Package ‘dbnR’

March 25, 2020

Type Package

Title Dynamic Bayesian Network Learning and Inference

Version 0.3.4

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 a modification of Trabelsi (2013) <doi:10.1007/978-3-642-41398-8_34> dynamic max-min hill climbing algorithm for structure learning 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)

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 7.1.0

NeedsCompilation yes

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

Maintainer David Quesada <dkesada@gmail.com>

Repository CRAN

Date/Publication 2020-03-25 13:50:21 UTC
\section*{R topics documented:}

\begin{itemize}
  \item acc_successions \hfill \pageref{acc_successions}
  \item \texttt{add_attr_to_fit} \hfill \pageref{add_attr_to_fit}
  \item \texttt{approximate_inference} \hfill \pageref{approximate_inference}
  \item \texttt{approx_prediction_step} \hfill \pageref{approx_prediction_step}
  \item \texttt{calc_mu} \hfill \pageref{calc_mu}
  \item \texttt{calc_mu_cpp} \hfill \pageref{calc_mu_cpp}
  \item \texttt{calc_sigma} \hfill \pageref{calc_sigma}
  \item \texttt{calc_sigma_cpp} \hfill \pageref{calc_sigma_cpp}
  \item \texttt{check_time0_formatted} \hfill \pageref{check_time0_formatted}
  \item \texttt{create_blacklist} \hfill \pageref{create_blacklist}
  \item \texttt{dynamic_ordering} \hfill \pageref{dynamic_ordering}
  \item \texttt{exact_inference} \hfill \pageref{exact_inference}
  \item \texttt{exact_prediction_step} \hfill \pageref{exact_prediction_step}
  \item \texttt{expand_time_nodes} \hfill \pageref{expand_time_nodes}
  \item \texttt{fit_dbn_params} \hfill \pageref{fit_dbn_params}
  \item \texttt{fold_dt} \hfill \pageref{fold_dt}
  \item \texttt{fold_dt_rec} \hfill \pageref{fold_dt_rec}
  \item \texttt{forecast_ts} \hfill \pageref{forecast_ts}
  \item \texttt{learn_dbn_struc} \hfill \pageref{learn_dbn_struc}
  \item \texttt{merge_nets} \hfill \pageref{merge_nets}
  \item \texttt{motor} \hfill \pageref{motor}
  \item \texttt{mvn_inference} \hfill \pageref{mvn_inference}
  \item \texttt{node_levels} \hfill \pageref{node_levels}
  \item \texttt{plot_dynamic_network} \hfill \pageref{plot_dynamic_network}
  \item \texttt{plot_network} \hfill \pageref{plot_network}
  \item \texttt{predict_bn} \hfill \pageref{predict_bn}
  \item \texttt{predict_dt} \hfill \pageref{predict_dt}
  \item \texttt{time_rename} \hfill \pageref{time_rename}
\end{itemize}

\section*{Index}

\begin{center}
\begin{tabular}{ll}
acc_successions & \hfill Returns a vector with the number of consecutive nodes in each level
\end{tabular}
\end{center}

\section*{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)

\section*{Usage}

\begin{verbatim}
acc_successions(nodes, res = NULL, prev = 0, acc = 0)
\end{verbatim}
add_attr_to_fit

**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)`
approx_prediction_step

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

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
**calc_mu**

---

**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**

---

**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**

```r
calc_sigma(fit)
```

**Arguments**

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

**Value**

a numeric covariance matrix of the nodes

**Examples**

```r
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**

```r
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
check_time0_formatted  Checks if the vector of names are time formatted to t_0

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.

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
**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)
```

**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

**Value**

- the results of the forecast
**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
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**

```r
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
fold_dt_rec

Value
the extended data.table

Examples

```r
data(motor)
size <- 3
dt <- fold_dt(motor, size)
```

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

```r
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"
)
```

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

Value

the results of the forecast

Examples

```r
data(motor)
size = 3
obj <- c("pm_t_0", "torque_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))
```
**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, ...)`

**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
- `...`: additional parameters for `rsmax2` function

**Value**

the structure of the net

**Examples**

```r
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 4th 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 12 columns.

Source


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

```r
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", "torque_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[,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))
```

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

```r
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
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.
**predict_bn**

*Perform inference over a fitted GBN*

**Examples**

```r
dt_train <- dbnR::motor[200:2500]
obj <- c("pm", "torque")
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
```

**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**

```r
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

Perform 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**

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

# With a DBN
obj <- c("pm_t_0", "torque_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", "torque")
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))
```
time_rename

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)
Index

*Topic **datasets**
  
  motor, 14

  acc_successions, 2
  add_attr_to_fit, 3
  approx_prediction_step, 4
  approximate_inference, 3

  bn.fit, 10

  calc_mu, 5
  calc_mu_cpp, 5
  calc_sigma, 6
  calc_sigma_cpp, 6
  check_time0_formatted, 7
  create_blacklist, 7

  dynamic_ordering, 8

  exact_inference, 8
  exact_prediction_step, 9
  expand_time_nodes, 9

  fit_dbn_params, 10
  fold_dt, 10
  fold_dt_rec, 11
  forecast_ts, 11

  learn_dbn_struc, 13

  merge_nets, 13
  motor, 14
  mvn_inference, 14

  node.ordering, 15
  node_levels, 15

  plot_dynamic_network, 16
  plot_network, 16
  predict_bn, 14, 17
  predict_dt, 14, 17, 18

rsmax2, 13

  time_rename, 19