Package 'abn'

July 4, 2019

Version 2.1

Date 2019-06-06

Title Modelling Multivariate Data with Additive Bayesian Networks

Author Gilles Kratzer [aut, cre] (<https://orcid.org/0000-0002-5929-8935>),
Fraser Ian Lewis [aut],
Reinhard Furrer [ctb] (<https://orcid.org/0000-0002-6319-2332>),
Marta Pittavino [ctb] (<https://orcid.org/0000-0002-1232-1034>)

Maintainer Gilles Kratzer <gilles.kratzer@math.uzh.ch>

Imports methods, rjags

LinkingTo Rcpp, RcppArmadillo

Depends nnet, MASS, lme4, R (>= 2.15.1)

Suggests INLA, Rgraphviz, R.rsp

VignetteBuilder R.rsp

Additional_repositories https://inla.r-inla-download.org/R/stable/

SystemRequirements Gnu Scientific Library version >= 1.12

Description

Bayesian network analysis is a form of probabilistic graphical models which derives from empirical data a directed acyclic graph, DAG, describing the dependency structure between random variables. An additive Bayesian network model consists of a form of a DAG where each node comprises a generalized linear model, GLM. Additive Bayesian network models are equivalent to Bayesian multivariate regression using graphical modelling, they generalises the usual multivariable regression, GLM, to multiple dependent variables. 'abn' provides routines to help determine optimal Bayesian network models for a given data set, where these models are used to identify statistical dependencies in messy, complex data. The additive formulation of these models is equivalent to multivariate generalised linear modelling (including mixed models with iid random effects). The usual term to describe this model selection process is structure discovery. The core functionality is concerned with model selection - determining the most robust empirical model of data from interdependent variables. Laplace approximations are used to estimate goodness of fit metrics and model parameters, and wrappers are also included to the INLA package which can be obtained from <http://www.r-inla.org>. A comprehensive set of documented case studies, numerical accuracy/quality assurance exercises, and additional documentation are available from the 'abn' website <http://r-bayesian-networks.org>.

License GPL (>= 2)

LazyData true

URL http://r-bayesian-networks.org

RoxygenNote 6.0.1

NeedsCompilation yes

Repository CRAN

Date/Publication 2019-07-04 10:00:03 UTC

R topics documented:

. abn . *abn Package*

Description

abn is a collection of functions for fitting, selecting/learning, analysing, reporting Additive Bayesian Networks.

General overview

What is **abn**:

Bayesian network modelling is a data analysis technique which is ideally suited to messy, highly correlated and complex datasets. This methodology is rather distinct from other forms of statistical modelling in that its focus is on structure discovery - determining an optimal graphical model which describes the inter-relationships in the underlying processes which generated the data. It is a multivariate technique and can used for one or many dependent variables. This is a data driven approach, as opposed to, rely only on subjective expert opinion to determine how variables of interest are inter-related (for example: structural equation modelling).

The R package abn is designed to fit additive Bayesian models to observational datasets. It contains routines to score Bayesian Networks based on Bayesian or information-theoretic formulation of generalized linear models. It is equipped with exact search and greedy search algorithms to select the best network. The Bayesian implementation supports random effects to control for one layer clustering. It supports a possibly mixture of continuous, discrete and count data and input of prior knowledge at a structural level.

Author(s)

Fraser Iain Lewis and Gilles Kratzer

References

Lewis, F. I., and Ward, M. P. (2013). Improving epidemiologic data analyses through multivariate regression modelling. Emerging themes in epidemiology, 10(1), 4.

Kratzer, G., Pittavino, M, Lewis, F. I., and Furrer, R., (2017). abn: an R package for modelling multivariate data using additive Bayesian networks. R package version 1.1. https://CRAN.R-project.org/package=abn

Examples

```
## Citations:
citation('abn')
```

adg — *Dataset related to average daily growth performance and abattoir findings in pigs commercial production.*

Description

The case study dataset is about growth performance and abattoir findings in pigs commercial production in a selected set of 15 Canadian farms collected in March 1987.

Usage

```
adg
```

Format

An adapted data frame of the original dataset which consists of 341 observations of 8 variables and a grouping variable (farm).

farm farm ID.

female sex of the pig (1=female, 0=castrated).

age days elapsed from birth to slaughter (days).

adg average daily weight gain (grams).

pneumS presence of moderate to severe pneumonia.

eggs presence of fecal/gastrointestinal nematode eggs at time of slaughter.

wormCount count of nematodes in small intestine at time of slaughter.

livdam presence of liver damage (parasite-induced white spots).

AR presence of atrophic rhinitis.

References

Dohoo, Ian Robert, Wayne Martin, and Henrik Stryhn. Veterinary epidemiologic research. No. V413 DOHv. Charlottetown, Canada: AVC Incorporated, 2003.

buildscorecache — *Build a cache of goodness of fit metrics for each node in a DAG, possibly subject to user defined restrictions*

Description

Iterates over all valid parent combinations - subject to ban, retain and max. parent limits - for each node, or a subset of nodes, and computes a cache of log marginal likelihoods. This cache can then be used in different DAG structural search algorithms.

Usage

```
buildscorecache(data.df = NULL, data.dists = NULL, method = "bayes",
   group.var = NULL, adj.vars = NULL, cor.vars = NULL, dag.banned = NULL,
   dag.retained = NULL, max.parents = NULL, which.nodes=NULL,
   defn.res = NULL, centre = TRUE, dry.run = FALSE,
   control = list(), verbose = FALSE, ...)
```

Arguments

data.df	a data frame containing the data used for learning each node, binary variables must be declared as factors.
data.dists	a named list giving the distribution for each node in the network, see 'Details'.
method	should a "Bayes" or "mle" approach be used, see 'Details'.
group.var	only applicable for nodes to be fitted as a mixed model and gives the column name in data.df of the grouping variable which must be a factor denoting group membership.
adj.vars	a character vector giving the column names in data.df for which the network score has to be adjusted for, see 'Details'.
cor.vars	a character vector giving the column names in data.df for which a mixed model should be used to adjust for within group correlation or pure adjustment.
dag.banned	a matrix or a formula statement (see 'Details' for format) defining which arcs are not permitted - banned - see 'Details' for format. Note that colnames and rownames must be set, otherwise same row/column names as data.df will be assumed. If set as NULL an empty matrix is assumed.
dag.retained	a matrix or a formula statement (see 'Details' for format) defining which arcs are must be retained in any model search, see 'Details' for format. Note that colnames and rownames must be set, otherwise same row/column names as data.df will be assumed. If set as NULL an empty matrix is assumed.
max.parents	a constant or named list giving the maximum number of parents allowed, the list version allows this to vary per node.
which.nodes	a vector giving the column indices of the variables to be included, if ignored all variables are included.
defn.res	an optional user-supplied list of child and parent combinations, see 'Details'.
centre	should the observations in each Gaussian node first be standardised to mean zero and standard deviation one, defaults to TRUE.
dry.run	if TRUE then a list of the child nodes and parent combinations are returned but without estimation of node scores (log marginal likelihoods).
control	a list of control parameters. See 'Details'.
verbose	if TRUE then provides some additional output.
...	additional arguments passed for optimization.

Details

The function compute a cache of scores based on possible restictions (maximum complexity, retained and banned arcs). This cache of score could be used to select the optimal network in other function such as searchHeuristic or mostprobable. Two very different approaches are implemented: a Bayesian and a frequentist approache. They can be selected using the method argument.

If method="Bayes":

This function is used to calculate all individual node scores (log marginal likelihoods). This cache can then be fed into a model search algorithm. This function is very similar to fitabn - see that help page for details of the type of models used and in particular data.dists specification - but rather than fit a single complete DAG buildscorecache iterates over all different parent combinations for each node. There are three ways to customise the parent combinations through giving a matrix which contains arcs which are not allowed (banned), a matrix which contains arcs which must always be included (retained) and also a general complexity limit which restricts the maximum number of arcs allowed to terminate at a node (its number of parents), where this can differ from node to node. In these matrices, dag.banned and dag.retained, each row represents a node in the network, and the columns in each row define the parents for that particular node, see the example below for the specific format. If these are not supplied they are assumed to be empty matrices, i.e. no arcs banned or retained. Note that only rudimentary consistency checking is done here and some care should be taken not to provide conflicting restrictions in the ban and retain matrices.

The variable which.nodes is to allow the computation to be separated by node, for example over different cpus using say R CMD BATCH. This may useful and indeed likely essential with larger problems or those with random effects. Note that in this case the results must then be combined back into a list of identical format to that produced by an individual call to buildscorecache, comprising of all nodes (in same order as the columns in data.df) before sending to any search routines. Using dry.run can be useful here.

If method="mle":

This function is used to calculate all individual Information-Theoretic node scores. The possible Information-theoretic based network scores computed in buildscorecache are the maximum likelihood (mlik, called marginal likelihood in this context as it is computed node wise), the Akaike Information Criteria (aic), the Bayesian Information Criteria (bic) and the Minimum distance Length (mdl). The classical definitions of those metrics are given in Kratzer and Furrer (2018). This function computes a cache that can be fed into a model search algorithm. This function is very similar to fitabn with method="mle" - see that help page for details of the type of models used and in particular data.dists specification - but rather than fit a single complete DAG buildscorecache iterates over all admissible different parent combinations for each node. There are three ways to customise the parent combinations through giving a matrix which contains arcs which are not allowed (banned), a matrix which contains arcs which must always be included (retained) and also a general complexity limit which restricts the maximum number of arcs allowed to terminate at a node (its number of parents). In these matrices, dag.banned and dag.retained, each row represents a node in the network, and the columns in each row define the parents for that particular node, see the example below for the specific format. If these are not supplied they are assumed to be empty matrices, i.e. no arcs banned or retained. Note that only rudimentary consistency checking is done here and some care should be taken not to provide conflicting restrictions in the ban and retain matrices.

The control argument is a list that can supply any of the following components:

max.mode.error if the estimated modes from INLA differ by a factor of max.mode.error or more from those computed internally then results from INLA are replaced by those computed internally. To force INLA to always be used then max.mode.error=100, to force INLA to never be used max.mod.error=0. See 'Details'.

mean the prior mean for all the Gaussian additive terms for each node

prec the prior precision for all the Gaussian additive term for each node

loggam.shape the shape parameter in the Gamma distribution prior for the precision in a Gaussian node

loggam.inv.scale the inverse scale parameter in the Gamma distribution prior for the precision in a Gaussian node

max.iters total number of iterations allowed when estimating the modes in Laplace approximation

epsabs absolute error when estimating the modes in Laplace approximation for models with no random effects.

error.verbose logical,additional output in the case of errors occurring in the optimisation

epsabs.inner absolute error in the maximisation step in the (nested) Laplace approximation for each random effect term

max.iters.inner total number of iterations in the maximisation step in the nested Laplace approximation

finite.step.size suggested step length used in finite difference estimation of the derivatives for the (outer) Laplace approximation when estimating modes

hessian.params a numeric vector giving parameters for the adaptive algorithm which determines the optimal stepsize in the finite difference estimation of the hessian. First entry is the inital guess, second entry absolute error

max.iters.hessian integer, maximum number of iterations to use when determining an optimal finite difference approximation (nelder-mead)

max.hessian.error if the estimated log marginal likelihood when using an adaptive 5pt finite difference rule for the Hessian differs by more than max.hessian.error from when using an adaptive 3pt rule then continue to minimize the local error by switching to the Brent-Dekker root bracketing method, see 'Details'

factor.brent if using Brent-Dekker root bracketing method then define the outer most interval end points as the best estimate of h (stepsize) from the nelder-mead as (h/factor.brent,h*factor.brent)

maxiters.hessian.brent maximum number of iterations allowed in the Brent-Dekker method

num.intervals.brent the number of initial different bracket segments to try in the Brent-Dekker method

maxit integer given the maximum number of run for estimating network scores using an Iterative Reweighed Least Square algorithm.

tol real number giving the minimal tolerance expected to terminate the Iterative Reweighed Least Square algorithm to estimate network score.

The numerical routines used here are identical to those in `fitabn` and see that help page for further details and also the quality assurance section on the `http://r-bayesian-networks.org` of the abn website for more details.

Value

A named list of class abnCache.

children	a vector of the child node indexes (from 1) corresponding to the columns in data.df (ignoring any grouping variable)
node.defn	a matrix giving the parent combination
mlik	log marginal likelihood value for each node combination. If the model cannot be fitted then NA is returned.
error.code	if non-zero then either the root finding algorithm (glm nodes) or the maximisation algorithm (glmm nodes) terminated in an unusual way suggesting a possible unreliable result, or else the finite difference hessian estimation produced and error or warning (glmm nodes). NULL if method="mle".
error.code.desc	a textual description of the error.code. NULL if method="mle"
hessian.accuracy	An estimate of the error in the final mlik value for each parent combination - this is the absolute difference between two different adaptive finite difference rules where each computes the mlik value. NULL if method="mle"
data.df	a version of the original data (for internal use only in other functions such as mostprobable).
data.dists	the named list of nodes distributions (for internal use only in other functions such as mostprobable).
max.parents	the maximum number of parents (for internal use only in other functions such as mostprobable).
dag.retained	the matrix encoding the retained arcs (for internal use only in other functions such as search.heuristic).
dag.banned	the matrix encoding the banned arcs (for internal use only in other functions such as search.heuristic).
aic	aic value for each node combination. If the model cannot be fitted then NaN is returned. NULL if method="Bayes".
bic	bic value for each node combination. If the model cannot be fitted then NaN is returned. NULL if method="Bayes".
mdl	mdl value for each node combination. If the model cannot be fitted then NaN is returned. NULL if method="Bayes".

Author(s)

Fraser Iain Lewis and Gilles Kratzer

References

Lewis, F. I., and McCormick, B. J. J. (2012). Revealing the complexity of health determinants in resource poor settings. *American Journal Of Epidemiology*. DOI:10.1093/aje/KWS183).

Further information about **abn** can be found at:
http://r-bayesian-networks.org

Examples

```
## Not run:
#####################################################################
## Example 1
#####################################################################

mydat <- ex0.dag.data[,c("b1","b2","g1","g2","b3","g3")] ## take a subset of cols

## setup distribution list for each node
mydists <- list(b1="binomial", b2="binomial", g1="gaussian",
              g2="gaussian", b3="binomial", g3="gaussian")

# Structural constrains
# ban arc from b2 to b1
# always retain arc from g2 to g1

## parent limits
max.par <- list("b1"=2, "b2"=2, "g1"=2, "g2"=2, "b3"=2, "g3"=2)

## now build cache accordingly to the structural constrains

res.c <- buildscorecache(data.df=mydat, data.dists=mydists,
              dag.banned= ~b1|b2, dag.retained= ~g1|g2, max.parents=max.par)


## repeat but using R-INLA. The mlik's should be virtually identical.
## now build cache
res.inla <- buildscorecache(data.df=mydat, data.dists=mydists,
              dag.banned= ~b1|b2, dag.retained= ~g1|g2, max.parents=max.par,
              max.mode.error=100)

## plot comparison - very similar
plot(res.c$mlik, res.inla$mlik, pch="+")
abline(0, 1)


#####################################################################
## Example 2 - much bigger problem using glm - may take a while
#####################################################################

mydat <- ex2.dag.data ## this data comes with abn see ?ex2.dag.data

## setup distribution list for each node
mydists <- list(b1="binomial", g1="gaussian", p1="poisson",
                b2="binomial", g2="gaussian", p2="poisson",
                b3="binomial", g3="gaussian", p3="poisson",
                b4="binomial", g4="gaussian", p4="poisson",
                b5="binomial", g5="gaussian", p5="poisson",
                b6="binomial", g6="gaussian", p6="poisson")


## no explicit ban or retain restrictions set so dont need to supply ban
```

```
##  or retain matrices

## now build cache using internal code just for nodes 1,2 and 3
## e.g. "b1", "p1" and "g1"
mycache.c <- buildscorecache(data.df=mydat, data.dists=mydists,
                        max.parents=2, which.nodes=c(1:3))

######################################################################
## Example 3 - grouped data - random effects example e.g. glmm
######################################################################

mydat <- ex3.dag.data ## this data comes with abn see ?ex3.dag.data

mydists <- list(b1="binomial", b2="binomial", b3="binomial",
            b4="binomial", b5="binomial", b6="binomial", b7="binomial",
            b8="binomial", b9="binomial", b10="binomial",b11="binomial",
            b12="binomial", b13="binomial" )
max.par <- 2

## in this example INLA is used as default since these are glmm nodes
## when running this at node-parent combination 71 the default accuracy check on the
## INLA modes is exceeded (default is a max. of 10 percent difference from
## modes estimated using internal code) and a message is given that internal code
## will be used in place of INLA's results.

mycache <- buildscorecache(data.df=mydat, data.dists=mydists, group.var="group",
                        cor.vars=c("b1","b2","b3","b4","b5","b6","b7",
                                   "b8","b9","b10","b11","b12","b13"),
                        max.parents=max.par, which.nodes=c(1))

## End(Not run)
mydat <- ex0.dag.data[,c("b1","b2","g1","g2","b3","g3")] ## take a subset of cols

## setup distribution list for each node
mydists <- list(b1="binomial", b2="binomial", g1="gaussian",
             g2="gaussian", b3="binomial", g3="gaussian")

## now build cache of scores (goodness of fits for each node)
res.mle <- buildscorecache(data.df=mydat, data.dists=mydists,
                          max.parents=3, method="mle")
res.abn <- buildscorecache(data.df=mydat, data.dists=mydists,
                          max.parents=3, method="Bayes")

#plot(-res.mle$bic, res.abn$mlik)
```

---

compareDag *Compare two DAGs*

Description

Function that returns multiple graph metrics to compare two DAGs, known as confusion matrix or error matrix.

Usage

```
compareDag(ref, test, node.names = NULL)
```

Arguments

`ref`	a matrix or a formula statement (see details for format) defining the reference network structure, a directed acyclic graph (DAG). Note that rownames must be set or given in `node.names` if the DAG is given via a formula statement.
`test`	a matrix or a formula statement (see details for format) defining the test network structure, a directed acyclic graph (DAG). Note that rownames must be set or given in `node.names` if the DAG is given via a formula statement.
`node.names`	a vector of names if the DAGs are given via formula, see details.

Details

This R function returns standard Directed Acyclic Graph comparison metrics. In statistical classification those metrics are known as a confusion matrix or error matrix. Those metrics allows visualization of the difference between different DAGs. In the case where comparing TRUTH to learned structure or two learned structures those metrics allow the user to estimate the performance of the learning algorithm. In order to compute the metrics, a contingency table is computed of a pondered difference of the adjacency matrices od the two graphs.

The returns metrics are: TP = True Positive TN = True Negative FP = False Positive FN = False Negative CP = Condition Positive (ref) CN = Condition Negative (ref) PCP = Predicted Condition Positive (test) PCN = Predicted Condition Negative (test)

True Positive Rate

$$= \frac{\sum TP}{\sum CP}$$

False Positive Rate

$$= \frac{\sum FP}{\sum CN}$$

Accuracy

$$= \frac{\sum TP + \sum TN}{Totalpopulation}$$

G-measure

$$\sqrt{\frac{TP}{TP+FP} \cdot \frac{TP}{TP+FN}}$$

F1-Score

$$\frac{2\sum TP}{2\sum TP + \sum FN + \sum FP}$$

Positive Predictive Value

$$\frac{\sum TP}{\sum PCP}$$

False Ommision Rate

$$\frac{\sum FN}{\sum PCN}$$

Hamming-Distance: Number of changes needed to match the matrices.

The `ref` or `test` can be provided using a formula statement (similar to glm). A typical formula is `~ node1|parent1:parent2 + node2:node3|parent3`. The formula statement have to start with `~`. In this example, node1 has two parents (parent1 and parent2). node2 and node3 have the same parent3. The parents names have to exactly match those given in `node.names`. `:` is the separtor between either children or parents, `|` separates children (left side) and parents (right side), `+` separates terms, `.` replaces all the variables in `node.names`.

Value

A list matrix giving DAGs comparison metrics. The metrics are: True Positive Rate, False Positive Rate, Accuracy, G-measure, F1-Score, Positive Predictive Value, False Ommision Rate and the Hamming-Distance.

Author(s)

Gilles Kratzer

References

Sammut, Claude, and Geoffrey I. Webb. (2017). Encyclopedia of machine learning and data mining. Springer.

Further information about **abn** can be found at:
http://r-bayesian-networks.org

Examples

```
test.m <- matrix(data = 0, nrow = 3, ncol = 3)

ref.m <- matrix(data = c(0,0,0,
                         1,0,0,
                         1,0,0), nrow = 3, ncol = 3)

compareDag(ref = ref.m, test = test.m)
```

`createDag` *Create a letigimate DAGs*

Description

Create a legitimate DAGs in the abn format.

Usage

```
create_abnDag( dag, data.df = NULL, data.dists = NULL, ...)
```

Arguments

dag	a matrix or a formula specifying a DAG, see 'Details'.
data.df	named dataframe or named vector.
data.dists	named list giving the distribution for each node in the network. If not provided it will be sample and returned.
...	further arguments passed to or from other methods.

Details

An object of class class(abn) contains a named matrix describing the DAG and possibly additional objects such as the associated distributions of the nodes.

If the dag is specified with a formula, either data.df or data.dists is required with the . quantifier.

Value

An object of class abnDag containing a named matrix and a named list giving the distribution for each node.

Examples

```
create_abnDag( ~a+b|a, data.df=c("a"=1, "b"=1))
```

---

discretization *Discretization of a Possibly Continuous Data Frame of Random Variables based on their distribution*

Description

This function discretizes data frame of possibly continuous random variables through rules for discretization. The discretization algorithms are unsupervised and univariate. See details for the complete list (the number of state of each random variable could also be provided).

Usage

```
discretization(data.df = NULL,
               data.dists = NULL,
               discretization.method = "sturges",
               nb.states = FALSE)
```

Arguments

`data.df`	a data frame containing the data to discretize, binary variables must be declared as factors, other as numeric vector. The data frame must be named.
`data.dists`	a named list giving the distribution for each node in the network.
`discretization.method`	a character vector giving the discretization method to use; see details. If a number is provided, the variable will be discretized by equal binning.
`nb.states`	logical variable to select the output. If set to TRUE a list with the discretized data frame and the number of state of each variable is returned. If set to FALSE only the discretized data frame is returned.

Details

`discretization()` supports multiple rules for discretization. Below is the list of supported rules. IQR() stands for interquartile range.

`fd` stands for the Freedman Diaconis rule. The number of bins is given by

$$\frac{range(x) * n^{1/3}}{2 * IQR(x)}$$

The Freedman Diaconis rule is known to be less sensitive than the Scott's rule to outlier.

`doane` stands for doane's rule. The number of bins is given by

$$1 + \log_2 n + \log_2 1 + \frac{|g|}{\sigma_g}$$

is a modification of Sturges' formula which attempts to improve its performance with non-normal data.

`sqrt` The number of bins is given by:

$$\sqrt{(n)}$$

`cencov` stands for Cencov's rule. The number of bins is given by:

$$n^{1/3}$$

`rice` stands for Rice' rule. The number of bins is given by:

$$2n^{1/3}$$

`terrell-scott` stands for Terrell-Scott's rule. The number of bins is given by:

$$(2n)^{1/3}$$

This is known that Cencov, Rice and Terrell-Scott rules over-estimates k, compared to other rules due to his simplicity.

`sturges` stands for Sturges's rule. The number of bins is given by:

$$1 + \log_2(n)$$

`scott` stands for Scott's rule. The number of bins is given by:

$$range(x)/\sigma(x)n^{-1/3}$$

Value

The discretized dataframe or a list containing the table of counts for each bin the discretized data frame.

Author(s)

Gilles Kratzer

References

Garcia, S., et al. (2013). A survey of discretization techniques: Taxonomy and empirical analysis in supervised learning. *IEEE Transactions on Knowledge and Data Engineering*, 25.4, 734-750.

Cebeci, Z. and Yildiz, F. (2017). Unsupervised Discretization of Continuous Variables in a Chicken Egg Quality Traits Dataset. *Turkish Journal of Agriculture-Food Science and Technology*, 5.4, 315-320.

Examples

```
## Generate random variable
rv <- rnorm(n = 100, mean = 5, sd = 2)
dist <- list("gaussian")
names(dist) <- c("rv")

## Compute the entropy through discretization
entropyData(freqs.table = discretization(data.df = rv, data.dists = dist,
            discretization.method = "sturges", nb.states = FALSE))
```

`entropyData` *Computes an Empirical Estimation of the Entropy from a Table of Counts*

Description

This function empirically estimates the Shannon entropy from a table of counts using the observed frequencies.

Usage

```
entropyData(freqs.table)
```

Arguments

`freqs.table`	a table of counts.

Details

The general concept of entropy is defined for probability distributions. The 'entropyData' function estimates empirical entropy from data. The probability is estimated from data using frequency tables. Then the estimates are plug-in in the definition of the entropy to return the so-called empirical entropy. A common known problem of empirical entropy is that the estimations are biased due to the sampling noise. This is also known that the bias will decrease as the sample size increases.

Value

Shannon's entropy estimate on natural logarithm scale.

Author(s)

Gilles Kratzer

See Also

discretization

Examples

```
## Generate random variable
rv <- rnorm(n = 100, mean = 0, sd = 2)
dist <- list("gaussian")
names(dist) <- c("rv")

## Compute the entropy through discretization
entropyData(discretization(data.df = rv, data.dists = dist,
                        discretization.method = "fd", nb.states = FALSE))
```

---

essentialGraph *Plot an ABN graphic*

Description

Plot an ABN DAG using formula statement or a matrix in using Rgraphviz through the graphAM class

Usage

```
essentialGraph(dag, node.names = NULL, PDAG = "minimal")
```

Arguments

dag	a matrix or a formula statement (see 'Details' for format) defining the network structure, a directed acyclic graph (DAG).
node.names	a vector of names if the DAG is given via formula, see 'Details'.
PDAG	a character value that can be: minimal or complete, see 'Details'.

Details

This function returns an essential graph from a DAG. This can be useful if the learning procedure is defined up to a Markov class of equivalence. A minimal PDAG is defined as only directed edges are those who participate in v-structure. Whereas the completed PDAG: every directed edge corresponds to a compelled edge, and every undirected edge corresponds to a reversible edge.

The dag can be provided using a formula statement (similar to glm). A typical formula is `~ node1|parent1:parent2 + nod` The formula statement have to start with `~`. In this example, node1 has two parents (parent1 and parent2). node2 and node3 have the same parent3. The parents names have to exactly match those given in `node.names`. `:` is the separator between either children or parents, `|` separates children (left side) and parents (right side), `+` separates terms, `.` replaces all the variables in `node.names`.

Value

A matrix giving the PDAG.

Author(s)

Gilles Kratzer

References

West, D. B. (2001). Introduction to Graph Theory. Vol. 2. Upper Saddle River: Prentice Hall.

Further information about **abn** can be found at:
http://r-bayesian-networks.org

Examples

```
dag <- matrix(c(0,0,0, 1,0,0, 1,1,0), nrow = 3, ncol = 3)
dist <- list(a="gaussian", b="gaussian", c="gaussian")
colnames(dag) <- rownames(dag) <- names(dist)

essentialGraph(dag)
```

---

ex0.dag.data *Synthetic validation data set for use with abn library examples*

Description

300 observations simulated from a DAG with 10 binary variables, 10 Gaussian variables and 10 poisson variables.

Usage

```
ex0.dag.data
```

Format

A data frame, binary variables are factors. The relevant formulas are given below (note these do not give parameter estimates just the form of the relationships, e.g. logit()=1 means a logit link function and comprises of only an intercept term).

b1 binary, logit()=1

b2 binary, logit()=1

b3 binary, logit()=1

b4 binary, logit()=1

b5 binary, logit()=1

b6 binary, logit()=1

b7 binary, logit()=1

b8 binary, logit()=1

b9 binary, logit()=1

b10 binary, logit()=1

g1 gaussian, identity()=1

g2 gaussian, identity()=1

g3 gaussian, identity()=1

g4 gaussian, identity()=1

g5 gaussian, identity()=1

g6 gaussian, identity()=1

g7 gaussian, identity()=1

g8 gaussian, identity()=1

g9 gaussian, identity()=1

g10 gaussian, identity()=1

p1 poisson, log()=1

p2 poisson, log()=1

p3 poisson, log()=1

p4 poisson, log()=1

p5 poisson, log()=1

p6 poisson, log()=1

p7 poisson, log()=1

p8 poisson, log()=1

p9 poisson, log()=1

p10 poisson, log()=1

Examples

```
## Not run:
## The dataset was (essentially) generated using the following code:
datasize <- 300
tmp <- c(rep("y", as.integer(datasize/2)), rep("n", as.integer(datasize/2)))
set.seed(1)

ex0.dag.data <- data.frame(b1=sample(tmp, size=datasize, replace=TRUE),
                b2=sample(tmp, size=datasize, replace=TRUE),
                b3=sample(tmp, size=datasize, replace=TRUE),
                b4=sample(tmp, size=datasize, replace=TRUE),
                b5=sample(tmp, size=datasize, replace=TRUE),
                b6=sample(tmp, size=datasize, replace=TRUE),
                b7=sample(tmp, size=datasize, replace=TRUE),
                b8=sample(tmp, size=datasize, replace=TRUE),
                b9=sample(tmp, size=datasize, replace=TRUE),
                b10=sample(tmp, size=datasize, replace=TRUE),
                g1=rnorm(datasize, mean=0,sd=1),
                g2=rnorm(datasize, mean=0,sd=1),
                g3=rnorm(datasize, mean=0,sd=1),
                g4=rnorm(datasize, mean=0,sd=1),
                g5=rnorm(datasize, mean=0,sd=1),
                g6=rnorm(datasize, mean=0,sd=1),
                g7=rnorm(datasize, mean=0,sd=1),
                g8=rnorm(datasize, mean=0,sd=1),
                g9=rnorm(datasize, mean=0,sd=1),
                g10=rnorm(datasize, mean=0,sd=1),
                p1=rpois(datasize, lambda=10),
                p2=rpois(datasize, lambda=10),
                p3=rpois(datasize, lambda=10),
                p4=rpois(datasize, lambda=10),
                p5=rpois(datasize, lambda=10),
                p6=rpois(datasize, lambda=10),
                p7=rpois(datasize, lambda=10),
                p8=rpois(datasize, lambda=10),
                p9=rpois(datasize, lambda=10),
                p10=rpois(datasize, lambda=10))

## End(Not run)
```

`ex1.dag.data` *Synthetic validation data set for use with abn library examples*

Description

10000 observations simulated from a DAG with 10 variables from Poisson, Bernoulli and Gaussian distributions.

Usage

```
ex1.dag.data
```

Format

A data frame, binary variables are factors.The relevant formulas are given below (note these do not give parameter estimates just the form of the relationships, like in glm(), e.g. logit()=1+p1 means a logit link function and comprises of an intercept term and a term involving p1).

b1 binary, logit()=1

p1 poisson, log()=1

g1 gaussian, identity()=1

b2 binary, logit()=1

p2 poisson, log()=1+b1+p1

b3 binary, logit()=1+b1+g1+b2

g2 gaussian, identify()=1+p1+g1+b2

b4 binary, logit()=1+g1+p2

b5 binary, logit()=1+g1+g2

g3 gaussian, identity()=1+g1+b2

Examples

```
## The data is one realisation from the the underlying DAG:
ex1.true.dag <- matrix(data=c(
  0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,
  1,1,0,0,0,0,0,0,0,0,
  1,0,1,1,0,0,0,0,0,0,
  0,1,1,1,0,0,0,0,0,0,
  0,0,1,0,1,0,0,0,0,0,
  0,0,1,0,0,0,1,0,0,0,
  0,0,1,1,0,0,0,0,0,0), ncol=10, byrow=TRUE)

colnames(ex1.true.dag) <- rownames(ex1.true.dag) <-
    c("b1","p1","g1","b2","p2","b3","g2","b4","b5","g3")
```

`ex2.dag.data` *Synthetic validation data set for use with abn library examples*

Description

10000 observations simulated from a DAG with 18 variables three sets each from Poisson, Bernoulli and Gaussian distributions.

Usage

```
ex2.dag.data
```

Format

A data frame, binary variables are factors. The relevant formulas are given below (note these do not give parameter estimates just the form of the relationships, e.g. logit()=1 means a logit link function and comprises of only an intercept term).

b1 binary,logit()=1+g1+b2+b3+p3+b4+g4+b5

g1 gaussian,identity()=1

p1 poisson,log()=1+g6

b2 binary,logit()=1+p3+b4+p6

g2 gaussian,identify()=1+b2

p2 poisson,log()=1+b2

b3 binary,logit()=1+g1+g2+p2+g3+p3+g4

g3 gaussian,identify()=1+g1+p3+b4

p3 poisson,log()=1

b4 binary,logit()=1+g1+p3+p5

g4 gaussian,identify()=1+b4;

p4 poisson,log()=1+g1+b2+g2+b5

b5 binary,logit()=1+b2+g2+b3+p3+g4

g5 gaussian,identify()=1

p5 poisson,log()=1+g1+g5+b6+g6

b6 binary,logit()=1

g6 gaussian,identify()=1

p6 poisson,log()=1+g5

Examples

```
## The true underlying stochastic model has DAG - this data is a single realisation.
ex2.true.dag <- matrix(data = c(
   0,1,0,1,0,0,1,0,1,1,1,0,1,0,0,0,0,0,
   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,
   0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,1,
   0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
   0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
   0,1,0,0,1,1,0,1,1,0,1,0,0,0,0,0,0,0,
   0,1,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,
   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
   0,1,0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,
   0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,
   0,1,0,1,1,0,0,0,0,0,0,0,1,0,0,0,0,0,
   0,0,0,1,1,0,1,0,1,0,1,0,0,0,0,0,0,0,
   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
   0,1,0,0,0,0,0,0,0,0,0,0,1,0,1,1,0,
   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
```

```
   0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0
   ), ncol = 18, byrow = TRUE)

colnames(ex2.true.dag) <- rownames(ex2.true.dag) <- c("b1","g1","p1","b2",
  "g2","p2","b3","g3","p3","b4","g4","p4","b5","g5","p5","b6","g6","p6")
```

`ex3.dag.data` *Validation data set for use with abn library examples*

Description

1000 observations across with 13 binary variables and one grouping variable. Real (anonymised) data of unknown structure.

Usage

```
ex3.dag.data
```

Format

A data frame with 14 columns, where `b1,b2,...,b13` are binary variables encoded as factors and `group` is a factor with 100 factors defining the sampling groups (10 observations each).

`ex4.dag.data` *Valdiation data set for use with abn library examples*

Description

2000 observations across with 10 binary variables and one grouping variable. Real (anonymised) data of unknown structure.

Usage

```
ex4.dag.data
```

Format

A data frame with eleven columns: `group` factor with 85 levels defining sampling groups; `b1,...,b10` binary variables encoded as factors.

ex5.dag.data *Valdiation data set for use with abn library examples*

Description

434 observations across with 18 variables, 6 binary and 12 continuous, and one grouping variable. Real (anonymised) data of unknown structure.

Usage

```
ex5.dag.data
```

Format

A data frame with 19 columns: `b1,...,b6` binary variables, encoded as factors; `g1,...,g12` continuous variables. Finally, the column `group` defines sampling groups (encoded as a factor as well).

ex6.dag.data *Valdiation data set for use with abn library examples*

Description

800 observations across with 8 variables, 1 count, 2 binary and 4 continuous, and 1 grouping variable. Real (anonymised) data of unknown structure.

Usage

```
ex6.dag.data
```

Format

A data frame with eight columns. Binary variables are factors

p1 count

g1 continuous

g2 continuous

b1 binary

b2 binary

g3 continuous

g4 continuous

group factor,defines sampling groups

ex7.dag.data *Valdiation data set for use with abn library examples*

Description

10648 observations across with 3 variables, 2 binary and 1 grouping variable. Real (anonymised) data of unknown structure.

Usage

```
ex7.dag.data
```

Format

A data frame, binary variables are factors

b1 binary

b2 binary

group factor, defines sampling groups

expit *Expit, Logit and odds*

Description

Compute the expit and logit of a numerical vector. Transform odds to probability.

Usage

```
expit(x)
logit(x)
odds(x)
```

Arguments

x	vector of real values.

Details

`logit` computes the logit function:

$$logit(p) = \log \frac{p}{1-p}$$

`expit` computes the expit function:

$$expit(x) = \frac{e^x}{1+e^x}$$

`odds` transform an odd into a probability.

$$odds(x) = \frac{x}{1-x}$$

Those functions become numerically unstable if evaluated at the edge or the definition range.

Value

A real vector corresponding to the expit, the logit or the odds of the input values.

Author(s)

Gilles Kratzer

`fitabn` *Fit an additive Bayesian network model*

Description

Fits an additive Bayesian network to observed data and is equivalent to Bayesian or information theoretic multi-dimensional regression modelling. Two numerical options are available in the Bayesian settings, standard Laplace approximation or else an integrated nested Laplace approximation provided via a call to the R INLA library (see www.r-inla.org - this is not hosted on CRAN).

Usage

```
fitabn(object = NULL, dag.m = NULL, data.df = NULL,
       data.dists = NULL, method = "bayes", group.var = NULL,
       adj.vars = NULL, cor.vars = NULL, create.graph = FALSE,
       compute.fixed = FALSE, control = list(), verbose = FALSE,
       centre = TRUE, ...)
```

Arguments

object	an object of class 'abnLearned' produced by mostprobable, search.heuristic or search.hillclimber.
dag.m	a matrix or a formula statement (see details) defining the network structure, a directed acyclic graph (DAG), see details for format. Note that colnames and rownames must be set.
data.df	a data frame containing the data used for learning the network, binary variables must be declared as factors and no missing values all allowed in any variable.
data.dists	a named list giving the distribution for each node in the network, see details.
method	should a "Bayes" or "mle" approach be used, see details.
group.var	only applicable for mixed models and gives the column name in data.df of the grouping variable (which must be a factor denoting group membership).
adj.vars	a character vector giving the column names in data.df for which the network score has to be adjusted for, see details.
cor.vars	a character vector giving the column names in data.df for which a mixed model should be used.
create.graph	create an R graph class object which enables easy plotting of dag.m using plot(), requires Rgraphviz (hosted on bioconductor rather than CRAN).
compute.fixed	a logical flag, set to TRUE for computation of marginal posterior distributions, see details.
control	a list of control parameters. See details.
verbose	if TRUE then provides some additional output, in particular the code used to call INLA, if applicable.
centre	should the observations in each Gaussian node first be standardised to mean zero and standard deviation one.
...	additional arguments passed for optimization.

Details

If method="Bayes":

The procedure fitabn fits an additive Bayesian network model to data where each node (variable - a column in data.df) can be either: presence/absence (Bernoulli); continuous (Gaussian); or an unbounded count (poisson). The model comprises of a set of conditionally independent generalized linear regressions with or without random effects. Internal code is used by default for numerical estimation in nodes without random effects, and INLA is the default for nodes with random effects. This default behaviour can be over-ridden using max.mode.error. The default is max.mode.error=10 which means that the modes estimated from INLA output must be within 10% of those estimated using internal code, otherwise internal code is used rather than INLA. To force the use of INLA on all nodes use max.mode.error=100 which then ignores this check, to force the use of internal code then use max.mode.error=0. For the numerical reliability and perform of abn see http://r-bayesian-networks.org. Generally speaking INLA can be extremely fast and accurate, but in a number of cases it can perform very poorly and so some care is required (which is why there is an internal check on the modes). Binary variables must be declared as factors with two levels, and the argument data.dists must be a list with named arguments, one for each of the

variables in data.df (except a grouping variable - if present), where each entry is either "poisson","binomial", or "gaussian", see examples below. The "poisson" and "binomial" distributions use log and logit link functions respectively. Note that "binomial" here actually means only binary, one bernoulli trial per row in data.df.

If the data are grouped into correlated blocks - where in a standard regression context a mixed model might be used - then a network comprising of one or more nodes where a generalized linear mixed model is used (but limited to only a single random effect). This is achieved by specifying parameters group.var and cor.vars, where the former defines the group membership variable which should be a factor indicating which observations belong to the same grouping. The parameter cor.vars is a character vector which contains the names of the nodes for which a mixed model should be used. For example, in some problems it may be appropriate for all variables (except group.var) in data.df to be parametrised as a mixed model while in others it may only be a single variable for which grouping adjustment is required (as the remainder of variables are covariates measured at group level).

In the network structure definition, dag.m, each row represents a node in the network, and the columns in each row define the parents for that particular node, see the example below for the specific format. The dag.m can be provided using a formula statement (similar to glm). A typical formula is `~ node1|parent1:parent2 + node2:node3|parent3`. The formula statement have to start with ~. In this example, node1 has two parents (parent1 and parent2). node2 and node3 have the same parent3. The parents names have to exactly match those given in data.df. : is the separator between either children or parents, | separates children (left side) and parents (right side), + separates terms, . replaces all the variables in data.df.

If compute.fixed=TRUE then marginal posterior distributions for all parameters are computed. Note the current algorithm used to determine the evaluation grid is rather crude and may need to be manually refined using variate.vec (one parameter at a time) for publication quality density estimates. Note that a manual grid can only be used with internal code and not INLA (which uses its own grid). The end points are defined as where the value of the marginal density drops below a given threshold pdf.min.

If create.graph=TRUE then the model definition matrix in dag.m is used to create an R graph object (of type graphAM-class). See ?"graph-class" for details and the Rgraphviz documentation (which is extensive). The main purpose of this is simply to allow easy visualisation of the DAG structure via the graphviz library. A graph plot can easily be created by calling the method plot() on this object (see example below). Note, however, that the fonts and choice of scaling used here may be far less visually optimal than using graphviz direct (e.g via tographviz) for publication quality output. Also, re-scaling the plotting window may not result in a callback to re-optimise the visual position of the nodes and edges, and so if the window is re-sized then re-run the plot command to re-draw to the new scale.

When estimating the log marginal likelihood in models with random effects (using internal code rather than INLA) an attempt is made is minimize the error by comparing the estimates given between a 3pt and 5pt rule when estimating the Hessian in the Laplace approximation. The modes used in each case are identical. The first derivatives are computed using gsl's adaptive finite difference function and this is embedding inside the standard 3pt and 5pt rules for the second derivatives. In all cases a central difference approximation is tried first with a forward difference being a fall back (as the precision parameters are strictly positive). The error is minimised through choosing an optimal step size using gsl's Nelder-Mead optimization, and if this fails, (e.g. is greater than max.hessian.error) then the Brent-Dekker root bracketing method is used as a fall back. If the error cannot be reduced to below max.hessian.error then the step size which gave the lowest

error during the searches (across potentially many different initial bracket choices) is used for the final Hessian evaluations in the Laplace approximation.

If `method="mle"`:

The procedure `fitabn` with the argument `method= "mle"` fits an additive Bayesian network model to data where each node (variable - a column in data.df) can be either: presence/absence (Bernoulli); continuous (Gaussian); an unbounded count (Poisson); or a discrete variable (Multinomial). The model comprises of a set of conditionally independent generalized linear regressions with or without adjustment.

Binary and discrete variables must be declared as factors, and the argument data.dists must be a list with named arguments, one for each of the variables in `data.df`, where each entry is either "poisson","binomial", "multinomial" or "gaussian", see examples below. The "poisson" and "binomial" distributions use log and logit link functions respectively. Note that "binomial" here actually means only binary, one Bernoulli trial per row in data.df.

In the context of `fitabn` adjustment means that irrespective to the adjacency matrix the adjustment variable set (`adj.vars`) will be add as covariate to every node defined by `cor.vars`. If `cor.vars` is NULL then adjustment is over all variables in the `data.df`.

In the network structure definition, `dag.m`, each row represents a node in the network, and the columns in each row define the parents for that particular node, see the example below for the specific format. The `dag.m` can be provided using a formula statement (similar to glm). A typical formula is `~ node1|parent1:parent2 + node2:node3|parent3`. The formula statement have to start with `~`. In this example, node1 has two parents (parent1 and parent2). node2 and node3 have the same parent3. The parents names have to exactly match those given in `data.df`. `:` is the separator between either children or parents, `|` separates children (left side) and parents (right side), `+` separates terms, `.` replaces all the variables in `data.df`.

The Information-theoretic based network scores used in `fitabn` with argument `method="mle"` are the maximum likelihood (mlik, called marginal likelihood in this context as it is computed node wise), the Akaike Information Criteria (aic), the Bayesian Information Criteria (bic) and the Minimum distance Length (mdl). The classical definitions of those metrics are given in Kratzer and Furrer (2018).

The numerical routine are based on an iterative scheme to estimate the regression coefficients. The Iterative Reweighed Least Square (IRLS) programmed using Rcpp/RcppArmadrillo. One hard coded feature of `fitabn` with argument `method="mle"` is a conditional use of a bias reduced binomial regression when a classical glm fails to accurately estimates the maximum likelihood of the given model. Additionally a QR decomposition is performed to check for rank deficiency. If the model is rank deficient and the BR glm fails to estimate it, then predictors are sequentially removed. This feature aims at better estimating network scores when data sparsity is present.

A special care should be taken when intrepreting or even displaing p-values computed with `fitabn`. Indeed, the full model is already selected using goodness of fit metrics based on the (same) full dataset.

The control argument is a list that can supply any of the following components:

mean the prior mean for all the Gaussian additive terms for each node.

prec the prior precision for all the Gaussian additive terms for each node.

loggam.shape the shape parameter in the Gamma distributed prior for the precision in any Gaussian nodes, also used for group level precision is applicable.

loggam.inv.scale the inverse scale parameter in the Gamma distributed prior for the precision in any Gaussian nodes, also used for group level precision is applicable.

max.mode.error if the estimated modes from INLA differ by a factor of max.mode.error or more from those computed internally then results from INLA are replaced by those computed internally. To force INLA to always be used then max.mode.error=100, to force INLA to never be used max.mod.error=0. See details.

max.iters total number of iterations allowed when estimating the modes in Laplace approximation

epsabs absolute error when estimating the modes in Laplace approximation for models with no random effects.

error.verbose logical,additional output in the case of errors occurring in the optimisation

epsabs.inner absolute error in the maximisation step in the (nested) Laplace approximation for each random effect term

max.iters.inner total number of iterations in the maximisation step in the nested Laplace approximation

finite.step.size suggested step length used in finite difference estimation of the derivatives for the (outer) Laplace approximation when estimating modes

hessian.params a numeric vector giving parameters for the adaptive algorithm which determines the optimal step size in the finite difference estimation of the Hessian. First entry is the initial guess, second entry absolute error

max.iters.hessian integer, maximum number of iterations to use when determining an optimal finite difference approximation (nelder-mead)

max.hessian.error if the estimated log marginal likelihood when using an adaptive 5pt finite difference rule for the Hessian differs by more than max.hessian.error from when using an adaptive 3pt rule then continue to minimize the local error by switching to the Brent-Dekker root bracketing method, see details

factor.brent if using Brent-Dekker root bracketing method then define the outer most interval end points as the best estimate of h (stepsize) from the nelder-mead as (h/factor.brent,h*factor.brent)

maxiters.hessian.brent maximum number of iterations allowed in the Brent-Dekker method

num.intervals.brent the number of initial different bracket segments to try in the Brent-Dekker method

min.pdf the value of the posterior density function below which we stop the estimation, only used when computing marginals, see details.

n.grid recompute density on an equally spaced grid with `n.grid` points.

std.area logical, should the area under the estimated posterior density be standardised to exactly one, useful for error checking.

marginal.quantiles vector giving quantiles at which to compute the posterior marginal distribution at.

max.grid.iter gives number of grid points to estimate posterior density at when not explicitly specifying a grid, used to avoid excessively long computation.

marginal.node used in conjunction with `marginal.param` to allow bespoke estimate of a marginal density over a specific grid. value from 1 to number of nodes.

marginal.param used in conjunction with `marginal.node`. value of 1 is for intercept, see modes entry in results for appropriate number.

variate.vec a vector containing the places to evaluate the posterior marginal density, must be supplied if `marginal.node` is not null

maxit integer given the maximum number of run for estimating network scores using an Iterative Reweighed Least Square algorithm.

tol real number giving the minimal tolerance expected to terminate the Iterative Reweighed Least Square algorithm to estimate network score.

Value

An object of class `abnFit`. A named list. One entry for each of the variables in `data.df` (excluding the grouping variable, if present) which contains an estimate of the log marginal likelihood for that individual node. An entry "mlik" which is the total log marginal likelihood for the full ABN model. A vector of error.codes - non-zero if a numerical error or warning occurred, and a vector error.code.desc giving a text description of the error. A list `modes`, which contains all the mode estimates for each parameter at each node. A vector called Hessian accuracy which is the estimated local error in the log marginal likelihood for each node. If `compute.fixed=TRUE` then a list entry called `marginals` which contains a named entry for every parameter in the ABN and each entry in this list is a two column matrix where the first column is the value of the marginal parameter, say x, and the second column is the respective density value, pdf(x). In addition a list called `marginal.quantiles` is produced giving the quantiles for each marginal posterior distribution. If `create.graph=TRUE` then an additional entry `graph` which is of type class `graphAM`-class (**Rgraphviz** is created.

Author(s)

Fraser Iain Lewis and Gilles Kratzer

References

Lewis, F. I., and McCormick, B. J. J. (2012). Revealing the complexity of health determinants in resource poor settings. *American Journal Of Epidemiology*. DOI:10.1093/aje/KWS183.

Kratzer, G., and Furrer, R., 2018. Information-Theoretic Scoring Rules to Learn Additive Bayesian Network Applied to Epidemiology. Preprint; Arxiv: stat.ML/1808.01126.

Further information about **abn** can be found at:
http://r-bayesian-networks.org

See Also

buildscorecache

Examples

```
## Not run:
## Built-in dataset with a subset of cols
mydat <- ex0.dag.data[,c("b1","b2","b3","g1","b4","p2","p4")]


## setup distribution list for each node
mydists <- list(b1="binomial", b2="binomial", b3="binomial", g1="gaussian",
```

```
              b4="binomial", p2="poisson", p4="poisson")

## Null model - all independent variables
mydag.empty <- matrix(0, nrow=7, ncol=7)
colnames(mydag.empty) <- rownames(mydag.empty) <- names(mydat)

## Now fit the model to calculate its goodness-of-fit
myres <- fitabn(dag.m = mydag.empty, data.df = mydat, data.dists = mydists)

## Log-marginal likelihood goodness-of-fit for complete DAG
print(myres$mlik)

## Now repeat but include some dependencies first
mydag <- mydag.empty
mydag["b1","b2"] <- 1 # b1<-b2
mydag["b2","p4"] <- 1 # b2<-p4
mydag["b2","g1"] <- 1 # b2<-g1
mydag["g1","p2"] <- 1 # g1<-p2
mydag["b3","g1"] <- 1 # b3<-g1
mydag["b4","b1"] <- 1 # b4<-b1
mydag["p4","g1"] <- 1 # p4<-g1

myres <- fitabn(dag.m = mydag, data.df = mydat, data.dists = mydists)

## Or equivalentelly using the formula statement
myres <- fitabn(dag.m = ~b1|b2+b2|p4+g1+g1|p2+b3|g1+b4|b1+p4|g1,
                data.df = mydat, data.dists = mydists)

print(myres$mlik) ## a much poorer fit than full independence DAG

## Now also create the graph of the DAG via Rgraphviz
myres <- fitabn(dag.m = mydag,data.df = mydat, data.dists = mydists,
                  create.graph = TRUE)
plotabn(dag.m = mydag, data.dists = mydists, fitted.values.abn = myres$modes)

## A simple plot of some posterior densities the algorithm which chooses
## density points is very simple any may be rather sparse so also recompute
## the density over an equally spaced grid of 100 points between the two
## end points which had at f=min.pdf
## max.mode.error = 0 foces to use the internal c code
myres.c <- fitabn(dag.m = mydag, data.df = mydat, data.dists = mydists,
                  compute.fixed = TRUE, n.grid = 100, max.mode.error = 0)

print(names(myres.c$marginals)) ## gives all the different parameter names

## Repeat but use INLA for the numerics using max.mode.error=100
## as using internal code is the default here rather than INLA
myres.inla <- fitabn(dag.m = mydag, data.df = mydat, data.dists = mydists,
                     compute.fixed = TRUE, n.grid = 100, max.mode.error = 100)

## Plot posterior densities
par(mfrow=c(2,2), mai=c(.7,.7,.2,.1))
plot(myres.c$marginals$b1[["b1|(Intercept)"]], type="l", xlab="b1|(Intercept)")
```

```
lines(myres.inla$marginals$b1[["b1|(Intercept)"]], col="blue")
plot(myres.c$marginals$b2[["b2|p4"]], type="l", xlab="b2|p4")
lines(myres.inla$marginals$b2[["b2|p4"]], col="blue")
plot(myres.c$marginals$g1[["g1|precision"]], type="l", xlab="g1|precision")
lines(myres.inla$marginals$g1[["g1|precision"]], col="blue")
plot(myres.c$marginals$b4[["b4|b1"]], type="l", xlab="b4|b1")
lines(myres.inla$marginals$b4[["b4|b1"]], col="blue")

## A very simple mixed model example using built-in data specify DAG,
## only two variables using subset of variables from ex3.dag.data
## both variables are assumed to need (separate) adjustment for the
## group variable, i.e., a binomial glmm at each node


mydists <- list(b1="binomial",  b2="binomial")

## Compute marginal likelihood - use internal code via max.mode.error=0
## as using INLA is the default here.
## Model where b1 <- b2
myres.c <- fitabn(dag.m = ~b1|b2, data.df = ex3.dag.data[,c(1,2,14)], data.dists = mydists,
                  group.var = "group", cor.vars = c("b1","b2"),
                  max.mode.error=0)
print(myres.c) ## show all the output

## compare modes for node b1 with glmer()
require(MASS)
require(lme4)

m1 <- glmer(b1 ~ 1 + b2 + (1|group),
            family = "binomial", data = ex3.dag.data[,c(1,2,14)])

print(myres.c$modes$b1) ## almost identical to lme4 n.b. glmer() gives variance
##                          fitabn gives precision=1/variance

## Compare with INLA estimate
myres.inla <- fitabn(dag.m = ~b1|b2,data.df=ex3.dag.data[,c(1,2,14)],
                     data.dists = mydists, group.var = "group", cor.vars = c("b1","b2"),
                     compute.fixed = FALSE, max.mode.error = 100)

## Compare log marginal likelihoods for each node and total DAG:
cbind(INLA = unlist(myres.inla[1:3]), C = unlist(myres.c[1:3]),
             Delta = unlist(myres.inla[1:3]) - unlist(myres.c[1:3]))

## Now for marginals - INLA is strongly preferable for estimating marginals for nodes
## with random effects as it is far faster, but may not be reliable
## see http://r-bayesian-networks.org

# INLA's estimates of the marginals, using high n.grid=500
# as this makes the plots smoother - see below.
myres.inla <- fitabn(dag.m = ~b1|b2, data.df = ex3.dag.data[,c(1,2,14)],
                     data.dists = mydists,
                     group.var = "group", cor.vars = c("b1", "b2"),
                     compute.fixed = TRUE, n.grid = 500,
```

```
                    control = list(max.mode.error = 100, max.hessian.error = 10E-02))

## this is NOT recommended - marginal density estimation using fitabn in mixed models
## is really just for diagnostic purposes, better to use fitabn.inla() here
## but here goes...be patient
myres.c <- fitabn(dag.m = ~b1|b2, data.df = ex3.dag.data[,c(1,2,14)], data.dists = mydists,
                  group.var = "group", cor.vars=c("b1", "b2"), compute.fixed = TRUE,
                  control = list(max.mode.error = 0, max.hessian.error = 10E-02))

## compare marginals between internal and INLA.
par(mfrow=c(2,3))
## 5 parameters - two intercepts, one slope, two group level precisions
plot(myres.inla$marginals$b1[[1]], type="l", col="blue")
lines(myres.c$marginals$b1[[1]], col="brown", lwd=2)
plot(myres.inla$marginals$b1[[2]], type="l", col="blue")
lines(myres.c$marginals$b1[[2]], col="brown", lwd=2)
## the precision of group-level random effects
plot(myres.inla$marginals$b1[[3]],type="l", col="blue", xlim=c(0,2))
lines(myres.c$marginals$b1[[3]],col="brown",lwd=2)
plot(myres.inla$marginals$b2[[1]],type="l", col="blue")
lines(myres.c$marginals$b2[[1]],col="brown",lwd=2)
plot(myres.inla$marginals$b2[[1]], type="l", col="blue")
lines(myres.c$marginals$b2[[1]], col="brown", lwd=2)
## the precision of group-level random effects
plot(myres.inla$marginals$b2[[2]], type="l", col="blue", xlim=c(0,2))
lines(myres.c$marginals$b2[[2]], col="brown", lwd=2)

### these are very similar although not exactly identical

## use internal code but only to compute a single parameter over a specified grid
## this can be necessary if the simple auto grid finding functions does a poor job

#myres.c <- fitabn(dag.m = ~b1|b2, data.df = ex3.dag.data[,c(1,2,14)], data.dists = mydists,
#                  group.var = "group", cor.vars = c("b1", "b2"),
#                  centre = FALSE, compute.fixed = TRUE,
#                  marginal.node = 1, marginal.param = 3,## precision term in node 1
#                  variate.vec = seq(0.05, 1.5, len=25), max.hessian.error = 10E-02)

#par(mfrow=c(1,2))
#plot(myres.c$marginals[[1]], type="l", col="blue")## still fairly sparse
## An easy way is to use spline to fill in the density without recomputing other
## points provided the original grid is not too sparse.
#plot(spline(myres.c$marginals[[1]], n=100), type="b", col="brown")

## ------------------------------------------------------------------------------------
## This function contains an MLE implementation accessible through a method parameter
## use built-in simulated data set
## ------------------------------------------------------------------------------------

mydat <- ex0.dag.data[,c("b1","b2","b3","g1","b4","p2","p4")] ## take a subset of cols

## setup distribution list for each node
mydists <- list(b1="binomial", b2="binomial", b3="binomial",
```

```
                g1="gaussian", b4="binomial", p2="poisson", p4="poisson")


## now fit the model to calculate its goodness of fit
myres.mle <- fitabn(dag.m = ~b1|b2+b2|p4+g1+g1|p2+b3|g1+b4|b1+p4|g1,
                    data.df = mydat, data.dists = mydists, method = "mle")

myres.bayes <- fitabn(dag.m = ~b1|b2+b2|p4+g1+g1|p2+b3|g1+b4|b1+p4|g1,
                      data.df = mydat, data.dists = mydists, method = "bayes")

## print the output
## MLE
print(myres.mle)

#Bayes
print(myres.bayes)

## plot the model with parameter estimates
plotabn(dag.m = mydag, data.dists = mydists, fitted.values.abn.mle = myres.bayes$modes)

## End(Not run)
```

`infoDag` *Compute standard information for a DAG.*

Description

This function returns standard metrics for DAG description. A list that contains the number of nodes, the number of arcs, the average Markov blanket size, the neighbourhood average set size, the parent average set size and children average set size.

Usage

```
infoDag(dag, node.names = NULL)
```

Arguments

`dag`	a matrix or a formula statement (see details for format) defining the network structure, a directed acyclic graph (DAG). Note that row names must be set or given in `node.names`.
`node.names`	a vector of names if the DAG is given via formula, see details.

Details

This function returns a named list with the following entries: the number of nodes, the number of arcs, the average Markov blanket size, the neighbourhood average set size, the parent average set size and children average set size.

The dag can be provided using a formula statement (similar to glm). A typical formula is `~ node1|parent1:parent2 + nod`
The formula statement have to start with ~. In this example, node1 has two parents (parent1 and parent2). node2 and node3 have the same parent3. The parents names have to exactly match those given in `node.names`. `:` is the separator between either children or parents, `|` separates children (left side) and parents (right side), `+` separates terms, `.` replaces all the variables in `node.names`.

Value

A named list that contains following entries: the number of nodes, the number of arcs, the average Markov blanket size, the neighbourhood average set size, the parent average set size and children average set size.

Author(s)

Gilles Kratzer

References

West, D. B. (2001). Introduction to graph theory. Vol. 2. Upper Saddle River: Prentice Hall.

Further information about **abn** can be found at:
http://r-bayesian-networks.org

Examples

```
## Creating a dag:
dag <- matrix(c(0,0,0,0, 1,0,0,0, 1,1,0,1, 0,1,0,0), nrow = 4, ncol = 4)
dist <- list(a="gaussian", b="gaussian", c="gaussian", d="gaussian")
colnames(dag) <- rownames(dag) <- names(dist)

infoDag(dag)

## Not run:
plot(create_abnDag(dag))

## End(Not run)
```

`linkStrength` *A function that returns the strengths of the edge connections in a Bayesian Network learned form observational data.*

Description

A flexible implementation of multiple proxy for strength measures usefull for visualizing the edge connections in a Bayesian Network learned form observational data.

Usage

```
linkStrength(dag, data.df = NULL, data.dists = NULL,
           method = c("mi.raw", "mi.raw.pc", "mi.corr", "ls", "ls.pc", "stat.dist"),
              discretization.method = "doane")
```

Arguments

`dag`	a matrix or a formula statement (see details for format) defining the network structure, a directed acyclic graph (DAG). Note that rownames must be set or given in `data.dist` if the DAG is given via a formula statement.
`data.df`	a data frame containing the data used for learning each node, binary variables must be declared as factors.
`data.dists`	a named list giving the distribution for each node in the network, see 'Details'.
`method`	the method to be used. See 'Details'.
`discretization.method`	a character vector giving the discretization method to use. See `discretization`.

Details

This function returns multiple proxies for estimating the connection strength of the edges of a possibly discretized Bayesian network's dataset. The retuned connection strength measures are: the Raw Mutual Information (`mi.raw`), the Percentage Mutual information (`mi.raw.pc`), the Raw Mutual Information computed via correlation (`mi.corr`), the link strength (`ls`), the percentage link strength (`ls.pc`) and the statistical distance (`stat.dist`).

The general concept of entropy is defined for probability distributions. The probability is estimated from data using frequency tables. Then the estimates are plug-in in the definition of the entropy to return the so-called empirical entropy. A common known problem of empirical entropy is that the estimations are biased due to the sampling noise. This is also known that the bias will decrease as the sample size increases. The mutual information estimation is computed from the observed frequencies through a plugin estimator based on entropy. For the case of an arc going from the node X to the node Y and the remaining set of parent of Y is denoted as Z.

The mutual information is defined as I(X, Y) = H(X) + H(Y) - H(X, Y), where H() is the entropy.

The Percentage Mutual information is defined as PI(X,Y) = I(X,Y)/H(Y|Z).

The Mutual Information computed via correlation is defined as MI(X,Y) = -0.5 log(1-cor(X,Y)^2).

The link strength is defined as LS(X->Y) = U(Y|Z)-U(Y|X,Z).

The percentage link strength is defined as PLS(X->Y) = LS(X->Y) / H(Y|Z).

The statistical distance is defined as SD(X,Y) = 1- MI(X,Y) / max(H(X),H(Y)).

Value

The function returns a DAG wise matrix with the requested metric.

Author(s)

Gilles Kratzer

References

Boerlage, B. (1992). Link strength in Bayesian networks. Diss. University of British Columbia.

Ebert-Uphoff, Imme. "Tutorial on how to measure link strengths in discrete Bayesian networks." (2009).

Further information about **abn** can be found at:
http://r-bayesian-networks.org

Examples

```
dist <- list(a="gaussian", b="gaussian", c="gaussian")
data.param <- matrix(c(0,1,0, 0,0,1, 0,0,0), nrow = 3L, ncol = 3L, byrow = TRUE)

data.param.var <- matrix(0, nrow = 3L, ncol = 3L)
diag(data.param.var) <- c(0.1,0.1,0.1)

out <- simulateAbn(data.dists = dist,
    n.chains = 1, n.adapt = 1000, n.thin = 1, n.iter = 100,
    data.param = data.param, data.param.var = data.param.var)

linkStrength(data.param, data.df = out, data.dists = dist,
             method = "ls", discretization.method = "sturges")
```

---

mb *Compute the Markov blanket*

Description

This function computes the Markov blanket of a set of node given a DAG (Directed Acyclic Graph).

Usage

```
mb(dag, node, data.dists=NULL)
```

Arguments

dag	a matrix or a formula statement (see details for format) defining the network structure, a directed acyclic graph (DAG).
node	a character vector of the nodes for which the Markov Blanket should be returned.
data.dists	a named list giving the distribution for each node in the network, see details.

Details

This function return the Markov Blanket of a set of nodes given a DAG.

The dag can be provided using a formula statement (similar to glm). A typical formula is `~ node1|parent1:parent2 + nod`
The formula statement have to start with `~`. In this example, node1 has two parents (parent1 and parent2). node2 and node3 have the same parent3. The parents names have to exactly match those given in `name`. `:` is the separtor between either children or parents, `|` separates children (left side) and parents (right side), `+` separates terms, `.` replaces all the variables in `name`.

Examples

```
## Defining distribution and dag
dist <- list(a="gaussian", b="gaussian", c="gaussian", d="gaussian", e="binomial",
              f="binomial")
dag <- matrix(c(0,1,1,0,1,0,
    0,0,1,1,0,1,
    0,0,0,0,0,0,
    0,0,0,0,0,0,
    0,0,0,0,0,1,
    0,0,0,0,0,0), nrow = 6L, ncol = 6L, byrow = TRUE)

colnames(dag) <- rownames(dag) <- names(dist)

mb(dag, node = "b", data.dists = dist)
mb(dag, node = "e", data.dists = dist)
mb(dag, node = c("b","e"), data.dists = dist)
```

`miData` *Empirical Estimation of the Entropy from a Table of Counts*

Description

This function empirically estimates the Mutual Information from a table of counts using the observed frequencies.

Usage

```
miData(freqs.table, method = c("mi.raw", "mi.raw.pc"))
```

Arguments

`freqs.table`	a table of counts.
`method`	a character determining if the Mutual Information should be normalized.

Details

The mutual information estimation is computed from the observed frequencies through a plugin estimator based on entropy.

The plugin estimator is I(X, Y) = H (X) + H(Y) - H(X, Y), where H() is the entropy computed with `entropyData`.

Value

Mutual information estimate.

Author(s)

Gilles Kratzer

See Also

discretization

Examples

```
## Generate random variable
Y <- rnorm(n = 100, mean = 0, sd = 2)
X <- rnorm(n = 100, mean = 5, sd = 2)

dist <- list(Y="gaussian", X="gaussian")

miData(discretization(data.df = cbind(X,Y), data.dists = dist,
                discretization.method = "fd", nb.states = FALSE),
       method = "mi.raw")
```

---

mostprobable *Find most probable DAG structure*

Description

Find most probable DAG structure using exact order based approach due to Koivisto and Sood, 2004

Usage

```
mostprobable(score.cache, prior.choice=1, score="mlik", verbose=TRUE)
```

Arguments

score.cache	object of class abnCache typically outputted by from buildscorecache().
prior.choice	an integer, 1 or 2, where 1 is a uniform structural prior and 2 uses a weighted prior, see details
score	character giving which network score should be used to select the structure.
verbose	if TRUE then provides some additional output.

Details

The procedure runs the exact order based structure discovery approach of Koivisto and Sood (2004) to find the most probable posterior network (DAG). The local.score is the node cache, as created using buildscorecache (or manually provided the same format is used). Note that the scope of this search is given by the options used in local.score, for example by restricting the number of parents or the ban or retain constraints given there.

This routine can take a long time to complete and is highly sensitive to the number of nodes in the network. It is recommended to use this on a reduced data set to get an idea as to the computational practicality of this approach. In particular, memory usage can quickly increase to beyond what may be available. For additive models, problems comprising up to 20 nodes are feasible on most

machines, memory requirements can increase considerably after this, but then so does the run time making this less practical. It is recommended that some form of over-modelling adjustment is performed on this resulting DAG (unless dealing with vast numbers of observations), for example using parametric bootstrapping which is straightforward to implement in MCMC engines such as JAGS or WinBUGS. See the case studies at http://r-bayesian-networks.org for details.

The parameter prior.choice determines the prior used within each individual node for a given choice of parent combination. In Koivisto and Sood (2004) p.554 a form of prior is used which assumes that the prior probability for parent combinations comprising of the same number of parents are all equal. Specifically, that the prior probability for parent set G with cardinality |G| is proportional to 1/[n-1 choose |G|] where there are n total nodes. Note that this favours parent combinations with either very low or very high cardinality which may not be appropriate. This prior is used when prior.choice=2. When prior.choice=1 an uninformative prior is used where parent combinations of all cardinalities are equally likely. The latter is equivalent to the structural prior used in the heuristic searches e.g. `search.hillclimber`.

Note that the network score (log marginal likelihood) of the most probable DAG is not returned as it can easily be computed using `fitabn`, see examples below.

Value

An object of class `abnMostprobable`, which is a list containing: a matrix giving the DAG definition of the most probable posterior structure, the cache of pre-computed scores and the score used for selection.

Author(s)

Fraser Iain Lewis

References

Koivisto, M. V. (2004). Exact Structure Discovery in Bayesian Networks, Journal of Machine Learning Research, vol 5, 549-573.

Further information about **abn** can be found at:
http://r-bayesian-networks.org

Examples

```
## Not run:
## Use a subset of a built-in simulated data set
mydat <- ex0.dag.data[,c("b1","b2","g1","g2","p1","p2")]

## Setup distribution list for each node
mydists <- list(b1="binomial", b2="binomial",
              g1="gaussian", g2="gaussian",
              p1="poisson", p2="poisson")

## Parent limits
max.par <- list("b1"=1, "b2"=1, "g1"=1, "g2"=0, "p1"=1, "p2"=2)

## Now build cache with structural constrains:
##   now ban arc from b2 to b1
```

```
##   always retain arc from g2 to g1

mycache <- buildscorecache(data.df=mydat, data.dists=mydists,
                     dag.banned= ~b1|b2, dag.retained= ~g1|g2, max.parents=max.par)

## Now find the globally best DAG
mp.dag <- mostprobable(score.cache=mycache)

## Get the corresponding best goodness-of-fit network score
fitabn(object = mp.dag)$mlik

##############################
## Second example
##############################

## this data comes with abn see ?ex1.dag.data
mydat <- ex1.dag.data

## setup distribution list for each node
mydists <- list(b1="binomial", p1="poisson", g1="gaussian", b2="binomial",
              p2="poisson", b3="binomial", g2="gaussian", b4="binomial",
              b5="binomial", g3="gaussian")

## assum no constraints in ban nor retain

## parent limits
max.par <- list("b1"=4,"p1"=4,"g1"=4,"b2"=4,"p2"=4,"b3"=4,"g2"=4,"b4"=4,"b5"=4,"g3"=4)
## now build cache
mycache <- buildscorecache(data.df = mydat, data.dists = mydists, max.parents = max.par)

## Find the globally best DAG
mp.dag <- mostprobable(score.cache=mycache)
fitabn(object=mp.dag)$mlik

## plot the best model
myres <- fitabn(object=mp.dag,create.graph=TRUE)

plotabn(dag.m = mp.dag$dag, data.dists = mydists)#, fitted.values.abn = myres)

## fit the known true DAG
true.dag <- matrix(data=0,ncol=10, nrow = 10)
colnames(true.dag) <- rownames(true.dag) <- names(mydists)

true.dag["p2",c("b1","p1")] <- 1
true.dag["b3",c("b1","g1","b2")] <- 1
true.dag["g2",c("p1","g1","b2")] <- 1
true.dag["b4",c("g1","p2")] <- 1
true.dag["b5",c("g1","g2")] <- 1
true.dag["g3",c("g1","b2")] <- 1

fitabn(dag.m = true.dag, data.df = mydat, data.dists=mydists)$mlik

##################################################################
```

```
## example 3 - models with random effects
#####################################################################

## this data comes with abn see ?ex3.dag.data
mydat <- ex3.dag.data[,c(1:4,14)]

mydists <- list(b1="binomial", b2="binomial", b3="binomial",
              b4="binomial")

## This takes a few seconds
mycache.mixed <- buildscorecache(data.df = mydat, data.dists = mydists,
               group.var = "group", cor.vars = c("b1","b2","b3","b4"),
               max.parents=2, which.nodes=c(1:4))

## find the most probable DAG
mp.dag <- mostprobable(score.cache=mycache.mixed)

## get goodness of fit
fitabn(object = mp.dag, data.df = mydat, data.dists = mydists,
      group.var = "group", cor.vars = c("b1","b2","b3","b4"))$mlik

## End(Not run)
```

`or` *Odds Ratio from a Table*

Description

Compute the odds ratio from a table or a matrix.

Usage

```
or(x)
```

Arguments

`x`	a 2x2 table or matrix.

Details

Compute the odds ratio from a table or a matrix.

Value

A real value.

Author(s)

Gilles Kratzer

pigs.vienna — *Dataset related to diseases present in 'finishing pigs', animals about to enter the human food chain at an abattoir.*

Description

The data we consider here comprise of a randomly chosen batch of 50 pigs from each of 500 randomly chosen pig producers in the UK. The dataset consists of 25000 observations 10 binary variables and a group variable. These are 'finishing pigs', animals about to enter the human food chain at an abattoir. Further description of the data set is present on the vignette.

Format

A data frame with a mixture of 10 discrete variables, each of which is set as a factor, and a group variable.

PC Binary.

PT Binary.

MS Binary.

HS Binary.

TAIL Binary.

Abscess Binary.

Pyaemia Binary.

EPcat Binary.

PDcat Binary.

plbinary Binary.

batch Group variable, corresponding to the 500 different pig producers

References

Hartnack, S., et al. (2016) "Attitudes of Austrian veterinarians towards euthanasia in small animal practice: impacts of age and gender on views on euthanasia." BMC Veterinary Research 12.1: 26.

plotabn — *Plot an ABN graphic*

Description

Plot an ABN DAG using formula statement or a matrix in using Rgraphviz through the graphAM class.

Usage

```
plotabn(dag.m = NULL, data.dists = NULL, markov.blanket.node =
                NULL, fitted.values.abn = NULL, fitted.values.abn.mle
                = NULL, digit.precision = 2, arc.strength = NULL,
                edgemode = "directed", edgedir = "pc", node.fillcolor
                = "lightblue", edge.color = "black", edge.arrowwise =
                0.5, fontsize.node = 10, fontsize.edge = 5, plot =
                TRUE, node.fillcolor.list = NULL)
```

Arguments

`dag.m`	a matrix or a formula statement (see details for format) defining the network structure, a Directed Acyclic Graph (DAG). Note that rownames must be set or given in `data.dists`.
`data.dists`	a named list giving the distribution for each node in the network, see details.
`markov.blanket.node`	name of variables to display his Markov Blanket.
`fitted.values.abn`	modes outputted from `fitabn`.
`fitted.values.abn.mle`	regression coefficients outputted from `fitabn`.
`digit.precision`	number of digits of the output from `fitabn` or `fitabn`.
`arc.strength`	a named matrix containing evaluations of arcs strength which will change the arcs width (could be Mutual information, p-values, number of bootstrap retrieve samples or the outcome of the `link.strength`).
`edgemode`	a "character" vector specifying whether the graph is "directed" or "undirected".
`edgedir`	character giving the direction in which arcs should be plotted, `pc` (parent to child) or `cp` (child to parent).
`node.fillcolor`	the colour of the node.
`edge.color`	the colour of the edge.
`edge.arrowwise`	the thickness of the arrows. Not relevant if `arc.strength` is provided.
`fontsize.node`	the font size of the nodes names.
`fontsize.edge`	the font size of the arcs fitted values.
`plot`	logical variable, if set to `TRUE` return a plot of the DAG, if set to `FALSE` return the adjacency matrix of the given DAG.
`node.fillcolor.list`	the list of node that should be coloured.

Details

By default binomial nodes are squares, multinoial nodes are empty, Gaussian nodes are circles and poison nodes are ellipses.

The dag.m can be provided using a formula statement (similar to glm). A typical formula is ~ node1|parent1:parent2 + node2:node3|parent3. The formula statement have to start with ~. In this example, node1 has two parents (parent1 and parent2). node2 and node3 have the same parent3. The parents names have to exactly match those given in name. : is the separator between either children or parents, | separates children (left side) and parents (right side), + separates terms, . replaces all the variables in name.

Value

A matrix giving the DAG definition of the most probable posterior structure.

Author(s)

Gilles Kratzer

References

Further information about **abn** can be found at:
http://r-bayesian-networks.org

Examples

```
#Define distribution list
dist <- list(a="gaussian", b="gaussian", c="gaussian", d="gaussian", e="binomial", f="binomial")

#Define a matrix formulation
arc.strength <- matrix(c(0,0.5,0.5,0.7,0.1,0,
                         0,0,0.3,0.1,0,0.8,
                         0,0,0,0.35,0.66,0,
                         0,0,0,0,0.9,0,
                         0,0,0,0,0,0.8,
                         0,0,0,0,0,0),nrow = 6L, ncol = 6L, byrow = TRUE)

#Naming of the matrix
colnames(arc.strength) <- rownames(arc.strength) <- names(dist)

#Plot from a formula
plotabn(dag.m = ~a|b:c:e+b|c:d:f+e|f, data.dist = dist)

#Plot form a matrix
plotabn(dag.m = arc.strength, data.dist = dist)

#Creating adjency matrix
plotabn(dag.m = ~a|b:c:e+b|c:d:f+e|f, data.dist = dist, plot = FALSE)

#Arc strength
plotabn(dag.m = ~a|b:c:e+b|c:d:f+e|f, data.dist = dist, arc.strength = arc.strength)

#Markov blanket
plotabn(dag.m = ~a|b:c:e+b|c:d:f+e|f, data.dists = dist, markov.blanket.node = "e")
```

scoreContribution *Compute the score contribution of each individual observation in a network.*

Description

This function computes the score contribution of each individual observations to the total network score.

Usage

```
scoreContribution(object = NULL,
                  dag.m = NULL, data.df = NULL, data.dists = NULL,
                  verbose = FALSE)
```

Arguments

object	an object of class 'abnLearned' produced by mostprobable, search.heuristic or search.hillclimber.
dag.m	a matrix or a formula statement (see details) defining the network structure, a directed acyclic graph (DAG), see details for format. Note that colnames and rownames must be set.
data.df	a data frame containing the data used for learning the network, binary variables must be declared as factors and no missing values all allowed in any variable.
data.dists	a named list giving the distribution for each node in the network, see details.
verbose	if TRUE then provides some additional output.

Details

This function computes the score contribution of each individual observations to the total network score. This function is available only in the 'mle' settings. To do so one uses the glm and predict functions. This funtion is an attempt to perform diagnostic for an ABN analysis.

Value

A named list that contains the scores controbutions: maximum likelihood, aic, bic, mdl and diagonal values of the hat matrix.

Author(s)

Gilles Kratzer

Examples

```
## Use a subset of a built-in simulated data set
mydat <- ex1.dag.data[,c("b1","g1","p1")]

## setup distribution list for each node
mydists <- list(b1="binomial", g1="gaussian", p1="poisson")

## now build cache
mycache <- buildscorecache(data.df = mydat, data.dists = mydists, max.parents = 2, method = "mle")

## Find the globally best DAG
mp.dag <- mostprobable(score.cache=mycache, score="bic", verbose = FALSE)

out <- scoreContribution(object = mp.dag)

## Observations contribution per network node
boxplot(out$bic)
```

`searchHeuristic` *A familly of heuristic algorithms that aims at finding high scoring directed acyclic graphs*

Description

A flexible implementation of multiple greedy search algorithms to find high scoring network (DAG)

Usage

```
searchHeuristic(score.cache = NULL,
                        score = "mlik",
                        num.searches = 1,
                max.steps = 100,
                seed = 42,
                verbose = FALSE,
                start.dag = NULL,
                algo = "hc",
                tabu.memory = 10,
                temperature = 0.9, ...)
```

Arguments

`score.cache`	output from `buildscorecache()`.
`score`	which score should be used to score the network. Possible choices `aic, bic, mdl, mlik`.
`num.searches`	a positive integer giving the number of different search to run, see details.
`max.steps`	a constant giving the number of search steps per search, see details.

seed	a non-negative integer which sets the seed.
verbose	extra output, see output for details.
start.dag	a DAG given as a matrix, see details for format, which can be used to explicity provide a starting point for the structural search.
algo	which heuristic algorithm should be used. Possible choices are: hc, tabu, sa.
tabu.memory	a non-negative integer number to set the memory of the tabu search.
temperature	a real number giving the update in temperature for the sa (simulated annealing) search algorithm.
...	additional arguments passed for optimization.

Details

This function contains a flexible implementation of multiple greedy heuristic algorithm particularly well adapted to abn framework.

This function is a flexible implementation of multiple greedy search algorithms to find high scoring network.

The hill-climber procedure hc is similar, but not identical, to the method presented in Heckerman et al. 1995. Each search begins with a choice of node ordering and then randomly choose a DAG structure or a user given matrix. If a unique matrix is provided no attemp will

The procedure runs a greedy hill climbing search similar, but not identical, to the method presented initially in Heckerman et al. 1995. Each search begins with a randomly chosen DAG structure where this is constructed in such a way as to attempt to choose a DAG uniformly from the vast landscape of possible structures. The algorithm used is as follows: given a node cache (from buildscorecache), then within this set of all allowed local parent combinations, a random combination is chosen for each node. This is then combined into a full DAG which is then checked for cycles, where this check iterates over the nodes in a random order. If all nodes have at least one child (i.e. at least one cycle is present) then the first node examined has all its children removed, and the check for cycles is then repeated. If this has removed the only cycle present then this DAG is used at the starting point for the search, if not a second node is chosen (randomly) and the process is then repeated until a DAG is obtained.

The actual hill climbing algorithm itself differs slightly from that presented in Heckerman et al. as a full cache of all possible local combinations are available. At each hill-climbing step everything in the node cache is considered, in other words all possible single swaps between members of cache currently present in the DAG and those in the full cache. The single swap which provides the greatest increase in goodness of fit is chosen. A single swap here is the removal or addition of any one node-parent combination present in the cache while avoiding a cycle. This means that as well as all single arc changes (addition or removal), multiple arc changes are also considered at each same step, note however that arc reversal in this scheme takes two steps (as this requires first removal of a parent arc from one node and then addition of a parent arc to a different node). The original algorithm perturbed the current DAG by only a single arc at each step but also included arc reversal. The current implementation may not be any more efficient that the original but is arguably more natural given a pre-computed cache of local scores.

A start DAG may be provided in which case num.searches must equal 1 - this option is really just to provide a local search around a previously identified optimal DAG.

This function is designed for two different purposes: i) interactive visualisation; and ii) longer batch processing. The former prvoides easy visual "eyeballing" of data in terms of its majority consensus network (or similar threshold), which is a graphical structure which comprises of all arcs which feature in a given proportion (`support.threshold`) of locally optimal DAGs already identified during the run. For example, running 1000 searches with `trace=TRUE` will after each new search plot the current consensus network based on all previous searches. The general hope is that this structure will stabilize - become fixed - relatively quickly, at least for problems with smaller numbers of nodes. Note that library Rgraphviz is needed when `trace=TRUE`. When `trace=FALSE` then there is no graphical output and as such is rather faster. The format of results is identical in each case but note that the choice of random starting networks (the only random part in the algorithm) used when `trace=TRUE` and `trace=FALSE` will differ as these use different random number streams (the former uses R's random number generator, whereas the latter uses gsl's).

Value

An object of class `abnHillClimber` (which extends the class `abnLearnd`) and contains list with entires:

`init.score`	a vector giving network score for initial network from which the search commenced
`final.score`	a vector giving the network score for the locally optimal network
`init.dag`	list of matrices, inital DAGs
`final.dag`	list of matrices, locally optimal DAGs
`consensus`	a matrix holding a DAG
`support.threshold`	percentage supported used to create consensus matrix

Author(s)

Fraser Iain Lewis

References

Lewis, F. I., and McCormick B. J. J. (2012). Revealing the complexity of health determinants in resource poor settings. *American Journal Of Epidemiology*. DOI:10.1093/aje/KWS183).

Heckerman, D., Geiger, D. and Chickering, D. M. (1995). Learning Bayesian networks: The combination of knowledge and statistical data. *Machine Learning*, 20, 197-243.

Further information about **abn** can be found at:
http://r-bayesian-networks.org

Examples

```
## Not run:

###############################################
## example: use built-in simulated data set
###############################################
```

```
mydat <- ex1.dag.data ## this data comes with abn see ?ex1.dag.data

## setup distribution list for each node
mydists<-list(b1="binomial", p1="poisson", g1="gaussian", b2="binomial",
              p2="poisson", b3="binomial", g2="gaussian", b4="binomial",
              b5="binomial", g3="gaussian")

mycache <- buildscorecache(data.df = mydat, data.dists = mydists, max.parents = 2)

## Now peform 100 greedy searches
heur.res <- searchHeuristic(score.cache = mycache, data.dists = mydists,
              start.dag = "random", num.searches = 10,
              max.steps = 50)

## Plot (one) dag
plotabn(heur.res$dags[[1]], data.dists = mydists)

## End(Not run)
```

`searchHillclimber` *Find high scoring directed acyclic graphs using heuristic search.*

Description

Find high scoring network (DAG) structures using a random re-starts greedy hill-climber heuristic search.

Usage

```
searchHillclimber(score.cache = NULL, num.searches = 1, seed = 0,
         verbose = FALSE, timing.on = TRUE, start.dag = NULL,
         support.threshold = 0.5, create.graph = FALSE, dag.retained = NULL)
```

Arguments

`score.cache`	output from `buildscorecache()`.
`num.searches`	number of times to run the search.
`seed`	non-negative integer which sets the seed in the GSL random number generator.
`verbose`	extra output.
`timing.on`	extra output in terms of duration computation.
`start.dag`	a DAG given as a matrix, see details for format, which can be used to explicitly provide a starting point for the structural search,
`support.threshold`	the proportion of search results - each locally optimal DAG - in which each arc must appear to be a part of the consensus network.
`create.graph`	save the final graph as an R graph object.

`dag.retained` a DAG given as a matrix, see details for format. This is necessary if the score.cache was created using an explicit retain matrix, and the same retain matrix should be used here. dag.retained is used by the algorithm which generates the initial random DAG to ensure that the necessary arcs are retained.

Details

The procedure runs a greedy hill climbing search similar, but not identical, to the method presented initially in Heckerman et al. 1995. (Machine Learning, 20, 197-243). Each search begins with a randomly chosen DAG structure where this is constructed in such a way as to attempt to choose a DAG uniformly from the vast landscape of possible structures. The algorithm used is as follows: given a node cache (from buildscorecache()), then within this set of all allowed local parent combinations, a random combination is chosen for each node. This is then combined into a full DAG which is then checked for cycles, where this check iterates over the nodes in a random order. If all nodes have at least one child (i.e. at least one cycle is present) then the first node examined has all its children removed, and the check for cycles is then repeated. If this has removed the only cycle present then this DAG is used at the starting point for the search, if not a second node is chosen (randomly) and the process is then repeated until a DAG is obtained.

The actual hill climbing algorithm itself differs slightly from that presented in Heckerman et al. as a full cache of all possible local combinations are available. At each hill-climbing step everything in the node cache is considered, in other words all possible single swaps between members of cache currently present in the DAG and those in the full cache. The single swap which provides the greatest increase in goodness of fit is chosen. A single swap here is the removal or addition of any one node-parent combination present in the cache while avoiding a cycle. This means that as well as all single arc changes (addition or removal), multiple arc changes are also considered at each same step, note however that arc reversal in this scheme takes two steps (as this requires first removal of a parent arc from one node and then addition of a parent arc to a different node). The original algorithm perturbed the current DAG by only a single arc at each step but also included arc reversal. The current implementation may not be any more efficient that the original but is arguably more natural given a pre-computed cache of local scores.

A start DAG may be provided in which case num.searches must equal 1 - this option is really just to provide a local search around a previously identified optimal DAG.

This function is designed for two different purposes: i) interactive visualisation; and ii) longer batch processing. The former provides easy visual "eyeballing" of data in terms of its majority consensus network (or similar threshold), which is a graphical structure which comprises of all arcs which feature in a given proportion (`support.threshold`) of locally optimal DAGs already identified during the run. The general hope is that this structure will stabilize - become fixed - relatively quickly, at least for problems with smaller numbers of nodes.

Value

A list with entries:

`init.score` a vector giving network score for initial network from which the search commenced

`final.score` a vector giving the network score for the locally optimal network

`init.dag` list of matrices, initial DAGs

`final.dag` list of matrices, locally optimal DAGs

`consensus` a matrix holding a DAG

`support.threshold` percentage supported used to create consensus matrix

Author(s)

Fraser Iain Lewis

References

Lewis, F. I., and McCormick, B. J. J. (2012). Revealing the complexity of health determinants in resource poor settings. *American Journal Of Epidemiology*. DOI:10.1093/aje/KWS183).

Further information about **abn** can be found at:
http://r-bayesian-networks.org

Examples

```
## Not run:
##############################################
## example 1: use built-in simulated data set
##############################################

## this data comes with abn see ?ex1.dag.data
mydat <- ex1.dag.data

## setup distribution list for each node
mydists <- list(b1="binomial", p1="poisson", g1="gaussian", b2="binomial",
              p2="poisson", b3="binomial", g2="gaussian", b4="binomial",
              b5="binomial", g3="gaussian")

## Build cache may take some minutes for buildscorecache()
mycache <- buildscorecache(data.df = mydat, data.dists = mydists,
                           max.parents = 2);

# now peform 100 greedy searches
heur.res <- searchHillclimber(score.cache = mycache,
                 num.searches = 100, timing.on = FALSE)
plotabn(dag.m = heur.res$consensus, data.dists = mydists)

############################
## example 2 - glmm example
############################

## this data comes with abn see ?ex1.dag.data
mydat <- ex3.dag.data[,c(1:4,14)]

mydists <- list(b1="binomial", b2="binomial", b3="binomial",
              b4="binomial")

## This takes a few seconds
mycache.mixed <- buildscorecache(data.df = mydat, data.dists = mydists,
               group.var = "group", cor.vars = c("b1","b2","b3","b4"),
```

```
                max.parents=2, which.nodes=c(1:4))

## Now peform 50 greedy searches
heur.res <- searchHillclimber(score.cache = mycache.mixed, num.searches = 50,
                              timing.on = FALSE)
##  Plot the majority consensus network
plotabn(dag.m = heur.res$consensus, data.dists = mydists)


## End(Not run)
```

`simulateAbn` *Simulate from an ABN Network*

Description

Simulate one or more responses from an ABN network corresponding to a fitted object using formula statement or an adjacency matrix.

Usage

```
simulateAbn(data.dists = NULL,
                        data.param = NULL,
                        data.param.var = NULL,
                        data.param.mult = NULL,
                        n.chains =10,
                        n.adapt = 1000,
                        n.thin = 100,
                        n.iter = 10000,
                        bug.file=NULL,
                        verbose=TRUE,
                        simulate=TRUE,
                        keep.file=FALSE,
                        seed=42)
```

Arguments

`data.dists`	named list giving the distribution for each node in the network, see 'Details'.
`data.param`	named matrix, which have to be square with as many entries as the number of variables, each element is the coefficient (for specifications see 'Details') used in the glm to simulate responses.
`data.param.var`	optional matrix, which should be square and having as many entries as number of variables, which contains the precision values for gaussian nodes. Default is set to 1.
`data.param.mult`	optional matrix, which should be square and having as many entries as number of variables, which contains the multinomial coefficient.

n.chains	number of parallel chains for the model.
n.thin	number of parallel chains for the model.
n.iter	number of iteration to monitor.
n.adapt	number of iteration for adaptation. If n.adapt is set to zero, then no adaptation takes place.
bug.file	path/name of the user specific bug file.
verbose	logical. Default TRUE. Should R report extra information on progress?
simulate	logical. Default TRUE. If set to FALSE, no simulation will be run only creation of the bug file.
keep.file	logical. Default FALSE. If set to TRUE, the model.bug file generated will be kept afterwards.
seed	by default set to 42.

Details

This function use rjags to simulate data from a DAG. It first creates a bug file, in the actual repository, then use it to simulate the data. This function output a data frame. The bug file can be run using rjags separately.

The coefficients given in the data.param are: the logit of the probabilities for binomial nodes, the means of the gaussian nodes and the log of the Poison parameter. Additionally, a matrix data.param.var could give precision values for gaussian nodes (default is set to 1).

Binary and multinomial variables must be declared as factors, and the argument data.dists must be a list with named arguments, one for each of the variables in data.df (except a grouping variable - if present), where each entry is either "poisson","binomial", "multinomial" or "gaussian", see examples below. The "poisson" distributions use log and "binomial" and "multinomial" distributions logit link functions. Note that "binomial" here actually means only binary, one bernoulli trial per row in data.df.

The number of simulated data (rows of the outputted data frame) is given by n.iter divided by n.thin.

Value

A data frame containing simulated data.

Author(s)

Gilles Kratzer

References

Further information about **abn** can be found at:
http://r-bayesian-networks.org

Examples

```
## Define set of distributions:
dist<-list(a="gaussian", b="gaussian", c="gaussian", d="gaussian",
           e="binomial", f="binomial")

## Define parameter matrix:
data.param <- matrix(c(1,2,0.5,0,20,0,
                       0,1,3,10,0, 0.8,
                       0,0,1,0,0,0,
                       0,0,0,1,0,0,
                       0,0,0,0,0.5,1,
                       0,0,0,0,0,0), nrow = 6L, ncol = 6L, byrow = TRUE)

## Define precision matrix:
data.param.var <- matrix(0, nrow = 6L, ncol = 6L)
diag(data.param.var) <- c(10,20,30,40,0,0)

## Plot the dag
plotabn(dag.m = ~a|b:c:e+b|c:d:f+e|f, data.dists = dist, plot = FALSE)

## Simulate the data
simulateAbn(data.dists=dist, n.chains=1, n.thin=1, n.iter=1000,
            data.param=data.param, data.param.var=data.param.var)
```

`simulateDag` *Simulate DAGs*

Description

Simulate a Directed Acyclic Graph (ABN) with arbitrary arc density.

Usage

```
simulateDag(node.name = NULL,
            data.dists = NULL,
            nn = 0.5)
```

Arguments

`node.name`	a vector of character giving the names of the nodes. It gives the size of the simulated DAG.
`data.dists`	named list giving the distribution for each node in the network. If not provided it will be sample and returned.
`nn`	a real number between 0 and 1 giving the network density.

Details

This function generates DAGs by sampling triangular matrices and reorder randomly columns and rows. The network density (nn) is used column wise as binomial sampling probability. Then the matrix is named using the user provided names.

Value

An object of class abnDag a named matrix and a named list giving the distribution for each node.

Author(s)

Gilles Kratzer

References

Further information about **abn** can be found at:
http://r-bayesian-networks.org

Examples

```
## Example using Ozon entries:
dist <- list(Ozone="gaussian",   Solar.R="gaussian",  Wind="gaussian",
             Temp="gaussian",    Month="gaussian",    Day="gaussian")
out <- simulateDag(node.name = names(dist), data.dists = dist, nn = 0.8)
## Not run:
plot(out)

## End(Not run)
```

---

tographviz *Convert a DAG into graphviz format*

Description

Given a matrix defining a DAG create a text file suitable for plotting with graphviz

Usage

```
tographviz(dag.m, data.df, data.dists, group.var=NULL, outfile, directed=TRUE)
```

Arguments

dag.m	a matrix defining a DAG.
data.df	a data frame containing the data used for learning the network.
data.dists	a list with named arguments matching the names of the data frame which gives the distribution family for each variable. See fitabn for details.

group.var	only applicable for mixed models and gives the column name in data.df of the grouping variable (which must be a factor denoting group membership). See fitabn for details.
outfile	a character string giving the filename which will contain the graphviz graph.
directed	logical; if TRUE, a directed acyclic graph is produced, otherwise an undirected graph.

Details

Graphviz (http://www.graphviz.org) is visualisation software developed by AT&T and freely available. This function creates a text representation of the DAG, or the undirected graph, so this can be plotted using graphviz. Graphviz is available as an R package, Rgraphviz, through the Bioconductor project http://www.bioconductor.org/ (and requires a working installation of graphviz). Binary nodes will appear as squares, Gaussian as ovals and Poisson nodes as diamonds in the resulting graphviz network diagram. There are many other shapes possible for nodes and numerous other visual enhancements - see online graphviz documentation. Bespoke refinements can be added by editing the raw outfile produced. For full manual editing, particularly of the layout, or adding annotations, one easy solution is to convert a postscript format graph (produced in graphviz using the -Tps switch) into a vector format using a tool such as pstoedit http://www.pstoedit.net, and then edit using a vector drawing tool like xfig. This can then be resaved as postscript or pdf thus retaining full vector quality.

Value

Nothing is returned, but a file outfile written.

Author(s)

Fraser Iain Lewis

References

Further information about **abn** can be found at:
http://r-bayesian-networks.org

Examples

```
## On a typical linux system the following code constructs a nice
## looking pdf file 'graph.pdf'.
## Not run:
## Subset of a build-in dataset
mydat <- ex0.dag.data[,c("b1","b2","b3","g1","b4","p2","p4")]

## setup distribution list for each node
mydists <- list(b1="binomial", b2="binomial", b3="binomial",
                g1="gaussian", b4="binomial", p2="poisson",
                p4="poisson")
## specify DAG model
mydag <- matrix(c(   0,1,0,0,1,0,0, #
                     0,0,0,0,0,0,0, #
```

```
                    0,1,0,0,1,0,0, #
                    1,0,0,0,0,0,1, #
                    0,0,0,0,0,0,0, #
                    0,0,0,1,0,0,0, #
                    0,0,0,0,1,0,0  #
                    ), byrow=TRUE, ncol=7)

colnames(mydag) <- rownames(mydag) <- names(mydat)

## create file for processing with graphviz
tographviz(dag.m=mydag, data.df= mydat, data.dists=mydists,
          outfile="graph.dot", directed=TRUE)
## and then process using graphviz tools e.g. on linux
# system("dot -Tpdf -o graph.pdf graph.dot")
# system("evince graph.pdf")

## Example using data with a group variable  where b1<-b2
mydag <- matrix(c(0,1, 0,0), byrow=TRUE, ncol=2)

colnames(mydag) <- rownames(mydag) <- names(ex3.dag.data[,c(1,2)])
## specific distributions
mydists <- list(b1="binomial", b2="binomial")

## create file for processing with graphviz
tographviz(dag.m=mydag, data.df=ex3.dag.data[,c(1,2,14)], data.dists=mydists,
          group.var="group", outfile="graph.dot", directed=FALSE)
## and then process using graphviz tools e.g. on linux
# system("dot -Tpdf -o graph.pdf graph.dot")
# system("evince graph.pdf");

## End(Not run)
```

`var33` *simulated dataset from a DAG comprising of 33 variables*

Description

250 observations simulated from a DAG with 17 binary variables and 16 continuous. A DAG of this data features in the vignette. Note that the conditional dependence relations given are those in the population and may differ in the realization of 250 observations.

Format

A data frame with a mixture of discrete variables each of which is set as a factor and continuous variables. Joint distribution structure used to generate the data.

v1 Binary, independent.

v2 Gaussian, conditionally dependent upon v1.

v3 Binary, independent.

v4 Binary, conditionally dependent upon v3.

v5 Gaussian, conditionally dependent upon v6.

v6 Binary, conditionally dependent upon v4 and v7.

v7 Gaussian, conditionally dependent upon v8.

v8 Gaussian, conditionally dependent upon v9.

v9 Binary, conditionally dependent upon v10.

v10 Binary, independent.

v11 Binary, conditionally dependent upon v10, v12 and v19.

v12 Binary, independent.

v13 Gaussian, independent.

v14 Gaussian, conditionally dependent upon v13.

v15 Binary, conditionally dependent upon v14 and v21.

v16 Gaussian, independent.

v17 Gaussian, conditionally dependent upon v16 and v20.

v18 Binary, conditionally dependent upon v20.

v19 Binary, conditionally dependent upon v20.

v20 Binary, independent.

v21 Binary, conditionally dependent upon v20.

v22 Gaussian, conditionally dependent upon v21.

v23 Gaussian, conditionally dependent upon v21.

v24 Gaussian, conditionally dependent upon v23.

v25 Gaussian, conditionally dependent upon v23 and v26.

v26 Binary, conditionally dependent upon v20.

v27 Binary, independent.

v28 Binary, conditionally dependent upon v27, v29 and v31.

v29 Gaussian, independent.

v30 Gaussian, conditionally dependent upon v29.

v31 Gaussian, independent.

v32 Binary, conditionally dependent upon v21, v29 and v31.

v33 Gaussian, conditionally dependent upon v31.

Examples

```
## Constructing the DAG of the dataset:
dag33 <- matrix(0, 33, 33)
dag33[2,1] <- 1
dag33[4,3] <- 1
dag33[6,4] <- 1; dag33[6,7] <- 1
dag33[5,6] <- 1
```

```
dag33[7,8] <- 1
dag33[8,9] <- 1
dag33[9,10] <- 1
dag33[11,10] <- 1; dag33[11,12] <- 1; dag33[11,19] <- 1;
dag33[14,13] <- 1
dag33[17,16] <- 1; dag33[17,20] <- 1
dag33[15,14] <- 1; dag33[15,21] <- 1
dag33[18,20] <- 1
dag33[19,20] <- 1
dag33[21,20] <- 1
dag33[22,21] <- 1
dag33[23,21] <- 1
dag33[24,23] <- 1
dag33[25,23] <- 1; dag33[25,26] <- 1
dag33[26,20] <- 1
dag33[33,31] <- 1
dag33[33,31] <- 1
dag33[32,21] <- 1; dag33[32,31] <- 1; dag33[32,29] <- 1
dag33[30,29] <- 1
dag33[28,27] <- 1; dag33[28,29] <- 1; dag33[28,31] <- 1
```

Index