Package 'dbnR'

September 29, 2021

Type Package

Title Dynamic Bayesian Network Learning and Inference

Version 0.7.1

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 and 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)

Imports bnlearn (>= 4.5), data.table (>= 1.12.4), Rcpp (>= 1.0.2), magrittr (>= 1.5), R6 (>= 2.4.1), methods (>= 3.6.0)

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

LinkingTo Rcpp

URL https://github.com/dkesada/dbnR

License GPL-3

Encoding UTF-8

LazyData true

RoxygenNote 7.1.2

NeedsCompilation yes

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

Maintainer David Quesada <dkesada@gmail.com>

Repository CRAN

Date/Publication 2021-09-29 11:00:02 UTC

R topics documented:

`acc_successions` *Returns a vector with the number of consecutive nodes in each level*

Description

This method processes the vector of node levels to get the position of each node inside the level. E.g. c(1,1,1,2,2,3,4,4,5,5) turns into c(1,2,3,1,2,1,1,2,1,2)

Usage

```
acc_successions(nodes, res = NULL, prev = 0, acc = 0)
```

Arguments

`nodes`	a vector with the level of each node
`res`	the accumulative results of the sub successions
`prev`	the level of the previous node processed
`acc`	the accumulator of the index in the current sub successions

Value

the vector of sub successions in each level

`add_attr_to_fit` *Adds the mu vector and sigma matrix as attributes to the bn.fit or dbn.fit object*

Description

Adds the mu vector and sigma matrix as attributes to the bn.fit or dbn.fit object to allow performing exact MVN inference on both cases.

Usage

```
add_attr_to_fit(fit)
```

Arguments

`fit`	a fitted bn or dbn

Value

the fitted net with attributes

`approximate_inference` *Performs approximate inference forecasting with the GDBN over a data set*

Description

Given a bn.fit object, the size of the net and a data.set, performs approximate forecasting with bnlearns cpdist function over the initial evidence taken from the data set.

Usage

```
approximate_inference(dt, fit, size, obj_vars, ini, rep, len, num_p)
```

Arguments

`dt`	data.table object with the TS data
`fit`	bn.fit object
`size`	number of time slices of the net
`obj_vars`	variables to be predicted
`ini`	starting point in the data set to forecast.
`rep`	number of repetitions to be performed of the approximate inference
`len`	length of the forecast
`num_p`	number of particles to be used by bnlearn

Value

the results of the forecast

`approx_prediction_step` *Performs approximate inference in a time slice of the dbn*

Description

Given a bn.fit object and some variables, performs particle inference over such variables in the net for a given time slice.

Usage

```
approx_prediction_step(fit, variables, particles, n = 50)
```

Arguments

`fit`	bn.fit object
`variables`	variables to be predicted
`particles`	a list with the provided evidence
`n`	the number of particles to be used by bnlearn

Value

the inferred particles

`bn_translate_exp` *Experimental function that translates a natPosition vector into a DBN network.*

Description

Experimental function that translates a natPosition vector into a DBN network.

Usage

```
bn_translate_exp(ps, ordering_raw, n_arcs, nodes)
```

Arguments

`ps`	a position vector of natural numbers
`ordering_raw`	the ordering of the variables
`n_arcs`	the total number of arcs
`nodes`	the name of all the nodes in the network

Value

a bn object

`calc_mu` *Calculate the mu vector of means of a Gaussian linear network. Front end of a C++ function.*

Description

Calculate the mu vector of means of a Gaussian linear network. 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")
mu <- calc_mu(fit)
```

`calc_mu_cpp` *Calculate the mu vector of means of a Gaussian linear network. This is the C++ backend of the function.*

Description

Calculate the mu vector of means of a Gaussian linear network. This is the C++ backend of the function.

Usage

```
calc_mu_cpp(fit, order)
```

Arguments

`fit`	a bn.fit object as a Rcpp::List
`order`	a topological ordering of the nodes as a vector of strings

Value

the map with the nodes and their mu. Returns as a named numeric vector

---

`calc_sigma` *Calculate the sigma covariance matrix of a Gaussian linear network. Front end of a C++ function.*

Description

Calculate the sigma covariance matrix of a Gaussian linear network. Front end of a C++ function.

Usage

```
calc_sigma(fit)
```

Arguments

`fit`	a bn.fit or dbn.fit object

Value

a 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")
sigma <- calc_sigma(fit)
```

`calc_sigma_cpp` *Calculate the sigma covariance matrix of a Gaussian linear network. This is the C++ backend of the function.*

Description

Calculate the sigma covariance matrix of a Gaussian linear network. This is the C++ backend of the function.

Usage

```
calc_sigma_cpp(fit, order)
```

Arguments

`fit`	a bn.fit object as a Rcpp::List
`order`	a topological ordering of the nodes as a vector of strings

Value

the covariance matrix

`Causlist` *This file contains all the classes needed for the PSOHO structure learning algorithm. It was implemented as an independent package in https://github.com/dkesada/PSOHO and then merged into dbnR. All the original source files are merged into one to avoid bloating the R/ folder of the package.*

Description

Constructor of the 'Causlist' class

Arguments

`ordering`	a vector with the names of the nodes in t_0
`size`	number of timeslices of the DBN

Details

The classes are now not exported because the whole algorithm is encapsulated inside the package and only the resulting dbn structure is wanted. As a result, many security checks have been omitted. R6 class that defines causal lists in the PSO

The causal lists will be the base of the positions and the velocities in the pso part of the algorithm.

Value

A new 'causlist' object

Fields

`cl` List of causal units

`size` Size of the DBN

`ordering` String vector defining the order of the nodes in a timeslice

check_time0_formatted *Checks if the vector of names are time formatted to t0*

Description

This will check if the names are properly time formatted in t_0 to be folded into more time slices. A vector is well formatted in t_0 when all of its column names end in '_t_0'.

Usage

```
check_time0_formatted(obj)
```

Arguments

obj	the vector of names

Value

TRUE if it is well formatted. FALSE in other case.

cl_to_arc_matrix_cpp *Create a matrix with the arcs defined in a causlist object*

Description

Create a matrix with the arcs defined in a causlist object

Usage

```
cl_to_arc_matrix_cpp(cl, ordering, rows)
```

Arguments

cl	a causal list
ordering	a list with the order of the variables in t_0
rows	number of arcs in the network

Value

a list with a CharacterVector and a NumericVector

create_blacklist *Creates the blacklist of arcs from a folded data.table*

Description

This will create the blacklist of arcs that are not to be learned in the second phase of the dmmhc. This includes arcs backwards in time or inside time-slices.

Usage

```
create_blacklist(name, size, acc = NULL, slice = 1)
```

Arguments

name	the names of the first time slice, ended in _t_0
size	the number of time slices of the net. Markovian 1 would be size 2
acc	accumulator of the results in the recursion
slice	current time slice that is being processed

Value

the two column matrix with the blacklisted arcs

create_causlist_cpp *Create a causal list from a DBN. This is the C++ backend of the function.*

Description

Create a causal list from a DBN. This is the C++ backend of the function.

Usage

```
create_causlist_cpp(cl, net, size, ordering)
```

Arguments

cl	an initialized causality list
net	a dbn object treated as a list of lists
size	the size of the DBN
ordering	a list with the order of the variables in t_0

Value

a list with a CharacterVector and a NumericVector

`create_natcauslist_cpp` *Create a natural causal list from a DBN. This is the C++ backend of the function.*

Description

Create a natural causal list from a DBN. This is the C++ backend of the function.

Usage

```
create_natcauslist_cpp(cl, net, ordering)
```

Arguments

`cl`	an initialized causality list
`net`	a dbn object treated as a list of lists
`ordering`	a vector with the names of the variables in order

Value

the natCauslist equivalent to the DBN

`crop_names_cpp` *If the names of the nodes have "_t_0" appended at the end, remove it*

Description

If the names of the nodes have "_t_0" appended at the end, remove it

Usage

```
crop_names_cpp(names)
```

Arguments

`names`	a vector with the names of the nodes in t_0

Value

the vector with the names cropped

cte_times_vel_cpp *Multiply a Velocity by a constant real number*

Description

Multiply a Velocity by a constant real number

Usage

```
cte_times_vel_cpp(k, vl, abs_op, max_op)
```

Arguments

k	the constant real number
vl	the Velocity's causal list
abs_op	the final number of 1,-1 operations
max_op	the maximum number of directions in the causal list

Value

a list with the Velocity's new causal list and number of operations

dmmhc *Learns the structure of a markovian n DBN model from data*

Description

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

Usage

```
dmmhc(dt, size = 2, f_dt = NULL, blacklist = NULL, intra = TRUE, ...)
```

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
f_dt	previously folded dataset, in case some specific rows have to be removed after the folding
blacklist	an optional matrix indicating forbidden arcs between nodes
intra	if TRUE, the intra-slice arcs of the network will be learnt. If FALSE, they will be ignored
...	additional parameters for rsmax2 function

Value

the structure of the net

`dynamic_ordering` *Gets the ordering of a single time slice in a DBN*

Description

This method gets the structure of a DBN, isolates the nodes of a single time slice and then gives a topological ordering of them.

Usage

```
dynamic_ordering(structure)
```

Arguments

`structure`	the structure of the network.

Value

the ordered nodes of t_0

`exact_inference` *Performs exact inference forecasting with the GDBN over a data set*

Description

Given a bn.fit object, the size of the net and a data.set, performs exact forecasting over the initial evidence taken from the data set.

Usage

```
exact_inference(dt, fit, size, obj_vars, ini, len, prov_ev)
```

Arguments

`dt`	data.table object with the TS data
`fit`	bn.fit object
`size`	number of time slices of the net
`obj_vars`	variables to be predicted
`ini`	starting point in the data set to forecast.
`len`	length of the forecast
`prov_ev`	variables to be provided as evidence in each forecasting step

Value

the results of the forecast

`exact_inference_backwards` *Performs exact inference smoothing with the GDBN over a data set*

Description

Given a bn.fit object, the size of the net and a data.set, performs exact smoothing over the initial evidence taken from the data set. Take notice that the smoothing is done backwards in time, as opposed to forecasting.

Usage

```
exact_inference_backwards(dt, fit, size, obj_vars, ini, len, prov_ev)
```

Arguments

`dt`	data.table object with the TS data
`fit`	bn.fit object
`size`	number of time slices of the net
`obj_vars`	variables to be predicted. Should be in the oldest time step
`ini`	starting point in the data set to smooth
`len`	length of the smoothing
`prov_ev`	variables to be provided as evidence in each forecasting step. Should be in the oldest time step

Value

the results of the smoothing

`exact_prediction_step` *Performs exact inference in a time slice of the dbn*

Description

Given a bn.fit object and some variables, performs exact MVN inference over such variables in the net for a given time slice.

Usage

```
exact_prediction_step(fit, variables, evidence)
```

Arguments

fit	list with the mu and sigma of the MVN model
variables	variables to be predicted
evidence	a list with the provided evidence

Value

the inferred particles

---

expand_time_nodes *Extends the names of the nodes in t_0 to t_(max-1)*

Description

This method extends the names of the nodes to the given maximum and maintains the order of the nodes in each slice, so as to plotting the nodes in all slices relative to their homonyms in the first slice.

Usage

```
expand_time_nodes(name, acc, max, i)
```

Arguments

name	the names of the nodes in the t_0 slice
acc	accumulator of the resulting names in the recursion
max	number of time slices in the net
i	current slice being processed

Value

the extended names

filtered_fold_dt *Fold a dataset to a certain size and avoid overlapping of different time-series*

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 dataset

filter_same_cycle *Filter the instances in a data.table that have values of different ids in each row*

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 dataset

`fit_dbn_params` *Fits a markovian n DBN model*

Description

Fits the parameters of the DBN via MLE or BGE. 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 `bn.fit` function

Value

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")
```

`fold_dt` *Widens the dataset to take into account the t previous time slices*

Description

This 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
dt <- fold_dt(motor, size)
```

---

fold_dt_rec *Widens the dataset to take into account the t previous time slices*

Description

This 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. Recursive version not exported, the user calls from the handler 'fold_dt'

Usage

```
fold_dt_rec(dt, n_prev, size, slice = 1)
```

Arguments

dt	the data.table to be treated
n_prev	names of the previous time slice
size	number of time slices to unroll. Markovian 1 would be size 2
slice	the current time slice being treated. Should not be modified when first calling.

Value

the extended data.table

`forecast_ts` *Performs forecasting with the GDBN over a data set*

Description

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

Usage

```
forecast_ts(
  dt,
  fit,
  size,
  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
`obj_vars`	variables to be predicted
`ini`	starting point in the data set 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

the results of the forecast

Examples

```
size = 3
data(motor)
dt_train <- motor[200:2500]
dt_val <- motor[2501:3000]
obj <- c("pm_t_0")
net <- learn_dbn_struc(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")
res <- suppressWarnings(forecast_ts(f_dt_val, fit, size,
        obj_vars = obj, print_res = FALSE, plot_res = FALSE))
```

---

generate_random_network_exp

Experimental function that generates a random DBN and samples a dataset that defines it

---

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 dictionary with the original network structure and the sampled dataset

---

`initialize_cl_cpp` *Create a causality list and initialize it*

Description

Create a causality list and initialize it

Usage

```
initialize_cl_cpp(ordering, size)
```

Arguments

ordering	a list with the order of the variables in t_0
size	the size of the DBN

Value

a causality list

`init_cl_cpp` *Initialize the nodes vector*

Description

Initialize the vector in C++

Usage

```
init_cl_cpp(n_nodes)
```

Arguments

n_nodes	number of receiving nodes

Value

a list with the randomly initialized particles

init_list_cpp *Initialize the particles*

Description

Initialize the particles

Usage

```
init_list_cpp(nodes, size, n_inds)
```

Arguments

nodes	the names of the nodes
size	the size of the DBN
n_inds	the number of particles

Value

a list with the randomly initialized particles

learn_dbn_struc *Learns the structure of a markovian n DBN model from data*

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_struc(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 rsmax2 function

Value

the structure of the net

Examples

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

`merge_nets` *Merges and replicates the arcs in the static BN into all the time-slices in the DBN*

Description

This will join the static net and the state transition net by replicating the arcs in the static net in all the time slices.

Usage

```
merge_nets(net0, netCP1, size, acc = NULL, slice = 1)
```

Arguments

`net0`	the structure of the static net
`netCP1`	the state transition net
`size`	the number of time slices of the net. Markovian 1 would be size 2
`acc`	accumulator of the results in the recursion
`slice`	current time slice that is being processed

Value

the merged nets

`motor` *Multivariate time series dataset on the temperature of an electric motor*

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` *Performs inference over a multivariate normal distribution*

Description

Performs inference over a multivariate normal distribution given some evidence. After converting a Gaussian linear network to its MVN form, this kind of inference can be performed. It's recommended to use the `predict_bn` or `predict_dt` functions instead unless you need the posterior mean vector and covariance matrix.

Usage

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

Arguments

`mu`	the mean vector
`sigma`	the covariance matrix
`evidence`	a named vector with the values and names of the variables given as evidence

Value

the posterior mean and covariance matrix

Examples

```
as_named_vector <- function(dt){
  res <- as.numeric(dt)
  names(res) <- names(dt)

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

net <- learn_dbn_struc(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")

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

`natCauslist` *This file contains all the classes needed for the natPSOHO structure learning algorithm. It was implemented as an independent package in https://github.com/dkesada/natPSOHO and then merged into dbnR. All the original source files are merged into one to avoid bloating the R/ folder of the package.*

Description

Constructor of the 'natCauslist' class

Arguments

`ordering`	a vector with the names of the nodes in t_0
`ordering_raw`	a vector with the names of the nodes without the appended "_t_0"

Details

The classes are now not exported because the whole algorithm is encapsulated inside the package and only the resulting dbn structure is wanted. As a result, many security checks have been omitted. R6 class that defines causal lists in the PSO

The causal lists will be the base of the positions and the velocities in the pso part of the algorithm. They will not have the same structure as their binary counterparts, but their class skeleton will serve as a base.

Value

A new 'natCauslist' object

Fields

`cl` List of causal units

`ordering` String vector defining the order of the nodes in t_0

`ordering` String vector defining the order of the nodes without the appended "_t_0"

natcl_to_arc_matrix_cpp *Create a matrix with the arcs defined in a causlist object*

Description

Create a matrix with the arcs defined in a causlist object

Usage

```
natcl_to_arc_matrix_cpp(cl, ordering, rows)
```

Arguments

cl	a causal list
ordering	a list with the order of the variables in t_0
rows	number of arcs in the network

Value

a StringMatrix with the parent nodes and the children nodes

natParticle *R6 class that defines a Particle in the PSO algorithm*

Description

Constructor of the 'natParticle' class

Evaluate the score of the particle's position

Evaluate the score of the particle's position. Updates the local best if the new one is better.

Update the position of the particle with the velocity

Update the position of the particle given the constants after calculating the new velocity

Arguments

nodes	a vector with the names of the nodes
ordering	a vector with the names of the nodes in t_0
ordering_raw	a vector with the names of the nodes without the appended "_t_0"
max_size	maximum number of timeslices of the DBN
v_probs	vector of probabilities for the velocity sampling
p	parameter of the truncated geometric distribution
score	bnlearn score function used

dt	dataset to evaluate the fitness of the particle
in_cte	parameter that varies the effect of the inertia
gb_cte	parameter that varies the effect of the global best
gb_ps	position of the global best
lb_cte	parameter that varies the effect of the local best
r_probs	vector that defines the range of random variation of gb_cte and lb_cte

Details

A particle has a Position, a Velocity and a local best

Value

A new 'natParticle' object

The score of the current position

Fields

`ps` position of the particle

`cl` velocity of the particle

`velocity` that takes the particle to the global best

`velocity` that takes the particle to the local best

`lb` local best score obtained

`lb_ps` local best position found

`score` bnlearn score function used

`natPosition` *R6 class that defines DBNs as vectors of natural numbers*

Description

Constructor of the 'natPosition' class

Translate the vector into a DBN network

Uses this object private cl and transforms it into a DBN.

Add a velocity to the position

Given a natVelocity object, add it to the current position.

Return the static node ordering

This function takes as input a dbn and return the node ordering of the variables inside a timeslice. This ordering is needed to understand a position vector.

Translate a DBN into a position vector

This function takes as input a network from a DBN and transforms the structure into a vector of natural numbers if it is a valid DBN. Valid DBNs have only inter-timeslice edges and only allow variables in t_0 to have parents.

Generates a random position

This function takes as input the number of variables, the maximum size and the parameter p and returns a random position with arcs sampled either from the uniform distribution or from a truncated geometric distribution. Much faster than the binary implementation with lists of lists and random bn generation into translation.

Recount the number of arcs in the cl

Arguments

`nodes`	a vector with the names of the nodes
`ordering`	a vector with the names of the nodes in t_0
`ordering_raw`	a vector with the names of the nodes without the appended "_t_0"
`max_size`	Maximum number of timeslices of the DBN
`vl`	a natVelocity object
`net`	a dbn object
`n_vars`	the number of variables in t_0
`p`	the parameter of the truncated geometric sampler. If lesser or equal to 0, a uniform distribution will be used instead.

Details

A natPosition represents a single HO-DBN structure with a vector. Its function is to encode the solutions in the PSO framework. Each particle will have a position.

Value

A new 'natPosition' object

a dbn object

the ordering of the nodes in t_0

a random position

the number of arcs

Fields

`n_arcs` Number of arcs in the network

`max_size` Maximum number of timeslices of the DBN

`p` Parameter of the sampling truncated geometric distribution

`nodes` Names of the nodes in the network

`natPsoCtrl` *R6 class that defines the PSO controller*

Description

Constructor of the 'natPsoCtrl' class

If the names of the nodes have "_t_0" appended at the end, remove it

Initialize the particles for the algorithm to random positions and velocities.

Arguments

`n_it`	maximum number of iterations of the pso algorithm
`in_cte`	parameter that varies the effect of the inertia
`gb_cte`	parameter that varies the effect of the global best
`lb_cte`	parameter that varies the effect of the local best
`r_probs`	vector that defines the range of random variation of gb_cte and lb_cte
`cte`	boolean that defines whether the parameters remain constant or vary as the execution progresses
`nodes`	a vector with the names of the nodes
`ordering`	a vector with the names of the nodes in t_0
`max_size`	maximum number of timeslices of the DBN
`n_inds`	number of particles that the algorithm will simultaneously process
`v_probs`	vector that defines the random velocity initialization probabilities
`p`	parameter of the truncated geometric distribution for sampling edges
`score`	bnlearn score function used

Details

The controller will encapsulate the particles and run the algorithm. This time, it extends the class "PsoCtrl" in the "structure_learning_psoho.R" file, because both controllers are practically the same. The particles, positions and velocities are too different to extend one another though.

Value

A new 'natPsoCtrl' object

the ordering with the names cropped

natPsoho *Learn a DBN structure with a PSO approach*

Description

Given a dataset and the desired Markovian order, this function returns a DBN structure ready to be fitted. Original algorithm at https://link.springer.com/chapter/10.1007/978-3-030-86271-8_14

Usage

```
natPsoho(
  dt,
  size,
  f_dt = NULL,
  n_inds = 50,
  n_it = 50,
  in_cte = 1,
  gb_cte = 0.5,
  lb_cte = 0.5,
  v_probs = c(10, 65, 25),
  r_probs = c(-0.5, 1.5),
  score = "bge",
  p = 0.06,
  cte = TRUE
)
```

Arguments

dt	a data.table with the data of the network to be trained
size	Maximum number of timeslices of the DBN allowed. Markovian order 1 equals size 2, and so on
f_dt	previously folded dataset, in case some specific rows have to be removed after the folding
n_inds	Number of particles used in the algorithm
n_it	Maximum number of iterations that the algorithm can perform
in_cte	parameter that varies the effect of the inertia
gb_cte	parameter that varies the effect of the global best
lb_cte	parameter that varies the effect of the local best
v_probs	vector that defines the random velocity initialization probabilities
r_probs	vector that defines the range of random variation of gb_cte and lb_cte
score	bnlearn score function used
p	parameter of the truncated geometric distribution for sampling edges
cte	a boolean that determines whether the inertia, global best and local best parameters remain constant or vary as the algorithm progresses. Inertia and local best values decrease as the global best increases, to favor exploration at first and exploitation at the end

Value

A 'dbn' object with the structure of the best network found

`natVelocity` *R6 class that defines velocities in the PSO*

Description

Constructor of the 'natVelocity' class. Only difference with the natCauslist one is that it has a negative cl attribute.

Getter of the abs_op attribute.

return the number of operations that the velocity performs

Setter of the abs_op attribute. Intended for inside use only. This should be a 'protected' function in Java-like OOP, but there's no such thing in R6. This function should not be used from outside the package.

Randomizes the Velocity's directions.

Given two positions, returns the velocity that gets the first position to the other one.

Add both velocities directions

Multiply the Velocity by a constant real number

This function multiplies the Velocity by a constant real number. It is non deterministic by definition. When calculating k*|V|, the result will be floored and bounded to the set [-max_op, max_op], where max_op is the maximum number of arcs that can be present in the network.

Arguments

`ordering`	a vector with the names of the nodes in t_0
`ordering_raw`	a vector with the names of the nodes without the appended "_t_0"
`max_size`	maximum number of timeslices of the DBN
`n`	the new number of operations that the velocity performs
`probs`	the weight of each value -1,0,1. They define the probability that each of them will be picked
`p`	the parameter of the geometric distribution
`ps1`	the origin natPosition object
`ps2`	the objective natPosition object
`vl`	a Velocity object
`k`	a real number

Details

The velocities will be defined as two natural vectors where each element in them represents the arcs from a temporal family of nodes to a receiving node. 1-bits in the binary representation of this number represent arc additions/deletions

Value

A new 'natVelocity' object

the natVelocity that gets the ps1 to ps2

Fields

`abs_op` Total number of operations 1 or -1 in the velocity

`max_size` Maximum number of timeslices of the DBN

`cl_neg` Negative part of the velocity

---

nat_cte_times_vel_cpp *Multiply a Velocity by a constant real number*

Description

Multiply a Velocity by a constant real number

Usage

```
nat_cte_times_vel_cpp(k, vl, vl_neg, abs_op, max_size)
```

Arguments

`k`	the constant real number
`vl`	the Velocity's positive causal list
`vl_neg`	the Velocity's negative causal list
`abs_op`	the final number of 1,-1 operations
`max_size`	the maximum size of the network

Value

the new total number of operations

nat_pos_minus_pos_cpp *Subtracts two natPositions to obtain the natVelocity that transforms ps1 into ps2*

Description

Subtracts two natPositions to obtain the natVelocity that transforms ps1 into ps2

Usage

```
nat_pos_minus_pos_cpp(ps1, ps2, vl, vl_neg)
```

Arguments

ps1	the first position's causal list
ps2	the second position's causal list
vl	the natVelocity's positive causal list
vl_neg	the natVelocity's negative causal list

Value

the velocity's causal lists by reference and the number of operations by return

nat_pos_plus_vel_cpp *Add a velocity to a position*

Description

Add a velocity to a position

Usage

```
nat_pos_plus_vel_cpp(cl, vl, vl_neg, n_arcs)
```

Arguments

cl	the position's causal list
vl	the velocity's positive causal list
vl_neg	velocity's negative causal list
n_arcs	number of arcs present in the position. Remainder: can't return integers by reference, they get casted to 1 sized vectors

Value

the new position by reference and the new number of arcs by return

nat_vel_plus_vel_cpp *Adds two natVelocities*

Description

Adds two natVelocities represented as two numeric vectors: one with the positive part and one with the negative part. Adding them is a process that does a bitwise 'or' with both the positive and negative parts of the two velocities, adjusts the new abs_op, removes duplicated arcs in the final velocity by using a bitwise 'xor' with both parts and adjusts the final abs_op. The results are returned via modifying the original vl1 and vl1_neg by reference and returning the final abs_op normally. I can't have an integer edited by reference because it automatically gets casted and cannot be used to return values.

Usage

```
nat_vel_plus_vel_cpp(vl1, vl1_neg, vl2, vl2_neg, abs_op1, abs_op2)
```

Arguments

vl1	the first Velocity's positive part
vl1_neg	the first Velocity's negative part
vl2	the second Velocity's positive part
vl2_neg	the first Velocity's negative part
abs_op1	the number of 1,-1 operations in the first velocity
abs_op2	the number of 1,-1 operations in the second velocity

Value

the total number of resulting operations

nodes_gen_exp *Generates the names of the nodes in t_0 and in all the network*

Description

Given the names of the desired variables, this function generates the names of the variables in a DBN without needing a previous dataset. It's just a wrapper around the 'fold_dt' function.

Usage

```
nodes_gen_exp(ordering, size)
```

Arguments

ordering	the names of the variables
size	the desired size of the dbn

Value

a dictionary with the variable names in t_0 and in all other time slices

---

node_levels *Defines a level for every node in the net*

Description

Calculates the levels in which the nodes will be distributed when plotting the structure. This level is defined by their parent nodes: a node with no parents will always be in the level 0. Subsequently, the level of a node will be one more of the maximum level of his parents.

Usage

```
node_levels(net, order, lvl = 1, acc = NULL)
```

Arguments

net	the structure of the network.
order	a topological order of the nodes, with the orphan nodes in the first place. See node.ordering
lvl	current level being processed
acc	accumulator of the nodes already processed

Value

a matrix with the names of the nodes in the first row and their level on the second

one_hot *One hot encoder for natural numbers without the 0.*

Description

Given a natural number, return the natural number equivalent to its one-hot encoding. Examples: 3 -> 100 -> 4, 5 -> 10000 -> 16

Usage

```
one_hot(nat)
```

Arguments

nat	the natural number to convert

Value

the converted number

one_hot_cpp *One-hot encoder for natural numbers without the 0*

Description

Given a natural number, return the natural number equivalent to its one-hot encoding. Instead of pow, the '«' operator will be used. Examples: 3 -> 100 -> 4, 5 -> 10000 -> 16

Usage

```
one_hot_cpp(nat)
```

Arguments

nat	the natural number to convert

Value

the converted number

ordering_gen_exp *Generates the names of n variables.*

Description

Given the total number of variables, this function generates a character vector with variables named "Xi", where i is a number in the interval [0,n-1]

Usage

```
ordering_gen_exp(n)
```

Arguments

n	the total number of variables desired

Value

a character vector with the variable names

Particle *R6 class that defines a Particle in the PSO algorithm*

Description

Constructor of the 'Particle' class

Evaluate the score of the particle's position

Evaluate the score of the particle's position. Updates the local best if the new one is better.

Update the position of the particle with the velocity

Update the position of the particle given the constants after calculating the new velocity

Arguments

ordering	a vector with the names of the nodes in t_0
size	number of timeslices of the DBN
v_probs	vector that defines the random velocity initialization probabilities
score	bnlearn score function used
dt	dataset to evaluate the fitness of the particle
in_cte	parameter that varies the effect of the inertia
gb_cte	parameter that varies the effect of the global best
gb_ps	position of the global best
lb_cte	parameter that varies the effect of the local best
r_probs	vector that defines the range of random variation of gb_cte and lb_cte

Details

A particle has a Position, a Velocity and a local best

Value

A new 'Particle' object

The score of the current position

Fields

`ps` position of the particle

`cl` velocity of the particle

`lb` local best score obtained

`lb_ps` local best position found

`score` bnlearn score function used

`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.

Usage

```
plot_dynamic_network(structure, offset = 200)
```

Arguments

`structure`	the structure or fit of the network.
`offset`	the blank space between time slices

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_network *Plots a Bayesian networks in a hierarchical way*

Description

Calculates the levels of each node and then plots them in a hierarchical layout in visNetwork.

Usage

```
plot_network(structure)
```

Arguments

structure	the structure or fit of the network.

Examples

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

Position *R6 class that defines DBNs as causality lists*

Description

Constructor of the 'causlist' class

Translate the causality list into a DBN network

Uses this object private causality list and transforms it into a DBN.

Add a velocity to the position

Given a Velocity object, add it to the current position.

Given another position, returns the velocity that gets this position to the other.

Return the static node ordering

This function takes as input a dbn and return the node ordering of the variables inside a timeslice. This ordering is needed to understand a causal list.

Translate a DBN into a causality list

This function takes as input a network from a DBN and transforms the structure into a causality list if it is a valid DBN. Valid DBNs have only inter-timeslice edges and only allow variables in t_0 to have parents.

Generates a random DBN valid for causality list translation

This function takes as input a list with the names of the nodes and the desired size of the network and returns a random DBN structure.

Fixes a DBN structure to make it suitable for causality list translation

This function takes as input a DBN structure and removes the intra-timeslice arcs and the arcs that end in a node not in t_0.

Arguments

`vl`	a Velocity object
`ps`	a Position object return the Velocity that gets this position to the new one
`nodes`	a character vector with the names of the nodes in the net
`size`	the desired size of the DBN
`net`	the DBN structure
`nodes_t_0`	a vector with the names of the nodes in t_0

Details

A causality list has a list with causal units, a size representing the Markovian order of the network and a specific node ordering.

Value

A new 'causlist' object

a dbn object

the ordering of the nodes in t_0

a causlist object

a random dbn structure

the fixed network

Fields

`n_arcs` Number of arcs in the network

`nodes` Names of the nodes in the network

pos_minus_pos_cpp *Subtracts two Positions to obtain the Velocity that transforms one into the other*

Description

Subtracts two Positions to obtain the Velocity that transforms one into the other

Usage

```
pos_minus_pos_cpp(cl, ps, vl)
```

Arguments

cl	the first position's causal list
ps	the second position's causal list
vl	the Velocity's causal list

Value

a list with the Velocity's causal list and the number of operations

pos_plus_vel_cpp *Add a velocity to a position*

Description

Add a velocity to a position

Usage

```
pos_plus_vel_cpp(cl, vl, n_arcs)
```

Arguments

cl	the position's causal list
vl	the velocity's causal list
n_arcs	number of arcs present in the position

Value

a list with the modified position and the new number of arcs

predict_bn *Performs inference over a fitted GBN*

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.

Usage

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

Arguments

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

Value

the mean of the particles for each row

Examples

```
size = 3
data(motor)
dt_train <- motor[200:2500]
dt_val <- motor[2501:3000]
net <- learn_dbn_struc(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")
res <- f_dt_val[, predict_bn(fit, .SD), by = 1:nrow(f_dt_val)]
```

predict_dt *Performs inference over a test data set with a GBN*

Description

Performs inference over a test data set, plots the results and gives metrics of the accuracy of the results.

Usage

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

Arguments

fit	the fitted bn
dt	the test data set
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

Value

the prediction results

Examples

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

# With a DBN
obj <- c("pm_t_0")
net <- learn_dbn_struc(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")
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")
res <- suppressWarnings(predict_dt(fit, dt_val, obj_nodes = obj, verbose = FALSE))
```

PsoCtrl *R6 class that defines the PSO controller*

Description

Constructor of the 'PsoCtrl' class

Getter of the cluster attribute

Transforms the best position found into a bn structure and returns it

Main function of the pso algorithm.

Initialize the particles for the algorithm to random positions and velocities.

Evaluate the particles and update the global best

Modify the PSO parameters after each iteration

Arguments

n_it	maximum number of iterations of the pso algorithm
in_cte	parameter that varies the effect of the inertia
gb_cte	parameter that varies the effect of the global best
lb_cte	parameter that varies the effect of the local best
r_probs	vector that defines the range of random variation of gb_cte and lb_cte
cte	a boolean that determines whether the parameters remain constant or vary as the algorithm progresses. The increases and decreases are calculated as a function of the total number of iterations, decreasing until close to 0 and increasing until close to 1.
ordering	a vector with the names of the nodes in t_0
size	number of timeslices of the DBN
n_inds	number of particles that the algorithm will simultaneously process
v_probs	vector that defines the random velocity initialization probabilities
score	bnlearn score function used
dt	the dataset used to evaluate the position

Details

The controller will encapsulate the particles and run the algorithm

Value

A new 'PsoCtrl' object

the cluster attribute

the size attribute

Fields

parts list with all the particles in the algorithm

cl cluster for the parallel computations

n_it maximum number of iterations of the pso algorithm

in_cte parameter that varies the effect of the inertia

gb_cte parameter that varies the effect of the global best

lb_cte parameter that varies the effect of the local best

b_ps global best position found

b_scr global best score obtained

r_probs vector that defines the range of random variation of gb_cte and lb_cte

cte boolean that defines whether the parameters remain constant or vary as the execution progresses

in_var decrement of the inertia each iteration

gb_var increment of the global best parameter each iteration

lb_var increment of the local best parameter each iteration

psoho *Learn a DBN structure with a PSO approach*

Description

Given a dataset and the desired Markovian order, this function returns a DBN structure ready to be fitted. It requires a folded dataset. Original algorithm at https://doi.org/10.1109/BRC.2014.6880957

Usage

```
psoho(
  dt,
  size,
  f_dt = NULL,
  n_inds = 50,
  n_it = 50,
  in_cte = 1,
  gb_cte = 0.5,
  lb_cte = 0.5,
  v_probs = c(10, 65, 25),
  r_probs = c(-0.5, 1.5),
  score = "bge",
  cte = TRUE
)
```

Arguments

dt	a data.table with the data of the network to be trained
size	Number of timeslices of the DBN. Markovian order 1 equals size 2, and so on.
f_dt	previously folded dataset, in case some specific rows have to be removed after the folding
n_inds	Number of particles used in the algorithm.
n_it	Maximum number of iterations that the algorithm can perform.
in_cte	parameter that varies the effect of the inertia
gb_cte	parameter that varies the effect of the global best
lb_cte	parameter that varies the effect of the local best
v_probs	vector that defines the random velocity initialization probabilities
r_probs	vector that defines the range of random variation of gb_cte and lb_cte
score	bnlearn score function used
cte	a boolean that determines whether the inertia, global best and local best parameters remain constant or vary as the algorithm progresses. Inertia and local best values decrease as the global best increases, to favor exploration at first and exploitation at the end.

Value

A 'dbn' object with the structure of the best network found

`randomize_vl_cpp` *Randomize a velocity with the given probabilities*

Description

Randomize a velocity with the given probabilities

Usage

```
randomize_vl_cpp(vl, probs)
```

Arguments

`vl`	a velocity list
`probs`	the probabilities of each value in the set -1,0,1

Value

a velocity list with randomized values

`recount_arcs_exp` *Experimental function that recounts the number of arcs in the position*

Description

Experimental function that recounts the number of arcs in the position

Usage

```
recount_arcs_exp(ps)
```

Arguments

`ps`	a position vector of natural numbers

Value

the number of arcs

reduce_freq *Reduce the frequency of the time series data in a data.table*

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. Instead of the frequency in Hz, the number of seconds between rows is asked (Hz = 1/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

rename_nodes_cpp *Return a list of nodes with the time slice appended up to the desired size of the network*

Description

Return a list of nodes with the time slice appended up to the desired size of the network

Usage

```
rename_nodes_cpp(nodes, size)
```

Arguments

nodes	a list with the names of the nodes in the network
size	the size of the DBN

Value

a list with the renamed nodes in each timeslice

`smooth_ts` *Performs smoothing with the GDBN over a data set*

Description

Given a dbn.fit object, the size of the net and a folded data.set, 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,
  obj_vars,
  ini = dim(dt)[1],
  len = ini - 1,
  print_res = TRUE,
  plot_res = TRUE,
  prov_ev = NULL
)
```

Arguments

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

Value

the results of the smoothing

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.

Usage

```
time_rename(dt)
```

Arguments

dt	the data.table to be treated

Value

the renamed data.table

Examples

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

trunc_geom *Geometric distribution sampler truncated to a maximum*

Description

A geometric distribution sampler with probability 'p' restricted to values inside [1, max]. Because of this restriction, very low values of 'p' coupled with low 'max' return increasingly uniform populations in the interval [1, max].

Usage

```
trunc_geom(p, max)
```

Arguments

p	the parameter of the geometric distribution
max	the maximum value allowed to be sampled

Value

the sampled value

`Velocity` *R6 class that defines velocities affecting causality lists in the PSO*

Description

Getter of the abs_op attribute.

return the number of operations that the velocity performs

Setter of the abs_op attribute. Intended for inside use only. This should be a 'protected' function in Java-like OOP, but there's no such thing in R6. This function should not be used from outside the package.

Randomizes the Velocity's directions. If the seed provided is NULL, no seed will be used.

Given a position, returns the velocity that gets this position to the other.

Add both velocities directions

Multiply the Velocity by a constant real number

This function multiplies the Velocity by a constant real number. It is non deterministic by definition. When calculating k*|V|, the result will be floored and bounded to the set [-max_op, max_op], where max_op is the maximum number of arcs that can be present in the network.

Arguments

`n`	the new number of operations that the velocity performs
`probs`	the weight of each value -1,0,1. They define the probability that each of them will be picked
`seed`	the seed provided to the random number generation
`ps`	a Position object return the Velocity that gets this position to the new one
`vl`	a Velocity object
`k`	a real number

Details

The velocities will be defined as a causality list where each element in a causal unit is a pair (v, node) with v being either 0, 1 or -1. 0 means that arc remained the same, 1 means that arc was added and -1 means that arc was deleted.

Fields

`abs_op` Total number of operations 1 or -1 in the velocity

`vel_plus_vel_cpp` *Add two Velocities*

Description

Add two Velocities

Usage

```
vel_plus_vel_cpp(vl1, vl2, abs_op)
```

Arguments

`vl1`	the first Velocity's causal list
`vl2`	the second Velocity's causal list
`abs_op`	the final number of 1,-1 operations

Value

a list with the Velocity's causal list and the number of operations

Index