state of variables (discrete or continuous); 3. the names of the variables; 4. the cardinalities of the variables (if discrete), or the number of levels they have to be quantized into (if continuous). Names and cardinalities/leves can be guessed by looking at the data, but it is strongly advised to provide `_all_` of the informations, in order to avoid problems later on during the execution.

Data can be provided in form of `data.frame` or matrix. It can contain NAs. By default, NAs are indicated with `'?'`; to specify a different character for NAs, it is possible to provide also the `na.string.symbol` parameter. The values contained in the data have to be numeric (real for continuous variables, integer for discrete ones). The default range of values for a discrete variable  $X$  is  $[1, |X|]$ , with  $|X|$  being the cardinality of  $X$ . The same applies for the levels of quantization for continuous variables. If the value ranges for the data are different from the expected ones, it is possible to specify a different starting value (for the whole dataset) with the `starts.from` parameter. E.g. by `starts.from=0` we assume that the values of the variables in the dataset have range  $[0, |X|-1]$ . Please keep in mind that the internal representation of `bnstruct` starts from 1, and the original starting values are then lost.

It is possible to use two files, one for the data and one for the metadata, instead of providing manually all of the info. `bnstruct` requires the data files to be in a format subsequently described. The actual data has to be in (a text file containing data in) tabular format, one tuple per row, with the values for each variable separated by a space or a tab. Values for each variable have to be numbers, starting from 1 in case of discrete variables. Data files can have a first row containing the names of the corresponding variables.

In addition to the data file, a header file containing additional informations can also be provided. An header file has to be composed by three rows of tab-delimited values: 1. list of names of the variables, in the same order of the data file; 2. a list of integers representing the cardinality of the variables, in case of discrete variables, or the number of levels each variable has to be quantized in, in case of continuous variables; 3. a list that indicates, for each variable, if the variable is continuous (c or C), and thus has to be quantized before learning, or discrete (d or D).

### See Also

`BNDataset`

### Examples

```
## Not run:
dataset <- BNDataset()
dataset <- read.dataset(dataset, "file.data", "file.header")

## End(Not run)
```

---

read.dsc	<i>Read a network from a .dsc file.</i>
----------	---

---

**Description**

Read a network described in a .dsc-formatted file, and build a [BN](#) object.

**Usage**

```
read.dsc(x)
```

```
## S4 method for signature 'character'  
read.dsc(x)
```

**Arguments**

x                    the .dsc file, with absolute/relative position.

**Details**

The method relies on a coherent ordering of variable values and parameters in the file.

**Value**

a [BN](#) object.

---

read.net	<i>Read a network from a .net file.</i>
----------	---

---

**Description**

Read a network described in a .net-formatted file, and build a [BN](#) object.

**Usage**

```
read.net(x)
```

```
## S4 method for signature 'character'  
read.net(x)
```

**Arguments**

x                    the .net file, with absolute/relative position.

**Details**

The method relies on a coherent ordering of variable values and parameters in the file.

**Value**

a [BN](#) object.

---

sample.dataset	<i>sample a <a href="#">BNDataset</a> from a network of an inference engine.</i>
----------------	--

---

**Description**

sample a [BNDataset](#) from a network of an inference engine.

**Usage**

```
sample.dataset(x, n = 100, mar = 0)

## S4 method for signature 'BN'
sample.dataset(x, n = 100, mar = 0)

## S4 method for signature 'InferenceEngine'
sample.dataset(x, n = 100)
```

**Arguments**

x	a <a href="#">BN</a> or <a href="#">InferenceEngine</a> object.
n	number of items to sample.
mar	fraction [0,1] of missing values in the sampled dataset (missing at random), default value is 0 (no missing values).

**Value**

a [BNDataset](#)

---

sample.row	<i>sample a row vector of values for a network.</i>
------------	---

---

**Description**

sample a row vector of values for a network.

**Usage**

```
sample.row(x, mar = 0)

## S4 method for signature 'BN'
sample.row(x, mar = 0)
```

**Arguments**

x	a <a href="#">BN</a> or <a href="#">InferenceEngine</a> object.
mar	fraction [0,1] of missing values in the sampled vector (missing at random), default value is 0 (no missing values).

**Value**

a vector of values.

---

save.to.eps	<i>save a <a href="#">BN</a> picture as .eps file.</i>
-------------	--

---

**Description**

Save an image of a Bayesian Network as an .eps file.

**Usage**

```
save.to.eps(x, filename, ...)

## S4 method for signature 'BN,character'
save.to.eps(x, filename, ...)
```

**Arguments**

x	a <a href="#">BN</a> object
filename	name (with path, if needed) of the file to be created
...	parameters for the <a href="#">plot</a> method.

**See Also**

[plot](#)

**Examples**

```
## Not run:
save.to.eps(x, "out.eps")

## End(Not run)
```

---

scoring.func	<i>Read the scoring function used to learn the structure of a network.</i>
--------------	--

---

**Description**

Read the scoring function used in the [learn.structure](#) method. Outcome is meaningful only if the structure of a network has been learnt.

**Usage**

```
scoring.func(x)

## S4 method for signature 'BN'
scoring.func(x)
```

**Arguments**

x                    the [BN](#) object.

**Value**

the scoring function used.

---

scoring.func<-	<i>Set the scoring function used to learn the structure of a network.</i>
----------------	---

---

**Description**

Set the scoring function used in the [learn.structure](#) method.

**Usage**

```
scoring.func(x) <- value

## S4 replacement method for signature 'BN'
scoring.func(x) <- value
```

**Arguments**

x                    the [BN](#) object.  
value                the scoring function used.

**Value**

updated [BN](#).

---

shd	<i>compute the Structural Hamming Distance between two adjacency matrices.</i>
-----	--

---

**Description**

Compute the Structural Hamming Distance between two adjacency matrices, that is, the distance, in terms of edges, between two network structures. The lower the shd, the more similar are the two network structures.

**Usage**

```
shd(g1, g2)
```

**Arguments**

g1	first adjacency matrix.
g2	second adjacency matrix.

---

show	<i>Show method for objects.</i>
------	---------------------------------

---

**Description**

The show method allows to provide a custom aspect for the output that is generated when the name of an instance is given as command in an R session.

**Usage**

```
show(object)
```

**Arguments**

object	an object.
--------	------------

---

struct.algo	<i>Read the algorithm used to learn the structure of a network.</i>
-------------	---

---

**Description**

Read the algorithm used in the [learn.structure](#) method. Outcome is meaningful only if the structure of a network has been learnt.

**Usage**

```
struct.algo(x)

## S4 method for signature 'BN'
struct.algo(x)
```

**Arguments**

x                    the [BN](#) object.

**Value**

the structure learning algorithm used.

---

struct.algo<-	<i>Set the algorithm used to learn the structure of a network.</i>
---------------	--

---

**Description**

Set the algorithm used in the [learn.structure](#) method.

**Usage**

```
struct.algo(x) <- value

## S4 replacement method for signature 'BN'
struct.algo(x) <- value
```

**Arguments**

x                    the [BN](#) object.  
value                the scoring function used.

**Value**

updated [BN](#).



---

test.updated.bn	<i>check if an updated BN is present in an <a href="#">InferenceEngine</a>.</i>
-----------------	---

---

## Description

Check if an `InferenceEngine` actually contains an updated network, in order to provide the chance of a fallback and use the original network if no belief propagation has been performed. An `InferenceEngine` built specifying a set of interventions will contain an updated BN with altered structure and no conditional probability tables (unless they are computed by a belief propagation operation.)

## Usage

```
test.updated.bn(x)

## S4 method for signature 'InferenceEngine'
test.updated.bn(x)
```

## Arguments

x                    an `InferenceEngine`.

## Value

TRUE if an updated network is contained in the `InferenceEngine`, FALSE otherwise.

## Examples

```
## Not run:
dataset <- BNDataset("file.header", "file.data")
bn <- BN(dataset)
ie <- InferenceEngine(bn)
test.updated.bn(ie) # FALSE

observations(ie) <- list("observed.vars"=("A","G","X"), "observed.vals"=c(1,2,1))
ie <- belief.propagation(ie)
test.updated.bn(ie) # TRUE

interventions <- list("intervention.vars"=("A","G","X"), "intervention.vals"=c(1,2,1))
ie2 <- InferenceEngine(bn, interventions = interventions)
test.updated.bn(ie2) # TRUE

## End(Not run)
```

---

tune.knn.impute	<i>tune the parameter k of the knn algorithm used in imputation.</i>
-----------------	--

---

### Description

tune the parameter k of the knn algorithm used in imputation.

### Usage

```
tune.knn.impute(
  data,
  cat.var = 1:ncol(data),
  k.min = 1,
  k.max = 20,
  frac.miss = 0.1,
  n.iter = 20,
  seed = 0
)
```

### Arguments

data	a numerical matrix.
cat.var	vector containing the categorical variables
k.min	minimum value for k
k.max	maximum value for k
frac.miss	fraction of missing values to add
n.iter	number of iterations for each k
seed	random seed

### Value

matrix of error distributions

---

updated.bn	<i>get the updated BN object contained in an <a href="#">InferenceEngine</a>.</i>
------------	---

---

### Description

Return an updated network contained in an InferenceEngine.

### Usage

```
updated.bn(x)

## S4 method for signature 'InferenceEngine'
updated.bn(x)
```

**Arguments**

x                    an [InferenceEngine](#).

**Value**

the updated [BN](#) object contained in an [InferenceEngine](#).

---

updated.bn<-                    *set the updated [BN](#) object contained in an [InferenceEngine](#).*

---

**Description**

Add an updated network to an [InferenceEngine](#).

**Usage**

```
updated.bn(x) <- value
```

```
## S4 replacement method for signature 'InferenceEngine'
```

```
updated.bn(x) <- value
```

**Arguments**

x                    an [InferenceEngine](#).

value                the updated [BN](#) object contained in an [InferenceEngine](#).

---

variables                    *get variables of an object.*

---

**Description**

Get the list of variables (with their names) of a [BN](#) or [BNDataset](#).

**Usage**

```
variables(x)
```

```
## S4 method for signature 'BN'
```

```
variables(x)
```

```
## S4 method for signature 'BNDataset'
```

```
variables(x)
```

**Arguments**

x                    an object.

**Value**

vector of the variables names of the desired object.

---

`variables<-`                    *set variables of an object.*

---

**Description**

Set the list of variable names in a [BN](#) or [BNDataset](#) object.

**Usage**

```
variables(x) <- value
```

```
## S4 replacement method for signature 'BN'
variables(x) <- value
```

```
## S4 replacement method for signature 'BNDataset'
variables(x) <- value
```

**Arguments**

`x`                    an object.

`value`                vector containing the variable names of the object. Overwrites `num.nodes` slot if non-matching.

---

`wpdag`                    *get the WPDAG of an object.*

---

**Description**

Return the weighted partially directed acyclic graph of a network, when available (e.g. when bootstrap on dataset is performed).

**Usage**

```
wpdag(x)
```

```
## S4 method for signature 'BN'
wpdag(x)
```

**Arguments**

`x`                    an object.

**Value**

matrix containing the WPDAG of the object.

---

wpdag.from.dag	<i>Initialize a WPDAG from a DAG.</i>
----------------	---------------------------------------

---

### Description

Given a [BN](#) object with a dag, return a network with its wpdag set as the CPDAG computed starting from the dag.

### Usage

```
wpdag.from.dag(x, layering = NULL, layer.struct = NULL)
```

```
## S4 method for signature 'BN'
```

```
wpdag.from.dag(x, layering = NULL, layer.struct = NULL)
```

### Arguments

x                    a [BN](#) object.

layering            vector containing the layers each node belongs to.

layer.struct        0/1 matrix for indicating which layers can contain parent nodes for nodes in a layer (only for mmhc, mmpc).

### Value

a [BN](#) object with an initialized wpdag.

### See Also

[dag.to.cpdag](#)

### Examples

```
## Not run:  
net <- learn.network(dataset, layering=layering, layer.struct=layer.struct)  
wp.net <- wpdag.from.dag(net, layering, layer.struct=layer.struct)
```

```
## End(Not run)
```

---

```
wpdag<-          set WPDAG of the object.
```

---

### Description

Set the weighted partially directed acyclic graph of a network (e.g. in case bootstrap on dataset is performed).

### Usage

```
wpdag(x) <- value

## S4 replacement method for signature 'BN'
wpdag(x) <- value
```

### Arguments

x	an object.
value	matrix containing the WPDAG of the object.

---

```
write.dsc          Write a network saving it in a .dsc file.
```

---

### Description

Write a network on disk, saving it in a .dsc-formatted file.

### Usage

```
write.dsc(x, path = "./")

## S4 method for signature 'BN'
write.dsc(x, path = "./")
```

### Arguments

x	the <a href="#">BN</a> object.
path	the relative or absolute path of the directory of the created file.

---

`write_xgmml`*Write a network saving it in an XGMML file.*

---

### Description

Write a network on disk, saving it in an XGMML file, for importing it in Cytoscape.

### Usage

```
write_xgmml(  
  x,  
  path = "./network",  
  write.wpdag = FALSE,  
  node.col = rep("white", num.nodes(x)),  
  frac = 0.2,  
  max.weight = max(wpdag(x))  
)
```

```
## S4 method for signature 'BN'  
write_xgmml(  
  x,  
  path = "./network",  
  write.wpdag = FALSE,  
  node.col = rep("white", num.nodes(x)),  
  frac = 0.2,  
  max.weight = max(wpdag(x))  
)
```

### Arguments

<code>x</code>	the <a href="#">BN</a> object.
<code>path</code>	file name with relative or absolute path to be written.
<code>write.wpdag</code>	write the weighted PDAG computed using bootstrap samples or the MMPC structure algorithm, instead of the normaldag (default FALSE).
<code>node.col</code>	vector of colors for each node of the network (in R colnames).
<code>frac</code>	minimum fraction [0,1] of presence of an edge to be plotted (used in case of <code>write.wpdag=TRUE</code> ).
<code>max.weight</code>	maximum possible weight of an edge (used in case of <code>write.wpdag=TRUE</code> ).

# Index

add.observations<-, 4  
add.observations<- , InferenceEngine-method  
    (add.observations<-), 4  
asia, 5, 6  
asia\_10000, 5, 5, 6  
asia\_2\_layers, 6

belief.propagation, 7  
belief.propagation, InferenceEngine  
    (belief.propagation), 7  
belief.propagation, InferenceEngine-method  
    (belief.propagation), 7  
BN, 7, 8, 10, 18–20, 22, 25, 33, 40, 42–47, 49,  
    50, 52, 56, 57, 60–64, 67–70, 72–79  
BN (BN-class), 8  
bn, 8  
BN, BN-class (BN-class), 8  
bn, InferenceEngine (bn), 8  
bn, InferenceEngine-method (bn), 8  
BN-class, 8  
bn<- , 10  
bn<- , InferenceEngine-method (bn<-), 10  
BNdataset, 5, 8, 13–15, 17, 18, 21, 22, 26–32,  
    40, 42–47, 49, 52, 54, 55, 57, 58,  
    61–65, 68, 75, 76  
BNdataset (BNdataset-class), 10  
BNdataset, BNdataset-class  
    (BNdataset-class), 10  
BNdataset-class, 10  
boot, 13  
boot, BNdataset (boot), 13  
boot, BNdataset, numeric-method (boot), 13  
boots, 14, 26, 27, 30  
boots, BNdataset (boots), 14  
boots, BNdataset-method (boots), 14  
boots<- , 14  
boots<- , BNdataset-method (boots<-), 14  
bootstrap, 11, 13, 15, 18  
bootstrap, BNdataset (bootstrap), 15  
bootstrap, BNdataset-method (bootstrap),  
    15  
build.junction.tree, 15, 38  
build.junction.tree, InferenceEngine  
    (build.junction.tree), 15  
build.junction.tree, InferenceEngine-method  
    (build.junction.tree), 15

child, 16, 17  
child\_NA\_5000, 16, 17  
complete, 17  
complete, BNdataset (complete), 17  
complete, BNdataset-method (complete), 17  
cpts, 18  
cpts, BN (cpts), 18  
cpts, BN-method (cpts), 18  
cpts<- , 19  
cpts<- , BN-method (cpts<-), 19

dag, 19  
dag, BN (dag), 19  
dag, BN-method (dag), 19  
dag.to.cpdag, 20, 77  
dag<- , 21  
dag<- , BN-method (dag<-), 21  
data.file, 21, 21, 29  
data.file, BNdataset (data.file), 21  
data.file, BNdataset-method (data.file),  
    21  
data.file<- , 22  
data.file<- , BNdataset-method  
    (data.file<-), 22  
discreteness, 22  
discreteness, BN (discreteness), 22  
discreteness, BN-method (discreteness),  
    22  
discreteness, BNdataset (discreteness),  
    22  
discreteness, BNdataset-method  
    (discreteness), 22



- discreteness<-, 23
- discreteness<-, BN-method  
(discreteness<-), 23
- discreteness<-, BNDataset-method  
(discreteness<-), 23
- edge.dir.wpdag, 23
- em, 24
- em, InferenceEngine, BNDataset (em), 24
- em, InferenceEngine, BNDataset-method  
(em), 24
- get.most.probable.values, 25
- get.most.probable.values, BN  
(get.most.probable.values), 25
- get.most.probable.values, BN-method  
(get.most.probable.values), 25
- get.most.probable.values, InferenceEngine  
(get.most.probable.values), 25
- get.most.probable.values, InferenceEngine-method  
(get.most.probable.values), 25
- has.boots, 14, 26, 27, 30
- has.boots, BNDataset (has.boots), 26
- has.boots, BNDataset-method (has.boots),  
26
- has.imputed.boots, 14, 26, 26, 30
- has.imputed.boots, BNDataset  
(has.imputed.boots), 26
- has.imputed.boots, BNDataset-method  
(has.imputed.boots), 26
- has.imputed.data, 27, 28, 32, 63
- has.imputed.data, BNDataset  
(has.imputed.data), 27
- has.imputed.data, BNDataset-method  
(has.imputed.data), 27
- has.raw.data, 27, 28, 32, 63, 64
- has.raw.data, BNDataset (has.raw.data),  
28
- has.raw.data, BNDataset-method  
(has.raw.data), 28
- header.file, 28
- header.file, BNDataset (header.file), 28
- header.file, BNDataset-method  
(header.file), 28
- header.file<-, 29
- header.file<-, BNDataset-method  
(header.file<-), 29
- imp.boots, 14, 26, 27, 30
- imp.boots, BNDataset (imp.boots), 30
- imp.boots, BNDataset-method (imp.boots),  
30
- imp.boots<-, 30
- imp.boots<-, BNDataset-method  
(imp.boots<-), 30
- impute, 11, 16, 18, 31
- impute, BNDataset (impute), 31
- impute, BNDataset-method (impute), 31
- imputed.data, 27, 28, 31, 32
- imputed.data, BNDataset (imputed.data),  
31
- imputed.data, BNDataset-method  
(imputed.data), 31
- imputed.data<-, 32
- imputed.data<-, BNDataset-method  
(imputed.data<-), 32
- InferenceEngine, 4, 7, 8, 10, 16, 24, 25, 32,  
34–39, 51, 56, 59–61, 68, 69, 73–75
- InferenceEngine  
(InferenceEngine-class), 32
- InferenceEngine, InferenceEngine-class  
(InferenceEngine-class), 32
- InferenceEngine-class, 32
- initialize, BN-method (BN-class), 8
- initialize, BNDataset-method  
(BNDataset-class), 10
- initialize, InferenceEngine-method  
(InferenceEngine-class), 32
- interventions, 34
- interventions, InferenceEngine  
(interventions), 34
- interventions, InferenceEngine-method  
(interventions), 34
- interventions<-, 35
- interventions<-, InferenceEngine-method  
(interventions<-), 35
- jpts, 35
- jpts, InferenceEngine (jpts), 35
- jpts, InferenceEngine-method (jpts), 35
- jpts<-, 36
- jpts<-, InferenceEngine-method (jpts<-),  
36
- jt.cliques, 37
- jt.cliques, InferenceEngine  
(jt.cliques), 37
- jt.cliques, InferenceEngine-method  
(jt.cliques), 37

- jt.cliques<-, 37
- jt.cliques<-, InferenceEngine-method  
(jt.cliques<-), 37
- junction.tree, 38
- junction.tree, InferenceEngine  
(junction.tree), 38
- junction.tree, InferenceEngine-method  
(junction.tree), 38
- junction.tree<-, 38
- junction.tree<-, InferenceEngine-method  
(junction.tree<-), 38
  
- knn.impute, 39
  
- layering, 40
- layering, BN (layering), 40
- layering, BN-method (layering), 40
- learn.dynamic.network, 40, 46
- learn.dynamic.network, BN  
(learn.dynamic.network), 40
- learn.dynamic.network, BN-method  
(learn.dynamic.network), 40
- learn.dynamic.network, BNDataset  
(learn.dynamic.network), 40
- learn.dynamic.network, BNDataset-method  
(learn.dynamic.network), 40
- learn.network, 40, 42, 43
- learn.network, BN (learn.network), 43
- learn.network, BN-method  
(learn.network), 43
- learn.network, BNDataset  
(learn.network), 43
- learn.network, BNDataset-method  
(learn.network), 43
- learn.params, 42, 46, 46
- learn.params, BN, BNDataset  
(learn.params), 46
- learn.params, BN, BNDataset-method  
(learn.params), 46
- learn.structure, 42, 46, 47, 70, 72
- learn.structure, BN, BNDataset  
(learn.structure), 47
- learn.structure, BN, BNDataset-method  
(learn.structure), 47
  
- marginals, 51
- marginals, InferenceEngine (marginals),  
51
  
- marginals, InferenceEngine-method  
(marginals), 51
  
- name, 52
- name, BN (name), 52
- name, BN-method (name), 52
- name, BNDataset (name), 52
- name, BNDataset-method (name), 52
- name<-, 52
- name<-, BN-method (name<-), 52
- name<-, BNDataset-method (name<-), 52
- node.sizes, 53
- node.sizes, BN (node.sizes), 53
- node.sizes, BN-method (node.sizes), 53
- node.sizes, BNDataset (node.sizes), 53
- node.sizes, BNDataset-method  
(node.sizes), 53
- node.sizes<-, 53
- node.sizes<-, BN-method (node.sizes<-),  
53
- node.sizes<-, BNDataset-method  
(node.sizes<-), 53
- num.boots, 54
- num.boots, BNDataset (num.boots), 54
- num.boots, BNDataset-method (num.boots),  
54
- num.boots<-, 54
- num.boots<-, BNDataset-method  
(num.boots<-), 54
- num.items, 55
- num.items, BNDataset (num.items), 55
- num.items, BNDataset-method (num.items),  
55
- num.items<-, 55
- num.items<-, BNDataset-method  
(num.items<-), 55
- num.nodes, 25, 56, 58, 62
- num.nodes, BN (num.nodes), 56
- num.nodes, BN-method (num.nodes), 56
- num.nodes, InferenceEngine (num.nodes),  
56
- num.nodes, InferenceEngine-method  
(num.nodes), 56
- num.nodes<-, 56
- num.nodes<-, BN-method (num.nodes<-), 56
- num.nodes<-, InferenceEngine-method  
(num.nodes<-), 56
- num.time.steps, 57
- num.time.steps, BN (num.time.steps), 57

- num.time.steps, BN-method  
(num.time.steps), [57](#)
- num.time.steps, BNDataset  
(num.time.steps), [57](#)
- num.time.steps, BNDataset-method  
(num.time.steps), [57](#)
- num.time.steps<-, [57](#)
- num.time.steps<-, BN-method  
(num.time.steps<-), [57](#)
- num.time.steps<-, BNDataset-method  
(num.time.steps<-), [57](#)
- num.variables, [58](#)
- num.variables, BNDataset  
(num.variables), [58](#)
- num.variables, BNDataset-method  
(num.variables), [58](#)
- num.variables<-, [58](#)
- num.variables<-, BNDataset-method  
(num.variables<-), [58](#)
  
- observations, [59](#)
- observations, InferenceEngine  
(observations), [59](#)
- observations, InferenceEngine-method  
(observations), [59](#)
- observations<-, [59](#)
- observations<-, InferenceEngine-method  
(observations<-), [59](#)
  
- plot, [60](#), [69](#)
- plot, BN (plot), [60](#)
- plot.BN (plot), [60](#)
- plot.BN, BN (plot), [60](#)
- print, [61](#)
- print, BN (print), [61](#)
- print, BNDataset (print), [61](#)
- print, InferenceEngine (print), [61](#)
- print.BN (print), [61](#)
- print.BN, BN (print), [61](#)
- print.BNDataset (print), [61](#)
- print.BNDataset, BNDataset (print), [61](#)
- print.InferenceEngine (print), [61](#)
- print.InferenceEngine, InferenceEngine  
(print), [61](#)
  
- quantiles, [62](#), [63](#)
- quantiles, BN (quantiles), [62](#)
- quantiles, BN-method (quantiles), [62](#)
- quantiles, BNDataset (quantiles), [62](#)
- quantiles, BNDataset-method (quantiles),  
[62](#)
- quantiles<-, [62](#)
- quantiles<-, BN-method (quantiles<-), [62](#)
- quantiles<-, BNDataset-method  
(quantiles<-), [62](#)
  
- raw.data, [27](#), [28](#), [32](#), [63](#), [64](#)
- raw.data, BNDataset (raw.data), [63](#)
- raw.data, BNDataset-method (raw.data), [63](#)
- raw.data<-, [64](#)
- raw.data<-, BNDataset-method  
(raw.data<-), [64](#)
- read.bif, [64](#)
- read.bif, character (read.bif), [64](#)
- read.bif, character-method (read.bif), [64](#)
- read.dataset, [11](#), [32](#), [64](#), [65](#)
- read.dataset, BNDataset, character, character  
(read.dataset), [65](#)
- read.dataset, BNDataset, character, character-method  
(read.dataset), [65](#)
- read.dsc, [67](#)
- read.dsc, character (read.dsc), [67](#)
- read.dsc, character-method (read.dsc), [67](#)
- read.net, [67](#)
- read.net, character (read.net), [67](#)
- read.net, character-method (read.net), [67](#)
  
- sample.dataset, [68](#)
- sample.dataset, BN (sample.dataset), [68](#)
- sample.dataset, BN-method  
(sample.dataset), [68](#)
- sample.dataset, InferenceEngine  
(sample.dataset), [68](#)
- sample.dataset, InferenceEngine-method  
(sample.dataset), [68](#)
- sample.row, [68](#)
- sample.row, BN (sample.row), [68](#)
- sample.row, BN-method (sample.row), [68](#)
- save.to.eps, [69](#)
- save.to.eps, BN, character (save.to.eps),  
[69](#)
- save.to.eps, BN, character-method  
(save.to.eps), [69](#)
- scoring.func, [70](#)
- scoring.func, BN (scoring.func), [70](#)
- scoring.func, BN-method (scoring.func),  
[70](#)
- scoring.func<-, [70](#)

scoring.func<- ,BN-method  
    (scoring.func<-), 70

shd, 71

show, 71

show,AllTheClasses-method (show), 71

show,BN-method (show), 71

show,BNDataset-method (show), 71

show,InferenceEngine-method (show), 71

struct.algo, 72

struct.algo,BN (struct.algo), 72

struct.algo,BN-method (struct.algo), 72

struct.algo<- , 72

struct.algo<- ,BN-method  
    (struct.algo<-), 72

test.updated.bn, 73

test.updated.bn,InferenceEngine  
    (test.updated.bn), 73

test.updated.bn,InferenceEngine-method  
    (test.updated.bn), 73

tune.knn.impute, 74

updated.bn, 74

updated.bn,InferenceEngine  
    (updated.bn), 74

updated.bn,InferenceEngine-method  
    (updated.bn), 74

updated.bn<- , 75

updated.bn<- ,InferenceEngine-method  
    (updated.bn<-), 75

variables, 75

variables,BN (variables), 75

variables,BN-method (variables), 75

variables,BNDataset (variables), 75

variables,BNDataset-method (variables),  
    75

variables<- , 76

variables<- ,BN-method (variables<-), 76

variables<- ,BNDataset-method  
    (variables<-), 76

wpdag, 76

wpdag,BN (wpdag), 76

wpdag,BN-method (wpdag), 76

wpdag.from.dag, 20, 77

wpdag.from.dag,BN (wpdag.from.dag), 77

wpdag.from.dag,BN-method  
    (wpdag.from.dag), 77

wpdag<- , 78

wpdag<- ,BN-method (wpdag<-), 78

write.dsc, 78

write.dsc,BN (write.dsc), 78

write.dsc,BN-method (write.dsc), 78

write\_xgmml, 79

write\_xgmml,BN (write\_xgmml), 79

write\_xgmml,BN-method (write\_xgmml), 79