

# Package ‘dbnR’

July 22, 2025

**Type** Package

**Title** Dynamic Bayesian Network Learning and Inference

**Version** 0.7.9

**Description** Learning and inference over dynamic Bayesian networks of arbitrary Markovian order. Extends some of the functionality offered by the 'bnlearn' package to learn the networks from data and perform exact inference. It offers three structure learning algorithms for dynamic Bayesian networks: Trabelsi G. (2013) <[doi:10.1007/978-3-642-41398-8\\_34](https://doi.org/10.1007/978-3-642-41398-8_34)>, Santos F.P. and Maciel C.D. (2014) <[doi:10.1109/BRC.2014.6880957](https://doi.org/10.1109/BRC.2014.6880957)>, Quesada D., Bielza C. and Larrañaga P. (2021) <[doi:10.1007/978-3-030-86271-8\\_14](https://doi.org/10.1007/978-3-030-86271-8_14)>. It also offers the possibility to perform forecasts of arbitrary length. A tool for visualizing the structure of the net is also provided via the 'visNetwork' package.

**Depends** R (>= 3.5.0), bnlearn (>= 4.5)

**Imports** data.table (>= 1.12.4), Rcpp (>= 1.0.2), magrittr (>= 1.5), R6 (>= 2.4.1), stats (>= 3.6.0), MASS (>= 7.3-55)

**Suggests** visNetwork (>= 2.0.8), grDevices (>= 3.6.0), utils (>= 3.6.0), graphics (>= 3.6.0), testthat (>= 2.1.0)

**LinkingTo** Rcpp

**URL** <https://github.com/dkesada/dbnR>

**License** GPL-3

**Encoding** UTF-8

**LazyData** true

**RoxygenNote** 7.3.1

**NeedsCompilation** yes

**Author** David Quesada [aut, cre],  
Gabriel Valverde [ctb]

**Maintainer** David Quesada <dquesada@gmail.com>

**Repository** CRAN

**Date/Publication** 2024-06-19 15:10:03 UTC

## Contents

AIC.dbn . . . . .	3
AIC.dbn.fit . . . . .	3
all.equal.dbn . . . . .	4
all.equal.dbn.fit . . . . .	4
as.character.dbn . . . . .	5
BIC.dbn . . . . .	5
BIC.dbn.fit . . . . .	6
calc_mu . . . . .	6
calc_sigma . . . . .	7
coef.dbn.fit . . . . .	8
degree . . . . .	8
filtered_fold_dt . . . . .	9
filter_same_cycle . . . . .	9
fitted.dbn.fit . . . . .	10
fit_dbn_params . . . . .	11
fold_dt . . . . .	11
forecast_ts . . . . .	12
generate_random_network_exp . . . . .	13
learn_dbn_struct . . . . .	14
logLik.dbn . . . . .	15
logLik.dbn.fit . . . . .	15
mean.dbn.fit . . . . .	16
motor . . . . .	16
mvn_inference . . . . .	17
nodes . . . . .	18
nodes<- . . . . .	18
plot.dbn . . . . .	19
plot.dbn.fit . . . . .	19
plot_dynamic_network . . . . .	20
plot_static_network . . . . .	21
predict.dbn.fit . . . . .	21
predict_bn . . . . .	22
predict_dt . . . . .	22
print.dbn . . . . .	23
print.dbn.fit . . . . .	24
rbn.dbn.fit . . . . .	24
reduce_freq . . . . .	25
residuals.dbn.fit . . . . .	25
score . . . . .	26
shift_values . . . . .	26
sigma.dbn.fit . . . . .	27
smooth_ts . . . . .	28
time_rename . . . . .	29
[[<-.dbn.fit . . . . .	30
\$<-.dbn.fit . . . . .	30

---

AIC.dbn	<i>Calculate the AIC of a dynamic Bayesian network</i>
---------	--

---

**Description**

Generic method for calculating the Akaike information criterion (AIC) of a "dbn" S3 object given some data. Calls bnlearn's [AIC](#) underneath.

**Usage**

```
## S3 method for class 'dbn'  
AIC(object, ..., k)
```

**Arguments**

object	the structure of the network
...	additional parameters for the network scoring
k	the penalty parameter

**Value**

the AIC score of the network

---

AIC.dbn.fit	<i>Calculate the AIC of a dynamic Bayesian network</i>
-------------	--

---

**Description**

Generic method for calculating the Akaike information criterion (AIC) of a "dbn.fit" S3 object given some data. Calls bnlearn's [AIC](#) underneath.

**Usage**

```
## S3 method for class 'dbn.fit'  
AIC(object, ..., k)
```

**Arguments**

object	the fitted network
...	additional parameters for the network scoring
k	the penalty parameter

**Value**

the AIC score of the network

---

all.equal.dbn      *Check if two network structures are equal to each other*

---

**Description**

Generic method for checking the equality of two "dbn" S3 objects. Calls bnlearn's [all.equal](#) underneath.

**Usage**

```
## S3 method for class 'equal.dbn'  
all(target, current, ...)
```

**Arguments**

target	"dbn" object
current	the other "dbn" object
...	additional parameters

**Value**

boolean result of the comparison

---

all.equal.dbn.fit      *Check if two fitted networks are equal to each other*

---

**Description**

Generic method for checking the equality of two "dbn.fit" S3 objects. Calls bnlearn's [all.equal](#) underneath.

**Usage**

```
## S3 method for class 'equal.dbn.fit'  
all(target, current, ...)
```

**Arguments**

target	"dbn.fit" object
current	the other "dbn.fit" object
...	additional parameters

**Value**

boolean result of the comparison

---

as.character.dbn	<i>Convert a network structure into a model string</i>
------------------	--

---

**Description**

Generic method for converting a "dbn" S3 object into a string. Calls bnlearn's [as.character](#) underneath.

**Usage**

```
## S3 method for class 'dbn'  
as.character(x, ...)
```

**Arguments**

x	a "dbn" object
...	additional parameters

**Value**

string representing the DBN model

---

BIC.dbn	<i>Calculate the BIC of a dynamic Bayesian network</i>
---------	--

---

**Description**

Generic method for calculating the Bayesian information criterion (BIC) of a "dbn" S3 object given some data. Calls bnlearn's [BIC](#) underneath.

**Usage**

```
## S3 method for class 'dbn'  
BIC(object, ...)
```

**Arguments**

object	the structure of the network
...	additional parameters for the network scoring

**Value**

the BIC score of the network

---

BIC.dbn.fit	<i>Calculate the BIC of a dynamic Bayesian network</i>
-------------	--

---

**Description**

Generic method for calculating the Bayesian information criterion (BIC) of a "dbn.fit" S3 object given some data. Calls bnlearn's [BIC](#) underneath.

**Usage**

```
## S3 method for class 'dbn.fit'  
BIC(object, ...)
```

**Arguments**

object	the fitted network
...	additional parameters for the network scoring

**Value**

the BIC score of the network

---

calc_mu	<i>Calculate the mu vector from a fitted BN or DBN</i>
---------	--

---

**Description**

Given a "bn.fit" or a "dbn.fit" object, calculate the mu vector of the equivalent multivariate Gaussian distribution. Front end of a C++ function.

**Usage**

```
calc_mu(fit)
```

**Arguments**

fit	a bn.fit or dbn.fit object
-----	----------------------------

**Value**

a named numeric vector of the means of each variable

## Examples

```
dt_train <- dbnR::motor[200:2500]
net <- bnlearn::mmhc(dt_train)
fit <- bnlearn::bn.fit(net, dt_train, method = "mle-g")
mu <- dbnR::calc_mu(fit)

f_dt_train <- dbnR::fold_dt(dt_train, size = 2)
net <- dbnR::learn_dbn_struct(dt_train, size = 2)
fit <- dbnR::fit_dbn_params(net, f_dt_train)
mu <- dbnR::calc_mu(fit)
```

---

calc\_sigma

*Calculate the sigma covariance matrix from a fitted BN or DBN*

---

## Description

Given a "bn.fit" or a "dbn.fit" object, calculate the sigma covariance matrix of the equivalent multivariate Gaussian distribution. Front end of a C++ function.

## Usage

```
calc_sigma(fit)
```

## Arguments

fit                    a bn.fit or dbn.fit object

## Value

a named numeric covariance matrix of the nodes

## Examples

```
dt_train <- dbnR::motor[200:2500]
net <- bnlearn::mmhc(dt_train)
fit <- bnlearn::bn.fit(net, dt_train, method = "mle-g")
sigma <- dbnR::calc_sigma(fit)

f_dt_train <- dbnR::fold_dt(dt_train, size = 2)
net <- dbnR::learn_dbn_struct(dt_train, size = 2)
fit <- dbnR::fit_dbn_params(net, f_dt_train)
sigma <- dbnR::calc_sigma(fit)
```

---

<code>coef.dbn.fit</code>	<i>Extracts the coefficients of a DBN</i>
---------------------------	---

---

### Description

Generic method for "dbn.fit" S3 objects. Calls `bnlearn` underneath.

### Usage

```
## S3 method for class 'dbn.fit'
coef(object, ...)
```

### Arguments

<code>object</code>	the fitted network
<code>...</code>	additional parameters

### Value

the coefficients of the network

---

<code>degree</code>	<i>Calculates the degree of a list of nodes</i>
---------------------	---

---

### Description

#' Generic method for calculating the degree of a list of nodes in a BN or a DBN. Calls `bnlearn`'s [degree](#) underneath. I have to redefine the generic and mask the original for it to work on both `bn` and `dbn` objects without the user having to import `bnlearn`.

### Usage

```
degree(object, Nodes, ...)
```

### Arguments

<code>object</code>	a "bn", "dbn", "bn.fit" or "dbn.fit" object
<code>Nodes</code>	which nodes to check
<code>...</code>	additional parameters

### Value

the degree of the nodes



---

filtered_fold_dt	<i>Fold a dataset avoiding overlapping of different time series</i>
------------------	---

---

### Description

If the dataset that is going to be folded contains several different time series instances of the same process, folding it could introduce false rows with data from different time series. Given an id variable that labels the different instances of a time series inside a dataset and a desired size, this function folds the dataset and avoids mixing data from different origins in the same instance.

### Usage

```
filtered_fold_dt(dt, size, id_var, clear_id_var = TRUE)
```

### Arguments

dt	data.table to be folded
size	the size of the data.table
id_var	the variable that labels each individual instance of the time series
clear_id_var	boolean that decides whether or not the id_var column is deleted

### Value

the filtered data.table

### Examples

```
dt <- dbnR::motor[201:2500]
dt[, n_sec := rep(seq(46), each = 50)] # I'll create sequences of 50 instances each
f_dt <- dbnR::fold_dt(dt, size = 2)
dim(f_dt)
f_dt <- dbnR::filtered_fold_dt(dt, size = 2, id_var = "n_sec")
dim(f_dt) # The filtered folded dt has a row less for each independent sequence
```

---

filter_same_cycle	<i>Filter the instances in a data.table with different ids in each row</i>
-------------------	--

---

### Description

Given an id variable that labels the different instances of a time series inside a dataset, discard the rows that have values from more than 1 id.

### Usage

```
filter_same_cycle(f_dt, size, id_var)
```

**Arguments**

f_dt	folded data.table
size	the size of the data.table
id_var	the variable that labels each individual instance of the time series

**Value**

the filtered data.table

**Examples**

```
dt <- dbnR::motor[201:2500]
dt[, n_sec := rep(seq(46), each = 50)] # I'll create sequences of 50 instances each
f_dt <- dbnR::fold_dt(dt, size = 2)
f_dt[50, .SD, .SDcols = c("n_sec_t_0", "n_sec_t_1")]
f_dt <- dbnR::filter_same_cycle(f_dt, size = 2, id_var = "n_sec")
f_dt[50, .SD, .SDcols = c("n_sec_t_0", "n_sec_t_1")]
```

---

fitted.dbn.fit

*Extracts the fitted values of a DBN*


---

**Description**

Generic method for "dbn.fit" S3 objects. Calls bnlearn underneath.

**Usage**

```
## S3 method for class 'dbn.fit'
fitted(object, ...)
```

**Arguments**

object	the fitted network
...	additional parameters

**Value**

the fitted values of the network

---

fit_dbn_params	<i>Fits a markovian n DBN model</i>
----------------	-------------------------------------

---

**Description**

Fits the parameters of the DBN via MLE. The "mu" vector of means and the "sigma" covariance matrix are set as attributes of the dbn.fit object for future exact inference.

**Usage**

```
fit_dbn_params(net, f_dt, ...)
```

**Arguments**

net	the structure of the DBN
f_dt	a folded data.table
...	additional parameters for the <a href="#">bn.fit</a> function

**Value**

a "dbn.fit" S3 object with the fitted net

**Examples**

```
size = 3
dt_train <- dbnR::motor[200:2500]
net <- learn_dbn_struc(dt_train, size)
f_dt_train <- fold_dt(dt_train, size)
fit <- fit_dbn_params(net, f_dt_train, method = "mle-g")
```

---

fold_dt	<i>Widens the dataset to take into account the t previous time slices</i>
---------	---

---

**Description**

This function will widen the dataset to put the t previous time slices in each row, so that it can be used to learn temporal arcs in the second phase of the dmmhc.

**Usage**

```
fold_dt(dt, size)
```

**Arguments**

dt	the data.table to be treated
size	number of time slices to unroll. Markovian 1 would be size 2

**Value**

the extended data.table

**Examples**

```
data(motor)
size <- 3
f_dt <- fold_dt(motor, size)
```

---

forecast\_ts

*Performs forecasting with the GDBN over a dataset*

---

**Description**

Given a dbn.fit object, the size of the net and a folded dataset, performs a forecast over the initial evidence taken from the dataset.

**Usage**

```
forecast_ts(
  dt,
  fit,
  size = NULL,
  obj_vars,
  ini = 1,
  len = dim(dt)[1] - ini,
  rep = 1,
  num_p = 50,
  print_res = TRUE,
  plot_res = TRUE,
  mode = "exact",
  prov_ev = NULL
)
```

**Arguments**

dt	data.table object with the TS data
fit	dbn.fit object
size	number of time slices of the net. Deprecated, will be removed in the future
obj_vars	variables to be predicted
ini	starting point in the dataset to forecast.
len	length of the forecast
rep	number of times to repeat the approximate forecasting
num_p	number of particles in the approximate forecasting
print_res	if TRUE prints the mae and sd metrics of the forecast

plot\_res           if TRUE plots the results of the forecast  
 mode                "exact" for exact inference, "approx" for approximate  
 prov\_ev             variables to be provided as evidence in each forecasting step

**Value**

a list with the original time series values and the results of the forecast

**Examples**

```

size = 3
data(motor)
dt_train <- motor[200:900]
dt_val <- motor[901:1000]
obj <- c("pm_t_0")
net <- learn_dbn_struct(dt_train, size)
f_dt_train <- fold_dt(dt_train, size)
f_dt_val <- fold_dt(dt_val, size)
fit <- fit_dbn_params(net, f_dt_train, method = "mle-g")
res <- suppressWarnings(forecast_ts(f_dt_val, fit,
  obj_vars = obj, len = 10, print_res = FALSE, plot_res = FALSE))

```

---

generate\_random\_network\_exp

*Generate a random DBN and a sampled dataset*

---

**Description**

This function generates both a random DBN and a dataset that can be used to learn its structure from data. It's intended for experimental use.

**Usage**

```

generate_random_network_exp(
  n_vars,
  size,
  min_mu,
  max_mu,
  min_sd,
  max_sd,
  min_coef,
  max_coef,
  seed = NULL
)

```

**Arguments**

n_vars	number of desired variables per time-slice
size	desired size of the networks
min_mu	minimum mean allowed for the variables
max_mu	maximum mean allowed for the variables
min_sd	minimum standard deviation allowed for the variables
max_sd	maximum standard deviation allowed for the variables
min_coef	minimum coefficient allowed for the parent nodes
max_coef	maximum coefficient allowed for the parent nodes
seed	the seed of the experiment

**Value**

a list with the original network structure and the sampled dataset

---

learn_dbn_struct	<i>Learns the structure of a markovian n DBN model from data</i>
------------------	--

---

**Description**

Learns a gaussian dynamic Bayesian network from a dataset. It allows the creation of markovian n nets rather than only markov 1.

**Usage**

```
learn_dbn_struct(dt, size = 2, method = "dmmhc", f_dt = NULL, ...)
```

**Arguments**

dt	the data.frame or data.table to be used
size	number of time slices of the net. Markovian 1 would be size 2
method	the structure learning method of choice to use
f_dt	previously folded dataset, in case some specific rows have to be removed after the folding
...	additional parameters for <a href="#">rsmx2</a> function

**Value**

a "dbn" S3 object with the structure of the network

**Examples**

```
data("motor")
net <- learn_dbn_struct(motor, size = 3)
```

---

logLik.dbn	<i>Calculate the log-likelihood of a dynamic Bayesian network</i>
------------	---

---

**Description**

Generic method for calculating the log-likelihood of a "dbn" S3 object given some data. Calls bnlearn's [logLik](#) underneath.

**Usage**

```
## S3 method for class 'dbn'
logLik(object, dt, ...)
```

**Arguments**

object	the structure of the network
dt	the dataset to calculate the score of the network
...	additional parameters for the network scoring

**Value**

the log-likelihood score of the network

---

logLik.dbn.fit	<i>Calculate the log-likelihood of a dynamic Bayesian network</i>
----------------	---

---

**Description**

Generic method for calculating the log-likelihood of a "dbn.fit" S3 object given some data. Calls bnlearn's [logLik](#) underneath.

**Usage**

```
## S3 method for class 'dbn.fit'
logLik(object, dt, ...)
```

**Arguments**

object	the fitted network
dt	the dataset to calculate the score of the network
...	additional parameters for the network scoring

**Value**

the log-likelihood score of the network

---

<code>mean.dbn.fit</code>	<i>Average the parameters of multiple <code>dbn.fit</code> objects with identical structures</i>
---------------------------	--

---

### Description

Generic method for "dbn.fit" S3 objects. Calls `bnlearn` underneath.

### Usage

```
## S3 method for class 'dbn.fit'
mean(x, ...)
```

### Arguments

<code>x</code>	the fitted network
<code>...</code>	additional parameters

### Value

the averaged parameters

---

<code>motor</code>	<i>Multivariate time series dataset on the temperature of an electric motor</i>
--------------------	---

---

### Description

Data from several sensors on an electric motor that records different benchmark sessions of measurements at 2 Hz. The dataset is reduced to 3000 instances from the 60th session in order to include it in the package for testing purposes. For the complete dataset, refer to the source.

### Usage

```
data(motor)
```

### Format

An object of class `data.table` (inherits from `data.frame`) with 3000 rows and 11 columns.

### Source

Kaggle, <<https://www.kaggle.com/wkirgsn/electric-motor-temperature>>



---

mvn_inference	<i>Performs inference over a multivariate normal distribution</i>
---------------	---

---

## Description

Given some evidence, this function performs inference over a multivariate normal distribution. After converting a Gaussian linear network to its MVN form, this kind of inference can be performed. It's recommended to use `predict_dt` functions instead unless you need a more flexible inference method.

## Usage

```
mvn_inference(mu, sigma, evidence)
```

## Arguments

<code>mu</code>	the mean vector
<code>sigma</code>	the covariance matrix
<code>evidence</code>	a single row <code>data.table</code> or a named vector with the values and names of the variables given as evidence

## Value

a list with the posterior mean and covariance matrix

## Examples

```
size = 3
data(motor)
dt_train <- motor[200:2500]
dt_val <- motor[2501:3000]
obj <- c("pm_t_0")

net <- learn_dbn_struct(dt_train, size)
f_dt_train <- fold_dt(dt_train, size)
f_dt_val <- fold_dt(dt_val, size)
ev <- f_dt_val[1, .SD, .SDcols = obj]
fit <- fit_dbn_params(net, f_dt_train, method = "mle-g")

pred <- mvn_inference(calc_mu(fit), calc_sigma(fit), ev)
```

---

nodes	<i>Returns a list with the names of the nodes of a BN or a DBN</i>
-------	--

---

**Description**

Generic method for obtaining the names of the nodes in a BN or a DBN. Calls bnlearn's [nodes](#) underneath. I have to redefine the generic and mask the original for it to work on both bn and dbn objects without the user having to import bnlearn.

**Usage**

```
nodes(object, ...)
```

**Arguments**

object	a "bn", "dbn", "bn.fit" or "dbn.fit" object
...	additional parameters

**Value**

the names of the nodes

---

nodes<-	<i>Relabel the names of the nodes of a BN or a DBN</i>
---------	--

---

**Description**

Generic method for renaming the nodes in a BN or a DBN. Calls bnlearn's [nodes<-](#) underneath. I have to redefine the generic and mask the original for it to work on both bn and dbn objects without the user having to import bnlearn.

**Usage**

```
nodes(object) <- value
```

**Arguments**

object	a "bn", "dbn", "bn.fit" or "dbn.fit" object
value	a list with the new names

**Value**

the modified object

---

plot.dbn	<i>Plots a dynamic Bayesian network</i>
----------	---

---

**Description**

Generic method for plotting the "dbn" S3 objects. Calls [plot\\_dynamic\\_network](#) underneath.

**Usage**

```
## S3 method for class 'dbn'  
plot(x, ...)
```

**Arguments**

x	the structure of the network.
...	additional parameters for the visualization of a DBN

---

plot.dbn.fit	<i>Plots a fitted dynamic Bayesian network</i>
--------------	--

---

**Description**

Generic method for plotting the "dbn.fit" S3 objects. Calls [plot\\_dynamic\\_network](#) underneath.

**Usage**

```
## S3 method for class 'dbn.fit'  
plot(x, ...)
```

**Arguments**

x	the structure of the network.
...	additional parameters for the visualization of a DBN

---

plot\_dynamic\_network *Plots a dynamic Bayesian network in a hierarchical way*

---

### Description

To plot the DBN, this method first computes a hierarchical structure for a time slice and replicates it for each slice. Then, it calculates the relative position of each node with respect to his equivalent in the first slice. The result is a net where each time slice is ordered and separated from one another, where the leftmost slice is the oldest and the rightmost represents the present time. This function is also called by the generic plot function of "dbn" and "dbn.fit" S3 objects.

### Usage

```
plot_dynamic_network(  
  structure,  
  offset = 200,  
  subset_nodes = NULL,  
  reverse = FALSE  
)
```

### Arguments

structure	the structure or fit of the network.
offset	the blank space between time slices
subset_nodes	a vector containing the names of the subset of nodes to plot
reverse	reverse to the classic naming convention of the nodes. The oldest time-slice will now be t_0 and the most recent one t_n. Only for visualization purposes, the network is unmodified underneath. If using subset_nodes, remember that t_0 is now the oldest time-slice.

### Value

the visualization of the DBN

### Examples

```
size = 3  
dt_train <- dbnR::motor[200:2500]  
net <- learn_dbn_struc(dt_train, size)  
plot_dynamic_network(net)
```

---

plot\_static\_network     *Plots a Bayesian network in a hierarchical way*

---

### Description

This function calculates the levels of each node and then plots them in a hierarchical layout in `visNetwork`. Can be used in place of the generic plot function offered by `bnlearn` for "bn" and "bn.fit" S3 objects.

### Usage

```
plot_static_network(structure)
```

### Arguments

structure     the structure or fit of the network.

### Examples

```
dt_train <- dbnR::motor[200:2500]
net <- bnlearn::mmhc(dt_train)
plot_static_network(net)
fit <- bnlearn::bn.fit(net, dt_train, method = "mle-g")
plot_static_network(fit) # Works for both the structure and the fitted net
```

---

predict.dbn.fit     *Performs inference in every row of a dataset with a DBN*

---

### Description

Generic method for predicting a dataset with a "dbn.fit" S3 objects. Calls `predict_dt` underneath.

### Usage

```
## S3 method for class 'dbn.fit'
predict(object, ...)
```

### Arguments

object     a "dbn.fit" object  
...     additional parameters for the inference process

### Value

a data.table with the prediction results

---

predict_bn	<i>Performs inference over a fitted GBN</i>
------------	---

---

### Description

Performs inference over a Gaussian BN. It's thought to be used in a map for a data.table, to use as evidence each separate row. If not specifically needed, it's recommended to use the function [predict\\_dt](#) instead. This function is deprecated and will be removed in a future version.

### Usage

```
predict_bn(fit, evidence)
```

### Arguments

fit	the fitted bn
evidence	values of the variables used as evidence for the net

### Value

a data.table with the predictions

### Examples

```
size = 3
data(motor)
dt_train <- motor[200:2500]
dt_val <- motor[2501:3000]
net <- learn_dbn_struct(dt_train, size)
f_dt_train <- fold_dt(dt_train, size)
f_dt_val <- fold_dt(dt_val, size)
fit <- fit_dbn_params(net, f_dt_train, method = "mle-g")
res <- f_dt_val[, predict_bn(fit, .SD), .SDcols = c("pm_t_0", "coolant_t_0"), by = 1:nrow(f_dt_val)]
```

---

predict_dt	<i>Performs inference over a test dataset with a GBN</i>
------------	--

---

### Description

This function performs inference over each row of a folded data.table, plots the results and gives metrics of the accuracy of the predictions. Given that only a single row is predicted, the horizon of the prediction is at most 1. This function is also called by the generic predict method for "dbn.fit" objects. For long term forecasting, please refer to the [forecast\\_ts](#) function.

### Usage

```
predict_dt(fit, dt, obj_nodes, verbose = T, look_ahead = F)
```

**Arguments**

fit	the fitted bn
dt	the test dataset
obj_nodes	the nodes that are going to be predicted. They are all predicted at the same time
verbose	if TRUE, displays the metrics and plots the real values against the predictions
look_ahead	boolean that defines whether or not the values of the variables in <code>t_0</code> should be used when predicting, even if they are not present in <code>obj_nodes</code> . This decides if look-ahead bias is introduced or not.

**Value**

a data.table with the prediction results for each row

**Examples**

```
size = 3
data(motor)
dt_train <- motor[200:900]
dt_val <- motor[901:1000]

# With a DBN
obj <- c("pm_t_0")
net <- learn_dbn_struct(dt_train, size)
f_dt_train <- fold_dt(dt_train, size)
f_dt_val <- fold_dt(dt_val, size)
fit <- fit_dbn_params(net, f_dt_train, method = "mle-g")
res <- suppressWarnings(predict_dt(fit, f_dt_val, obj_nodes = obj, verbose = FALSE))

# With a Gaussian BN directly from bnlearn
obj <- c("pm")
net <- bnlearn::mmhc(dt_train)
fit <- bnlearn::bn.fit(net, dt_train, method = "mle-g")
res <- suppressWarnings(predict_dt(fit, dt_val, obj_nodes = obj, verbose = FALSE))
```

---

print.dbn

*Print method for "dbn" objects*


---

**Description**

Generic print method for "dbn" S3 objects. Calls bnlearn's print underneath

**Usage**

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

**Arguments**

x	the "dbn" object
...	additional parameters

---

print.dbn.fit	<i>Print method for "dbn.fit" objects</i>
---------------	---

---

**Description**

Generic print method for "dbn.fit" S3 objects. Calls bnlearn's print underneath

**Usage**

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

**Arguments**

x	the "dbn.fit" object
...	additional parameters

---

rbn.dbn.fit	<i>Simulates random samples from a fitted DBN</i>
-------------	---

---

**Description**

Generic method for "dbn.fit" S3 objects. Calls bnlearn's [rbn](#) underneath.

**Usage**

```
rbn.dbn.fit(x, n, ...)
```

**Arguments**

x	the fitted network
n	number of samples
...	additional parameters

**Value**

the sampled dataset



---

reduce_freq	<i>Reduce the frequency of the time series data in a data.table</i>
-------------	---

---

### Description

In a time series dataset, there is a time difference between one row and the next one. This function reduces the number of rows from its current frequency to the desired one by averaging batches of rows. Instead of the frequency in Hz, the number of seconds between rows is asked ( $\text{Hz} = 1/\text{s}$ ).

### Usage

```
reduce_freq(dt, obj_freq, curr_freq, id_var = NULL)
```

### Arguments

dt	the original data.table
obj_freq	the desired number of seconds between rows
curr_freq	the number of seconds between rows in the original dataset
id_var	optional variable that labels different time series in a dataset, to avoid averaging values from different processes

### Value

the data.table with the desired frequency

### Examples

```
# Let's assume that the dataset has a frequency of 4Hz, 0.25 seconds between rows
dt <- dbnR::motor
dim(dt)
# Let's change the frequency to 2Hz, 0.5 seconds between rows
dt <- reduce_freq(dt, obj_freq = 0.5, curr_freq = 0.2)
dim(dt)
```

---

residuals.dbn.fit	<i>Returns the residuals from fitting a DBN</i>
-------------------	---

---

### Description

Generic method for "dbn.fit" S3 objects. Calls bnlearn underneath.

### Usage

```
## S3 method for class 'dbn.fit'
residuals(object, ...)
```

**Arguments**

object            the fitted network  
 ...               additional parameters

**Value**

the residuals of fitting the network

---

score	<i>Computes the score of a BN or a DBN</i>
-------	--

---

**Description**

Generic method for computing the score of a BN or a DBN. Calls bnlearn's [nodes](#) underneath. I have to redefine the generic and mask the original for it to work on both bn and dbn objects without the user having to import bnlearn.

**Usage**

```
score(object, ...)
```

**Arguments**

object            a "bn" or "dbn" object  
 ...               additional parameters

**Value**

the score of the network

---

shift_values	<i>Move the window of values backwards in a folded dataset row</i>
--------------	--

---

**Description**

This function moves the values in  $t_0, t_1, \dots, t_{n-1}$  in a folded dataset row to  $t_1, t_2, \dots, t_n$ . All the variables in  $t_0$  will be imputed with NAs and the obtained row can be used to forecast up to any desired point.

**Usage**

```
shift_values(f_dt, row)
```

**Arguments**

f\_dt            a folded dataset  
row            the index of the row that is going to be processed

**Value**

a one row data.table the shifted values

**Examples**

```
dt <- dbnR::motor  
f_dt <- dbnR::fold_dt(dt, size = 2)  
s_row <- dbnR::shift_values(f_dt, row = 500)
```

---

sigma.dbn.fit            *Returns the standard deviation of the residuals from fitting a DBN*

---

**Description**

Generic method for "dbn.fit" S3 objects. Calls bnlearn underneath.

**Usage**

```
## S3 method for class 'dbn.fit'  
sigma(object, ...)
```

**Arguments**

object            the fitted network  
...            additional parameters

**Value**

the standard deviation residuals of fitting the network

---

`smooth_ts`*Performs smoothing with the GDBN over a dataset*

---

## Description

Given a `dbn.fit` object, the size of the net and a folded dataset, performs a smoothing of a trajectory. Smoothing is the opposite of forecasting: given a starting point, predict backwards in time to obtain the time series that generated that point.

## Usage

```
smooth_ts(  
  dt,  
  fit,  
  size = NULL,  
  obj_vars,  
  ini = dim(dt)[1],  
  len = ini - 1,  
  print_res = TRUE,  
  plot_res = TRUE,  
  prov_ev = NULL  
)
```

## Arguments

<code>dt</code>	data.table object with the TS data
<code>fit</code>	<code>dbn.fit</code> object
<code>size</code>	number of time slices of the net. Deprecated, will be removed in the future
<code>obj_vars</code>	variables to be predicted. Should be in the oldest time step
<code>ini</code>	starting point in the dataset to smooth
<code>len</code>	length of the smoothing
<code>print_res</code>	if TRUE prints the mae and sd metrics of the smoothing
<code>plot_res</code>	if TRUE plots the results of the smoothing
<code>prov_ev</code>	variables to be provided as evidence in each smoothing step. Should be in the oldest time step

## Value

a list with the original values and the results of the smoothing

**Examples**

```

size = 3
data(motor)
dt_train <- motor[200:900]
dt_val <- motor[901:1000]
obj <- c("pm_t_2")
net <- learn_dbn_struct(dt_train, size)
f_dt_train <- fold_dt(dt_train, size)
f_dt_val <- fold_dt(dt_val, size)
fit <- fit_dbn_params(net, f_dt_train, method = "mle-g")
res <- suppressWarnings(smooth_ts(f_dt_val, fit,
                                obj_vars = obj, len = 10, print_res = FALSE, plot_res = FALSE))

```

---

time\_rename

*Renames the columns in a data.table so that they end in '\_t\_0'*


---

**Description**

This will rename the columns in a data.table so that they end in '\_t\_0', which will be needed when folding the data.table. If any of the columns already ends in '\_t\_0', a warning will be issued and no further operation will be done. There is no need to use this function to learn a DBN unless some operation with the variable names wants to be done prior to folding a dataset.

**Usage**

```
time_rename(dt)
```

**Arguments**

dt                    the data.table to be treated

**Value**

the renamed data.table

**Examples**

```

data("motor")
dt <- time_rename(motor)

```

---

[[<-.dbn.fit                      *Replacement function for parameters inside DBNs*

---

### Description

Generic parameter replacement method for "dbn.fit" S3 objects. Calls bnlearn underneath.

### Usage

```
## S3 replacement method for class 'dbn.fit'
x[[name]] <- value
```

### Arguments

x	the fitted network
name	name of the node to replace its parameters
value	the new parameters

### Value

the modified network

---

\$<-.dbn.fit                      *Replacement function for parameters inside DBNs*

---

### Description

Generic parameter replacement method for "dbn.fit" S3 objects. Calls bnlearn underneath.

### Usage

```
## S3 replacement method for class 'dbn.fit'
x$name <- value
```

### Arguments

x	the fitted network
name	name of the node to replace its parameters
value	the new parameters

### Value

the modified network

# Index

## \* datasets

motor, 16

[[<- .dbn.fit, 30

\$<- .dbn.fit, 30

AIC, 3

AIC.dbn, 3

AIC.dbn.fit, 3

all.equal, 4

all.equal.dbn, 4

all.equal.dbn.fit, 4

as.character, 5

as.character.dbn, 5

BIC, 5, 6

BIC.dbn, 5

BIC.dbn.fit, 6

bn.fit, 11

calc\_mu, 6

calc\_sigma, 7

coef.dbn.fit, 8

degree, 8, 8

filter\_same\_cycle, 9

filtered\_fold\_dt, 9

fit\_dbn\_params, 11

fitted.dbn.fit, 10

fold\_dt, 11

forecast\_ts, 12, 22

generate\_random\_network\_exp, 13

learn\_dbn\_struct, 14

logLik, 15

logLik.dbn, 15

logLik.dbn.fit, 15

mean.dbn.fit, 16

motor, 16

mvn\_inference, 17

nodes, 18, 18, 26

nodes<-, 18

plot.dbn, 19

plot.dbn.fit, 19

plot\_dynamic\_network, 19, 20

plot\_static\_network, 21

predict.dbn.fit, 21

predict\_bn, 22

predict\_dt, 17, 21, 22, 22

print.dbn, 23

print.dbn.fit, 24

rbn, 24

rbn.dbn.fit, 24

reduce\_freq, 25

residuals.dbn.fit, 25

rsmx2, 14

score, 26

shift\_values, 26

sigma.dbn.fit, 27

smooth\_ts, 28

time\_rename, 29