Package 'abn'

September 6, 2023

Version 3.0.0

Date 2023-09-01

Title Modelling Multivariate Data with Additive Bayesian Networks

Author Matteo Delucchi [aut, cre] (<https://orcid.org/0000-0002-9327-1496>),
Reinhard Furrer [aut] (<https://orcid.org/0000-0002-6319-2332>),
Gilles Kratzer [aut] (<https://orcid.org/0000-0002-5929-8935>),
Fraser Iain Lewis [aut] (<https://orcid.org/0000-0003-4580-2712>),
Marta Pittavino [ctb] (<https://orcid.org/0000-0002-1232-1034>),
Kalina Cherneva [ctb]

Maintainer Matteo Delucchi <matteo.delucchi@math.uzh.ch>

Depends R (>= 4.0.0)

Imports methods, rjags, nnet, lme4, graph, Rgraphviz, doParallel, foreach, mclogit, stringi, Rcpp

LinkingTo Rcpp, RcppArmadillo

Suggests INLA, knitr, R.rsp, testthat (>= 3.0.0), entropy, moments, boot, brglm

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 <https://www.r-inla.org>.
The computing library JAGS <https://mcmc-jags.sourceforge.io> is used to simulate 'abn'-like data.
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

BugReports https://git.math.uzh.ch/mdeluc/abn/-/issues

Config/testthat/edition 3

RoxygenNote 7.2.2

Encoding UTF-8

NeedsCompilation yes

Repository CRAN

Date/Publication 2023-09-06 08:50:05 UTC

R topics documented:

`abn.version` *abn Version Information*

Description

`abn.version()` provides detailed information about the running version of **abn** or the **abn** components.

Usage

```
abn.version(what = c("abn", "system"))
```

Arguments

`what`	detailed information about the version of **abn** or the system (see returns).

Value

`abn.version(what = "system")` is a list with character-string components

- R`R.version.string`
- abnessentially `abn.version$version.string`
- GSL, JAGS, INLAversion numbers thereof

`abn.version(what = "abn")` is a list with character-string components

- statusthe status of the version (e.g., `"beta"`)
- majorthe major version number
- minorthe minor version number
- yearthe year the version was released
- monththe month the version was released
- daythe day the version was released
- version.stringa `character` string concatenating the info above, useful for plotting, etc.

`abn.version` is a list of class `"simple.list"` which has a `print` method.

See Also

`R.version`

Examples

```
abn.version()$version.string
## Not run:
  abn.version("system")

## End(Not run)
```

---

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).

- ARpresence of atrophic rhinitis.
- pneumSpresence of moderate to severe pneumonia.
- femalesex of the pig (1=female, 0=castrated).
- livdampresence of liver damage (parasite-induced white spots).
- eggspresence of fecal/gastrointestinal nematode eggs at time of slaughter.
- wormCountcount of nematodes in small intestine at time of slaughter.
- agedays elapsed from birth to slaughter (days).
- adgaverage daily weight gain (grams).
- farmfarm ID.

Details

When using the data to fit an additive Bayesian network, the variables `AR`, `pneumS`, `female`, `livdam`, `eggs` are considered binomial, `wormcount` Poisson, `age` and `adg` Gaussian. The variable `farm` can be used to adjust for grouping.

References

Kratzer, G., Lewis, F.I., Comin, A., Pittavino, M. and Furrer, R. (2019). "Additive Bayesian Network Modelling with the R Package abn". arXiv preprint arXiv:1911.09006. Dohoo, Ian Robert, Wayne Martin, and Henrik Stryhn. Veterinary epidemiologic research. No. V413 DOHv. Charlottetown, Canada: AVC Incorporated, 2003.

AIC.abnFit *Print AIC of objects of class* abnFit

Description

Print AIC of objects of class abnFit

Usage

```
## S3 method for class 'abnFit'
AIC(object, digits = 3L, verbose = TRUE, ...)
```

Arguments

object	Object of class abnFit
digits	number of digits of the results.
verbose	print additional output.
...	additional parameters. Not used at the moment.

BIC.abnFit *Print BIC of objects of class* abnFit

Description

Print BIC of objects of class abnFit

Usage

```
## S3 method for class 'abnFit'
BIC(object, digits = 3L, verbose = TRUE, ...)
```

Arguments

object	Object of class abnFit
digits	number of digits of the results.
verbose	print additional output.
...	additional parameters. Not used at the moment.

build.control *Control the iterations in* buildScoreCache

Description

Allow the user to set restrictions in the buildScoreCache for both the Bayesian and the MLE approach. Control function similar to fit.control.

Usage

```
build.control(
  method = "bayes",
  max.mode.error = 10,
  mean = 0,
  prec = 0.001,
  loggam.shape = 1,
  loggam.inv.scale = 5e-05,
  max.iters = 100,
  epsabs = 1e-07,
  error.verbose = FALSE,
  trace = 0L,
  epsabs.inner = 1e-06,
  max.iters.inner = 100,
  finite.step.size = 1e-07,
  hessian.params = c(1e-04, 0.01),
  max.iters.hessian = 10,
  max.hessian.error = 0.5,
  factor.brent = 100,
  maxiters.hessian.brent = 100,
  num.intervals.brent = 100,
  n.grid = 250,
  ncores = 0,
  max.irls = 100,
  tol = 1e-08,
  tolPwrss = 1e-07,
  check.rankX = "message+drop.cols",
  check.scaleX = "message+rescale",
  check.conv.grad = "message",
  check.conv.singular = "message",
  check.conv.hess = "message",
  xtol_abs = 1e-06,
  ftol_abs = 1e-06,
  trace.mblogit = FALSE,
  catcov.mblogit = "free",
  epsilon = 1e-06,
  seed = 9062019L
)
```

Arguments

method	a character that takes one of two values: "bayes" or "mle". Overrides method argument from buildScoreCache.
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 always to be used, then max.mode.error=100, to force INLA never to be used max.mod.error=0. See also fitAbn.
mean	the prior mean for all the Gaussian additive terms for each node. INLA argument control.fixed=list(mean.intercept=...) and control.fixed=list(mean=...).
prec	the prior precision ($\tau = \frac{1}{\sigma^2}$) for all the Gaussian additive term for each node. INLA argument control.fixed=list(prec.intercept=...) and control.fixed=list(prec=...).
loggam.shape	the shape parameter in the Gamma distribution prior for the precision in a Gaussian node. INLA argument control.family=list(hyper = list(prec = list(prior="loggamma",pa loggam.inv.scale)))).
loggam.inv.scale	the inverse scale parameter in the Gamma distribution prior for the precision in a Gaussian node. INLA argument control.family=list(hyper = list(prec = list(prior="loggamma",param=c(loggam.shape, loggam.inv.scale)))).
max.iters	total number of iterations allowed when estimating the modes in Laplace approximation. passed to .Call("fit_single_node", ...).
epsabs	absolute error when estimating the modes in Laplace approximation for models with no random effects. Passed to .Call("fit_single_node", ...).
error.verbose	logical, additional output in the case of errors occurring in the optimization. Passed to .Call("fit_single_node", ...).
trace	Non-negative integer. If positive, tracing information on the progress of the "L-BFGS-B" optimization is produced. Higher values may produce more tracing information. (There are six levels of tracing. To understand exactly what these do see the source code.). Passed to .Call("fit_single_node", ...).
epsabs.inner	absolute error in the maximization step in the (nested) Laplace approximation for each random effect term. Passed to .Call("fit_single_node", ...).
max.iters.inner	total number of iterations in the maximization step in the nested Laplace approximation. Passed to .Call("fit_single_node", ...).
finite.step.size	suggested step length used in finite difference estimation of the derivatives for the (outer) Laplace approximation when estimating modes. Passed to .Call("fit_single_node", ...).
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 initial guess, second entry absolute error. Passed to .Call("fit_single_node", ...).
max.iters.hessian	integer, maximum number of iterations to use when determining an optimal finite difference approximation (Nelder-Mead). Passed to .Call("fit_single_node", ...).

`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. Passed to `.Call("fit_single_node", ...)`.

`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 $h/factor.brent, h * factor.brent)$. Passed to `.Call("fit_single_node", ...)`.

`maxiters.hessian.brent`
: maximum number of iterations allowed in the Brent-Dekker method. Passed to `.Call("fit_single_node", ...)`.

`num.intervals.brent`
: the number of initial different bracket segments to try in the Brent-Dekker method. Passed to `.Call("fit_single_node", ...)`.

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

`ncores`
: The number of cores to parallelize to, see 'Details'. If >0, the number of CPU cores to be used. -1 for all available -1 core. Only for `method="mle"`.

`max.irls`
: total number of iterations for estimating network scores using an Iterative Reweighed Least Square algorithm. Is this DEPRECATED?

`tol`
: real number giving the minimal tolerance expected to terminate the Iterative Reweighed Least Square algorithm to estimate network score. Passed to `irls_binomial_cpp_fast_br` and `irls_poisson_cpp_fast`.

`tolPwrss`
: numeric scalar passed to `glmerControl` - the tolerance for declaring convergence in the penalized iteratively weighted residual sum-of-squares step. Similar to `tol`.

`check.rankX`
: character passed to `lmerControl` and `glmerControl` - specifying if `rankMatrix(X)` should be compared with `ncol(X)` and if columns from the design matrix should possibly be dropped to ensure that it has full rank. Defaults to `message+drop.cols`.

`check.scaleX`
: character passed to `lmerControl` and `glmerControl` - check for problematic scaling of columns of fixed-effect model matrix, e.g. parameters measured on very different scales. Defaults to `message+rescale`.

`check.conv.grad`
: character passed to `lmerControl` and `glmerControl` - checking the gradient of the deviance function for convergence. Defaults to `message` but can be one of "ignore" - skip the test; "warning" - warn if test fails; "message" - print a message if test fails; "stop" - throw an error if test fails.

`check.conv.singular`
: character passed to `lmerControl` and `glmerControl` - checking for a singular fit, i.e. one where some parameters are on the boundary of the feasible space (for example, random effects variances equal to 0 or correlations between random effects equal to +/- 1.0). Defaults to `message` but can be one of "ignore" - skip the test; "warning" - warn if test fails; "message" - print a message if test fails; "stop" - throw an error if test fails.

check.conv.hess
: character passed to lmerControl and glmerControl - checking the Hessian of the deviance function for convergence. Defaults to message but can be one of "ignore" - skip the test; "warning" - warn if test fails; "message" - print a message if test fails; "stop" - throw an error if test fails.

xtol_abs
: Defaults to 1e-6 stop on small change of parameter value. Only for method='mle', group.var=.... Default convergence tolerance for fitted (g)lmer models is reduced to the value provided here if default values did not fit. This value here is passed to the optCtrl argument of (g)lmer (see help of lme4::convergence()).

ftol_abs
: Defaults to 1e-6 stop on small change in deviance. Similar to xtol_abs.

trace.mblogit
: logical indicating if output should be produced for each iteration. Directly passed to trace argument in mclogit.control. Is independent of verbose.

catcov.mblogit
: Defaults to "free" meaning that there are no restrictions on the covariances of random effects between the logit equations. Set to "diagonal" if random effects pertinent to different categories are uncorrelated or "single" if random effect variances pertinent to all categories are identical.

epsilon
: Defaults to 1e-8. Positive convergence tolerance ϵ that is directly passed to the control argument of mclogit::mblogit as mclogit.control. Only for method='mle', group.var=....

seed
: a non-negative integer which sets the seed in set.seed(seed).

Details

Parallelization over all children is possible via the function foreach of the package **doParallel**. ncores=0 or ncores=1 use single threaded foreach. ncores=-1 uses all available cores but one.

Value

Named list according the provided arguments.

See Also

fit.control.

Other buildScoreCache: buildScoreCache()

Examples

```
ctrlmle <- abn::build.control(method = "mle",
                        ncores = 0,
                        max.irls = 100,
                        tol = 10^-11,
                        tolPwrss = 1e-7,
                        check.rankX = "message+drop.cols",
                        check.scaleX = "message+rescale",
                        check.conv.grad = "message",
                        check.conv.singular = "message",
                        check.conv.hess = "message",
                        xtol_abs = 1e-6,
```

```
                          ftol_abs = 1e-6,
                          trace.mblogit = FALSE,
                          catcov.mblogit = "free",
                          epsilon = 1e-6,
                          seed = 9062019L)
ctrlbayes <- abn::build.control(method = "bayes",
                             max.mode.error = 10,
                             mean = 0, prec = 0.001,
                             loggam.shape = 1,
                             loggam.inv.scale = 5e-05,
                             max.iters = 100,
                             epsabs = 1e-07,
                             error.verbose = FALSE,
                             epsabs.inner = 1e-06,
                             max.iters.inner = 100,
                             finite.step.size = 1e-07,
                             hessian.params = c(1e-04, 0.01),
                             max.iters.hessian = 10,
                             max.hessian.error = 0.5,
                             factor.brent = 100,
                             maxiters.hessian.brent = 100,
                             num.intervals.brent = 100,
                             tol = 10^-8,
                             seed = 9062019L)
```

`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 scores (AIC, BIC, log marginal likelihood). 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 = NULL,
verbose = FALSE,
...)

buildScoreCache.bayes(
  data.df = NULL,
  data.dists = NULL,
  group.var = NULL,
  cor.vars = NULL,
  dag.banned = NULL,
  dag.retained = NULL,
  max.parents = NULL,
  which.nodes = NULL,
  defn.res = NULL,
  dry.run = FALSE,
  centre = TRUE,
  force.method = NULL,
  mylist = NULL,
  grouped.vars = NULL,
  group.ids = NULL,
  control = build.control(method = "bayes"),
  verbose = FALSE
)

forLoopContent(
  row.num,
  mycache,
  data.dists,
  data.df.multi,
  adj.vars,
  data.df,
  data.df.lvl,
  group.var,
  group.ids,
  control,
  n,
  verbose
)

buildScoreCache.mle(
  data.df = NULL,
  data.dists = NULL,
  max.parents = NULL,
  adj.vars = NULL,
  cor.vars = NULL,
  dag.banned = NULL,
```

```
  dag.retained = NULL,
  which.nodes = NULL,
  centre = TRUE,
  defn.res = NULL,
  dry.run = FALSE,
  verbose = FALSE,
  force.method = NULL,
  group.var = NULL,
  grouped.vars = NULL,
  group.ids = NULL,
  control = build.control(method = "mle")
)
```

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	variable name for nodes to be fitted as variable intercept as in a mixed-effects model ("Bayes" and "mle") 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 ("bayes" only).
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 (only for method="bayes". A constant can be a single integer, a numeric vector of the length of variables with the same integer for all variable (e.g. c(2,2)) or a named list with all values being the same (e.g. list("A"=2, "B"=2)).
which.nodes	a vector giving the column indices of the variables to be included, if ignored all variables are included. This is used to subset data.df.
defn.res	an optional user-supplied list of child and parent combinations, see 'Details'.
centre	should the observations in each Gaussian node first be standardized 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 build.control for the names of the settable control values and their effect.
verbose	if TRUE then provides some additional output.
...	additional arguments passed for optimization.
force.method	"notset", "INLA" or "C". This is specified in buildScoreCache(control=list(max.mode.error=...))
mylist	result returned from check.valid.data.
grouped.vars	result returned from check.valid.groups.
group.ids	result returned from check.valid.groups.
row.num	number of child-node (mostly corresponds to child node index e.g. in dag).
mycache	prepared cache.
data.df.multi	extended data.df for one-hot-encoded multinomial variables.
data.df.lvl	copy of original data.df.
n	corresponds to nvars, number of variables in data.dists.

Details

The function computes a cache of scores based on possible restrictions (maximum complexity, retained and banned arcs). 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, creating a cache of scores. This cache of score could be used to select the optimal network in other function such as searchHeuristic or mostProbable. 'dag.banned' and 'dag.retained' specify which arcs are forced to be absent or present in the DAG, respectively. If provided as matrix, rows represent child nodes and columns their parents for elements with a value $=1$.

Two very different approaches are implemented: a Bayesian and frequentist approaches. They can be selected using the method argument.

If method="bayes"**::**

This function is used to calculate all individual node scores (log marginal likelihoods). 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 behavior can be overridden using control=list(max.mode.error=...). The default is max.mode.error=10, which means that the modes estimated from INLA output must be within 10\ Otherwise, the 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 more details, see fitAbn. 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 formats to that produced by an individual call to buildScoreCache, comprising of all nodes (in the same order as the columns in data.df) before sending it 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. 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 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 `searchHeuristic`).
- `dag.banned` the matrix encoding the banned arcs (for internal use only in other functions such as `searchHeuristic`).
- `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"`.

list

Functions

- `buildScoreCache.bayes()`: Fit a given DAG to data with method="bayes".
- `forLoopContent()`: Internal function called by `buildScoreCache.mle()`.
- `buildScoreCache.mle()`: Fit a given DAG to data with method="mle".

References

Kratzer, Gilles, Fraser Lewis, Arianna Comin, Marta Pittavino, and Reinhard Furrer. "Additive Bayesian Network Modeling with the R Package Abn." Journal of Statistical Software 105 (January 28, 2023): 1–41. https://doi.org/10.18637/jss.v105.i08.

Kratzer, G., Lewis, F.I., Comin, A., Pittavino, M., and Furrer, R. (2019). "Additive Bayesian Network Modelling with the R Package abn". arXiv:1911.09006.

Kratzer, G., and Furrer, R., (2018). "Information-Theoretic Scoring Rules to Learn Additive Bayesian Network Applied to Epidemiology". arXiv:1808.01126.

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: r-bayesian-networks.org.

See Also

`fitAbn`

Other buildScoreCache: `build.control()`

Other Bayes: `calc.node.inla.glmm()`, `calc.node.inla.glm()`, `fitAbn()`, `getmarginals()`

Examples

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

## Subset of the build-in dataset, see  ?ex0.dag.data
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 constraints
# 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 the cache of pre-computed scores accordingly to the structural constraints

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:
if(requireNamespace("INLA", quietly = TRUE)){
  res.inla <- buildScoreCache(data.df=mydat, data.dists=mydists,
                          dag.banned= ~b1|b2, dag.retained= ~g1|g2, max.parents=max.par,
                              control=list(max.mode.error=100))

  ## comparison - very similar
  difference <- res.c$mlik - res.inla$mlik
}

## Comparison Bayes with MLE (unconstrained):
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")
## of course different, but smame order:
plot(-res.mle$bic, res.abn$mlik)

  ###################################################################
  ## Example 2 - mle with several cores
  ###################################################################

  ## Many variables, few observations
  mydat <- ex0.dag.data
  mydists <- as.list(rep(c("binomial", "gaussian", "poisson"), each=10))
  names(mydists) <- names(mydat)

  # system.time( {
  # res.mle1 <- buildScoreCache(data.df=mydat, data.dists=mydists,
  #                           max.parents=2, method="mle", ncores=2) })
  # system.time( {
  # res.mle2 <- buildScoreCache(data.df=mydat, data.dists=mydists,
  #                           max.parents=2, method="mle") })


  ###################################################################
  ## 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)
```

`calc.node.inla.glm` *Fit a given regression using INLA*

Description

Internal wrapper to INLA and are called from `fitAbn.bayes` and `buildScoreCache.bayes`.

Usage

```
calc.node.inla.glm(
  child.loc,
  dag.m.loc,
  data.df.loc,
  data.dists.loc,
  ntrials.loc,
  exposure.loc,
  compute.fixed.loc,
  mean.intercept.loc,
  prec.intercept.loc,
  mean.loc,
  prec.loc,
  loggam.shape.loc,
  loggam.inv.scale.loc,
  verbose.loc
)
```

Arguments

`child.loc`	index of current child node.
`dag.m.loc`	dag as matrix.
`data.df.loc`	data df,
`data.dists.loc`	list of distributions.
`ntrials.loc`	`rep(1,dim(data.df)[1])`.
`exposure.loc`	`rep(1,dim(data.df)[1])`.

compute.fixed.loc
: TRUE.

mean.intercept.loc
: the prior mean for all the Gaussian additive terms for each node. INLA argument control.fixed=list(mean.intercept=...) and control.fixed=list(mean=...).

prec.intercept.loc
: the prior precision for all the Gaussian additive term for each node. INLA argument control.fixed=list(prec.intercept=...) and control.fixed=list(prec=...).

mean.loc
: the prior mean for all the Gaussian additive terms for each node. INLA argument control.fixed=list(mean.intercept=...) and control.fixed=list(mean=...). Same as mean.intercept.loc.

prec.loc
: the prior precision for all the Gaussian additive term for each node. INLA argument control.fixed=list(prec.intercept=...) and control.fixed=list(prec=...). Same as prec.intercept.loc.

loggam.shape.loc
: the shape parameter in the Gamma distribution prior for the precision in a Gaussian node. INLA argument control.family=list(hyper = list(prec = list(prior="loggamma",pa loggam.inv.scale))).

loggam.inv.scale.loc
: the inverse scale parameter in the Gamma distribution prior for the precision in a Gaussian node. INLA argument control.family=list(hyper = list(prec = list(prior="loggamma",param=c(loggam.shape, loggam.inv.scale)))).

verbose.loc
: FALSE to not print additional output.

Value

If INLA failed, FALSE or an error is returned. Otherwise, the direct output from INLA is returned.

See Also

Other Bayes: buildScoreCache(), calc.node.inla.glmm(), fitAbn(), getmarginals()

---

calc.node.inla.glmm *Fit a given regression using INLA*

Description

Internal wrapper to INLA and are called from fitAbn.bayes and buildScoreCache.bayes.

Usage

```
calc.node.inla.glmm(
  child.loc,
  dag.m.loc,
  data.df.loc,
  data.dists.loc,
```

```
  ntrials.loc,
  exposure.loc,
  compute.fixed.loc,
  mean.intercept.loc,
  prec.intercept.loc,
  mean.loc,
  prec.loc,
  loggam.shape.loc,
  loggam.inv.scale.loc,
  verbose.loc
)
```

Arguments

child.loc	index of current child node.
dag.m.loc	dag as matrix.
data.df.loc	data df,
data.dists.loc	list of distributions.
ntrials.loc	rep(1,dim(data.df)[1]).
exposure.loc	rep(1,dim(data.df)[1]).
compute.fixed.loc	TRUE.
mean.intercept.loc	the prior mean for all the Gaussian additive terms for each node. INLA argument control.fixed=list(mean.intercept=...) and control.fixed=list(mean=...).
prec.intercept.loc	the prior precision for all the Gaussian additive term for each node. INLA argument control.fixed=list(prec.intercept=...) and control.fixed=list(prec=...).
mean.loc	the prior mean for all the Gaussian additive terms for each node. INLA argument control.fixed=list(mean.intercept=...) and control.fixed=list(mean=...). Same as mean.intercept.loc.
prec.loc	the prior precision for all the Gaussian additive term for each node. INLA argument control.fixed=list(prec.intercept=...) and control.fixed=list(prec=...). Same as prec.intercept.loc.
loggam.shape.loc	the shape parameter in the Gamma distribution prior for the precision in a Gaussian node. INLA argument control.family=list(hyper = list(prec = list(prior="loggamma",pa loggam.inv.scale)))).
loggam.inv.scale.loc	the inverse scale parameter in the Gamma distribution prior for the precision in a Gaussian node. INLA argument control.family=list(hyper = list(prec = list(prior="loggamma",param=c(loggam.shape, loggam.inv.scale)))).
verbose.loc	FALSE to not print additional output.

Value

If INLA failed, FALSE or an error is returned. Otherwise, the direct output from INLA is returned.

See Also

Other Bayes: buildScoreCache(), calc.node.inla.glm(), fitAbn(), getmarginals()

Cfunctions *Documentation of C Functions*

Description

This is mainly to circumvent issues in R CMD check.

---

check.valid.buildControls
Simple check on the control parameters

Description

Simple check on the control parameters

Usage

```
check.valid.buildControls(control, method = "bayes", verbose = FALSE)
```

Arguments

control	list of control arguments with new parameters supplied to buildScoreCache or fitAbn.
method	"bayes" or "mle" strategy from argument method=... in buildScoreCache or fitAbn. Defaults to "bayes".
verbose	when TRUE additional information is printed. Defaults to FALSE.

Value

list with all control arguments with respect to the method but with new values.

check.valid.dag — *Set of simple commonsense validity checks on the directed acyclic graph definition matrix*

Description

Set of simple commonsense validity checks on the directed acyclic graph definition matrix

Usage

```
check.valid.dag(
  dag = NULL,
  data.df = NULL,
  is.ban.matrix = FALSE,
  group.var = NULL
)
```

Arguments

dag	Named square matrix or a formula statement specifying a directed acyclic graph. If NULL an empty network is assumed.
data.df	data frame with names corresponding to variable/node names.
is.ban.matrix	Diagonals and cycles are not checked for ban-matrices. Defaults to FALSE.
group.var	not yet implemented

Value

dag as named square matrix

check.valid.data — *Set of simple commonsense validity checks on the data.df and data.dists arguments*

Description

Set of simple commonsense validity checks on the data.df and data.dists arguments

Usage

```
check.valid.data(data.df = NULL, data.dists = NULL, group.var = NULL)
```

Arguments

data.df	data frame with names corresponding to variable/node names.
data.dists	list specifying each columns distribution type. Names correspond to column names and values must be one of "gaussian", "binomial", "poisson", "multinomial".
group.var	not yet implemented

Value

list of sums of each distribution types (abbreviated) as names.

---

check.valid.fitControls

Simple check on the control parameters

Description

Simple check on the control parameters

Usage

```
check.valid.fitControls(control, method = "bayes", verbose = FALSE)
```

Arguments

control	list of control arguments with new parameters supplied to buildScoreCache or fitAbn.
method	"bayes" or "mle" strategy from argument method=... in buildScoreCache or fitAbn. Defaults to "bayes".
verbose	when TRUE additional information is printed. Defaults to FALSE.

Value

list with all control arguments with respect to the method but with new values.

check.valid.groups *Simple check on the grouping variable*

Description

Simple check on the grouping variable

Usage

```
check.valid.groups(
  group.var = NULL,
  data.df = NULL,
  cor.vars = NULL,
  verbose = FALSE
)
```

Arguments

group.var	Name of grouping variable.
data.df	data frame of all variables including the variable specified in group.var as factor.
cor.vars	Name(s) of variables to which the grouping should be applied to.
verbose	when TRUE additional information is printed. Defaults to FALSE.

Value

list with data.df, indexes of variables to which the grouping should be applied to and the associated group for each observation as integer.

check.valid.parents *Set of simple checks on the given parent limits*

Description

Set of simple checks on the given parent limits

Usage

```
check.valid.parents(data.df = NULL, max.parents = NULL, group.var = NULL)
```

Arguments

data.df	data frame
max.parents	single integer for one overall max parent limit. A list with names corresponding to variable/column names of data.df and individual parent limits. NULL for no parent limit restriction(s).
group.var	not yet implemented

Value

numeric vector of max number of parents per variable

---

check.which.valid.nodes *Set of simple checks on the list given as parent limits*

Description

Set of simple checks on the list given as parent limits

Usage

```
check.which.valid.nodes(data.df = NULL, which.nodes = NULL, group.var = NULL)
```

Arguments

data.df	data frame
which.nodes	a vector giving the column indices of the variables to be included, if ignored all variables are included.
group.var	not yet implemented

Value

integer vector of column indexes of valid nodes in data.df

---

coef.abnFit *Print coefficients of objects of class* abnFit

Description

Print coefficients of objects of class abnFit

Usage

```
## S3 method for class 'abnFit'
coef(object, digits = 3L, verbose = TRUE, ...)
```

Arguments

object	Object of class abnFit
digits	number of digits of the results.
verbose	print additional output.
...	additional parameters. Not used at the moment.

compareDag *Compare two DAGs or EGs*

Description

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

Usage

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

Arguments

ref	a matrix or a formula statement (see details for format) defining the reference network structure, a directed acyclic graph (DAG). Note that row names 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 row names 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.
checkDAG	should the DAGs be tested for DAGs (default).

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 ref or test can be provided using a formula statement (similar to GLM input). 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.

To test for essential graphs (or graphs) in general, the test for DAG need to be switched off checkDAG=FALSE. The function compareEG() is a wrapper to compareDag(, checkDAG=FALSE).

Value

- 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.

References

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

Examples

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

colnames(test.m) <- rownames(test.m) <- colnames(ref.m) <- colnames(ref.m) <- c("a", "b", "c")

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

compareEG *Compare two DAGs or EGs*

Description

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

Usage

```
compareEG(ref, test)
```

Arguments

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

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 ref or test can be provided using a formula statement (similar to GLM input). 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.

To test for essential graphs (or graphs) in general, the test for DAG need to be switched off checkDAG=FALSE. The function compareEG() is a wrapper to compareDag(, checkDAG=FALSE).

Value

- 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.

References

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

Examples

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

colnames(test.m) <- rownames(test.m) <- colnames(ref.m) <- colnames(ref.m) <- c("a", "b", "c")

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

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

Description

This function discretizes a data frame of possibly continuous random variables through rules for discretization. The discretization algorithms are unsupervised and univariate. See details for the complete list of discretization rules (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 and multinomial variables must be declared as factors, others as a 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

fd Freedman Diaconis rule. IQR() stands for interquartile range. 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 Doane's rule. The number of bins is given by

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

This 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` Cencov's rule. The number of bins is given by:

$$n^{1/3}$$

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

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

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

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

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

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

$$1 + \log_2(n)$$

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

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

Value

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

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. It is also known that the bias will decrease as the sample size increases.

Value

Shannon's entropy estimate on natural logarithm scale.

References

Cover, Thomas M, and Joy A Thomas. (2012). "Elements of Information Theory". John Wiley & Sons.

See Also

`discretization`

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))
```

essentialGraph *Construct the essential graph*

Description

Constructs different versions of the essential graph from a given DAG. External function that computes essential graph of a dag Minimal PDAG: The only directed edges are those who participate in v-structure Completed PDAG: very directed edge corresponds to a compelled edge, and every undirected edge corresponds to a reversible edge

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, aka acyclic partially directed graph (PDAG). 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 + 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 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.

References

West, D. B. (2001). Introduction to Graph Theory. Vol. 2. Upper Saddle River: Prentice Hall. Chickering, D. M. (2013) A Transformational Characterization of Equivalent Bayesian Network Structures, arXiv:1302.4938.

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).

- b1binary, logit()=1
- b2binary, logit()=1
- b3binary, logit()=1
- b4binary, logit()=1
- b5binary, logit()=1
- b6binary, logit()=1
- b7binary, logit()=1
- b8binary, logit()=1
- b9binary, logit()=1
- b10binary, logit()=1
- g1gaussian, identity()=1
- g2gaussian, identity()=1
- g3gaussian, identity()=1
- g4gaussian, identity()=1
- g5gaussian, identity()=1
- g6gaussian, identity()=1
- g7gaussian, identity()=1

- g8gaussian, identity()=1
- g9gaussian, identity()=1
- g10gaussian, identity()=1
- p1poisson, log()=1
- p2poisson, log()=1
- p3poisson, log()=1
- p4poisson, log()=1
- p5poisson, log()=1
- p6poisson, log()=1
- p7poisson, log()=1
- p8poisson, log()=1
- p9poisson, log()=1
- p10poisson, 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).

- b1binary, logit()=1
- p1poisson, log()=1
- g1gaussian, identity()=1
- b2binary, logit()=1
- p2poisson, log()=1+b1+p1
- b3binary, logit()=1+b1+g1+b2
- g2gaussian, identify()=1+p1+g1+b2
- b4binary, logit()=1+g1+p2
- b5binary, logit()=1+g1+g2
- g3gaussian, 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).

- b1binary,logit()=1+g1+b2+b3+p3+b4+g4+b5
- g1gaussian,identity()=1
- p1poisson,log()=1+g6
- b2binary,logit()=1+p3+b4+p6
- g2gaussian,identify()=1+b2
- p2poisson,log()=1+b2
- b3binary,logit()=1+g1+g2+p2+g3+p3+g4
- g3gaussian,identify()=1+g1+p3+b4
- p3poisson,log()=1
- b4binary,logit()=1+g1+p3+p5
- g4gaussian,identify()=1+b4;
- p4poisson,log()=1+g1+b2+g2+b5
- b5binary,logit()=1+b2+g2+b3+p3+g4
- g5gaussian,identify()=1
- p5poisson,log()=1+g1+g5+b6+g6
- b6binary,logit()=1
- g6gaussian,identify()=1
- p6poisson,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,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,
  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,
  0,1,0,1,1,0,0,0,0,0,0,1,0,0,0,0,0,
  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

- p1count
- g1continuous
- g2continuous
- b1binary
- b2binary
- g3continuous
- g4continuous
- groupfactor,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

- b1binary
- b2binary
- groupfactor, defines sampling groups

expit *expit of proportions*

Description

See also the C implementation `?abn::expit_cpp()`.

Usage

```
expit(x)
```

Arguments

x	numeric with values between `[0,1]`.

Value

numeric vector of same length as `x`.

expit_cpp *expit function*

Description

transform `x` either via the logit, or expit.

Usage

```
expit_cpp(x)
```

Arguments

x	a numeric vector

family.abnFit *Print family of objects of class* abnFit

Description

Print family of objects of class abnFit

Usage

```
## S3 method for class 'abnFit'
family(object, ...)
```

Arguments

object	Object of class abnFit
...	additional parameters. Not used at the moment.

FCV *Dataset related to Feline calicivirus infection among cats in Switzerland.*

Description

The dataset is about the Feline calicivirus (FCV) infection among cats in Switzerland. FCV is a virus that occurs worldwide in domestic cats but also in exotic felids. FCV is a highly contagious virus that is the major cause of upper respiratory disease or cat flue that affects felids. This is a complex disease caused by different viral and bacterial pathogens, i.e., FCV, FHV-1, *Mycoplasma felis*, *Chlamydia felis* and *Bordetella bronchiseptica*. It can be aggravated by retrovirus infections such as FeLV and FIV. This composite dynamic makes it very interesting for a BN modeling approach. The data were collected between September 2012 and April 2013.

Usage

```
FCV
```

Format

An adapted data frame of the original dataset, which consists of 300 observations of 15 variables.

- FCVFeline Calici Virus status (0/1).
- FHV_1Feline Herpes Virus 1 status (0/1).
- C_felisC-felis and Chlamydia felis status (0/1).
- M_felisMycoplasma felis status (0/1).
- B_bronchisepticaB-bronchiseptica & Bordetella bronchispetica status (0/1).

- FeLVfeline leukosis virus status (0/1).
- FIVfeline immunodeficiency virus status (0/1).
- Gingivostomatitisgingivostomatitis complex status (0/1).
- URTDURTD complex (upper respiratory complex) (0/1).
- Vaccinatedvaccination status (0/1).
- Pedigreepedigree (0/1).
- Outdooroutdoor access (0/1).
- Sexsex and castrated status (M, MN, F, FS).
- GroupSizenumber of cats in the group (counts).
- Ageage in year (continuous)\.

References

Berger, A., Willi, B., Meli, M. L., Boretti, F. S., Hartnack, S., Dreyfus, A., ... and Hofmann-Lehmann, R. (2015). Feline calicivirus and other respiratory pathogens in cats with Feline calicivirus-related symptoms and in clinically healthy cats in Switzerland. BMC Veterinary Research, 11(1), 282.

`fit.control` *Control the iterations in* `fitAbn`

Description

Allow the user to set restrictions in the `fitAbn` for both the Bayesian and the MLE approach. Control function similar to `build.control`.

Usage

```
fit.control(
  method = "bayes",
  max.mode.error = 10,
  mean = 0,
  prec = 0.001,
  loggam.shape = 1,
  loggam.inv.scale = 5e-05,
  max.iters = 100,
  epsabs = 1e-07,
  error.verbose = FALSE,
  trace = 0L,
  epsabs.inner = 1e-06,
  max.iters.inner = 100,
  finite.step.size = 1e-07,
  hessian.params = c(1e-04, 0.01),
  max.iters.hessian = 10,
```

```
  max.hessian.error = 1e-04,
  factor.brent = 100,
  maxiters.hessian.brent = 10,
  num.intervals.brent = 100,
  min.pdf = 0.001,
  n.grid = 250,
  std.area = TRUE,
  marginal.quantiles = c(0.025, 0.25, 0.5, 0.75, 0.975),
  max.grid.iter = 1000,
  marginal.node = NULL,
  marginal.param = NULL,
  variate.vec = NULL,
  ncores = 1,
  max.irls = 100,
  tol = 1e-11,
  tolPwrss = 1e-07,
  check.rankX = "message+drop.cols",
  check.scaleX = "message+rescale",
  check.conv.grad = "message",
  check.conv.singular = "message",
  check.conv.hess = "message",
  xtol_abs = 1e-06,
  ftol_abs = 1e-06,
  trace.mblogit = FALSE,
  catcov.mblogit = "free",
  epsilon = 1e-06,
  seed = 9062019L
)
```

Arguments

`method`	a character that takes one of two values: "bayes" or "mle". Overrides `method` argument from `buildScoreCache`.
`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 always to be used, then `max.mode.error=100`, to force INLA never to be used `max.mod.error=0`. See also `fitAbn`.
`mean`	the prior mean for all the Gaussian additive terms for each node. INLA argument `control.fixed=list(mean.intercept=...)` and `control.fixed=list(mean=...)`.
`prec`	the prior precision ($\tau = \frac{1}{\sigma^2}$) for all the Gaussian additive term for each node. INLA argument `control.fixed=list(prec.intercept=...)` and `control.fixed=list(prec=...)`.
`loggam.shape`	the shape parameter in the Gamma distribution prior for the precision in a Gaussian node. INLA argument `control.family=list(hyper = list(prec = list(prior="loggamma",pa loggam.inv.scale))))`.
`loggam.inv.scale`	the inverse scale parameter in the Gamma distribution prior for the precision in a Gaussian node. INLA argument `control.family=list(hyper = list(prec = list(prior="loggamma",param=c(loggam.shape, loggam.inv.scale))))`.

`max.iters` total number of iterations allowed when estimating the modes in Laplace approximation. passed to `.Call("fit_single_node", ...)`.

`epsabs` absolute error when estimating the modes in Laplace approximation for models with no random effects. Passed to `.Call("fit_single_node", ...)`.

`error.verbose` logical, additional output in the case of errors occurring in the optimization. Passed to `.Call("fit_single_node", ...)`.

`trace` Non-negative integer. If positive, tracing information on the progress of the "L-BFGS-B" optimization is produced. Higher values may produce more tracing information. (There are six levels of tracing. To understand exactly what these do see the source code.). Passed to `.Call("fit_single_node", ...)`.

`epsabs.inner` absolute error in the maximization step in the (nested) Laplace approximation for each random effect term. Passed to `.Call("fit_single_node", ...)`.

`max.iters.inner`
total number of iterations in the maximization step in the nested Laplace approximation. Passed to `.Call("fit_single_node", ...)`.

`finite.step.size`
suggested step length used in finite difference estimation of the derivatives for the (outer) Laplace approximation when estimating modes. Passed to `.Call("fit_single_node", ...)`.

`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 initial guess, second entry absolute error. Passed to `.Call("fit_single_node", ...)`.

`max.iters.hessian`
integer, maximum number of iterations to use when determining an optimal finite difference approximation (Nelder-Mead). Passed to `.Call("fit_single_node", ...)`.

`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. Passed to `.Call("fit_single_node", ...)`.

`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 $h/factor.brent, h * factor.brent)$. Passed to `.Call("fit_single_node", ...)`.

`maxiters.hessian.brent`
maximum number of iterations allowed in the Brent-Dekker method. Passed to `.Call("fit_single_node", ...)`.

`num.intervals.brent`
the number of initial different bracket segments to try in the Brent-Dekker method. Passed to `.Call("fit_single_node", ...)`.

`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 standardized 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 the number of nodes.

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

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

`ncores` The number of cores to parallelize to, see 'Details'. If >0, the number of CPU cores to be used. -1 for all available -1 core. Only for `method="mle"`.

`max.irls` total number of iterations for estimating network scores using an Iterative Reweighed Least Square algorithm. Is this DEPRECATED?

`tol` real number giving the minimal tolerance expected to terminate the Iterative Reweighed Least Square algorithm to estimate network score. Passed to `irls_binomial_cpp_fast_br` and `irls_poisson_cpp_fast`.

`tolPwrss` numeric scalar passed to `glmerControl` - the tolerance for declaring convergence in the penalized iteratively weighted residual sum-of-squares step. Similar to `tol`.

`check.rankX` character passed to `lmerControl` and `glmerControl` - specifying if `rankMatrix(X)` should be compared with `ncol(X)` and if columns from the design matrix should possibly be dropped to ensure that it has full rank. Defaults to `message+drop.cols`.

`check.scaleX` character passed to `lmerControl` and `glmerControl` - check for problematic scaling of columns of fixed-effect model matrix, e.g. parameters measured on very different scales. Defaults to `message+rescale`.

`check.conv.grad` character passed to `lmerControl` and `glmerControl` - checking the gradient of the deviance function for convergence. Defaults to `message` but can be one of "ignore" - skip the test; "warning" - warn if test fails; "message" - print a message if test fails; "stop" - throw an error if test fails.

`check.conv.singular` character passed to `lmerControl` and `glmerControl` - checking for a singular fit, i.e. one where some parameters are on the boundary of the feasible space (for example, random effects variances equal to 0 or correlations between random effects equal to +/- 1.0). Defaults to `message` but can be one of "ignore" - skip the test; "warning" - warn if test fails; "message" - print a message if test fails; "stop" - throw an error if test fails.

`check.conv.hess` character passed to `lmerControl` and `glmerControl` - checking the Hessian of the deviance function for convergence. Defaults to `message` but can be one

	of "ignore" - skip the test; "warning" - warn if test fails; "message" - print a message if test fails; "stop" - throw an error if test fails.
xtol_abs	Defaults to 1e-6 stop on small change of parameter value. Only for method='mle', group.var=.... Default convergence tolerance for fitted (g)lmer models is reduced to the value provided here if default values did not fit. This value here is passed to the optCtrl argument of (g)lmer (see help of lme4::convergence()).
ftol_abs	Defaults to 1e-6 stop on small change in deviance. Similar to xtol_abs.
trace.mblogit	logical indicating if output should be produced for each iteration. Directly passed to trace argument in mclogit.control. Is independent of verbose.
catcov.mblogit	Defaults to "free" meaning that there are no restrictions on the covariances of random effects between the logit equations. Set to "diagonal" if random effects pertinent to different categories are uncorrelated or "single" if random effect variances pertinent to all categories are identical.
epsilon	Defaults to 1e-8. Positive convergence tolerance ϵ that is directly passed to the control argument of mclogit::mblogit as mclogit.control. Only for method='mle', group.var=....
seed	a non-negative integer which sets the seed in set.seed(seed).

Details

Parallelization over all children is possible via the function foreach of the package **doParallel**. ncores=0 or ncores=1 use single threaded foreach. ncores=-1 uses all available cores but one.

Value

Named list according the provided arguments.

See Also

build.control.

Other fitAbn: fitAbn()

Examples

```
ctrlmle <- abn::fit.control(method = "mle",
                      max.irls = 100,
                      tol = 10^-11,
                      tolPwrss = 1e-7,
                      xtol_abs = 1e-6,
                      ftol_abs = 1e-6,
                      epsilon = 1e-6,
                      ncores = 2,
                      seed = 9062019L)
ctrlbayes <- abn::fit.control(method = "bayes",
                        mean = 0,
                        prec = 0.001,
                        loggam.shape = 1,
                        loggam.inv.scale = 5e-05,
```

```
                    max.mode.error = 10,
                    max.iters = 100,
                    epsabs = 1e-07,
                    error.verbose = FALSE,
                    epsabs.inner = 1e-06,
                    max.iters.inner = 100,
                    finite.step.size = 1e-07,
                    hessian.params = c(1e-04, 0.01),
                    max.iters.hessian = 10,
                    max.hessian.error = 1e-04,
                    factor.brent = 100,
                    maxiters.hessian.brent = 10,
                    num.intervals.brent = 100,
                    min.pdf = 0.001,
                    n.grid = 100,
                    std.area = TRUE,
                    marginal.quantiles = c(0.025, 0.25, 0.5, 0.75, 0.975),
                    max.grid.iter = 1000,
                    marginal.node = NULL,
                    marginal.param = NULL,
                    variate.vec = NULL,
                    ncores = 1,
                    seed = 9062019L)
```

`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 modeling. 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 r-inla.org - this is not hosted on CRAN).

Usage

```
fitAbn(object = NULL,
       dag = NULL,
       data.df = NULL,
       data.dists = NULL,
       method = NULL,
       group.var = NULL,
       adj.vars = NULL,
       cor.vars = NULL,
       centre = TRUE,
       compute.fixed = FALSE,
       control = NULL,
       verbose = FALSE,
```

```
        debugging = FALSE,
        ...)

fitAbn.bayes(
  dag = NULL,
  data.df = NULL,
  data.dists = NULL,
  group.var = NULL,
  cor.vars = NULL,
  centre = TRUE,
  compute.fixed = FALSE,
  control = fit.control(method = "bayes"),
  mylist = NULL,
  grouped.vars = NULL,
  group.ids = NULL,
  force.method = NULL,
  verbose = FALSE
)

fitAbn.mle(
  dag = NULL,
  data.df = NULL,
  data.dists = NULL,
  group.var = NULL,
  grouped.vars = NULL,
  group.ids = NULL,
  adj.vars = NULL,
  cor.vars = NULL,
  centre = TRUE,
  control = fit.control(method = "mle"),
  verbose = FALSE,
  debugging = FALSE
)

regressionLoop(
  i = NULL,
  dag = NULL,
  data.df = NULL,
  data.df.multi = NULL,
  data.dists = NULL,
  group.var = NULL,
  grouped.vars = NULL,
  group.ids = NULL,
  control = NULL,
  nvars = NULL,
  nobs = NULL,
  dag.multi = NULL,
  verbose = NULL
```

)

Arguments

object	an object of class abnLearned produced by mostProbable, searchHeuristic or searchHillClimber.
dag	a matrix or a formula statement (see details) defining the network structure, a directed acyclic graph (DAG), see details for format. Note that column names and row names must be set up.
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	if NULL, takes method of object, otherwise "bayes" or "mle" for the method to 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 (method = 'bayes' only).
centre	should the observations in each Gaussian node first be standardised to mean zero and standard deviation one.
compute.fixed	a logical flag, set to TRUE for computation of marginal posterior distributions, see details.
control	a list of control parameters. See fit.control for the names of the settable control values and their effect.
verbose	if TRUE then provides some additional output, in particular the code used to call INLA, if applicable.
debugging	if TRUE and method = 'mle' this enables to step into the for-loop.
...	additional arguments passed for optimization.
mylist	result returned from check.valid.data.
grouped.vars	result returned from check.valid.groups. Column indexes of all variables which are affected from grouping effect.
group.ids	result returned from check.valid.groups. Vector of group allocation for each observation (row) in 'data.df'.
force.method	"notset", "INLA" or "C". This is specified in buildScoreCache(control=list(max.mode.error=...))
i	number of child-node (mostly corresponds to child node index e.g. in dag).
data.df.multi	extended data.df for one-hot-encoded multinomial variables.
nvars	number of variables in data.dists.
nobs	number of observations in data.df.
dag.multi	extended dag for one-hot-encoded multinomial variables.

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). Multinomial distributions are only supported with `method = "mle"` (see below). 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 behavior can be overridden using `control=list(max.mode.error=...)`. The default is `max.mode.error=10`, which means that the modes estimated from INLA output must be within 10\ Otherwise, the 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 swift and accurate, but in several 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", "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`.

If the data are grouped into correlated blocks - wherein 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 that contains the names of the nodes for which a mixed model should be used. This is not yet implemented with `method = 'mle'`. For example, in some problems, it may be appropriate for all variables (except `group.var`) in data.df to be parametrized 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`, 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` 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 must 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 the 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`. When estimating the log marginal likelihood in models with random effects (using internal code rather than INLA), an attempt is made to 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 minimized through choosing an optimal step size using gsl's Nelder-Mead optimization, and if this fails, (e.g., is larger than `max.hessian.error`) then the Brent-Dekker root bracketing method is used as a fallback. 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.

If the data are grouped into correlated blocks - wherein a standard regression context a mixed-effect 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 intercept). This is achieved by specifying parameter `group.var` (`cor.vars` as with `method = "bayes"` is not yet implemented with `method = "mle"`). The parameter `group.var` defines the group membership variable, which should be a factor indicating which observations belong to the same grouping. This corresponds to `"1|group.var"` in the formula notation of e.g. **lme4**.

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`.

In the network structure definition, `dag`, 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` 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 is 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 Generalized Linear Model (GLM) fails to estimate the maximum likelihood of the given model accurately. 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 interpreting or even displaying 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 with separate arguments for the Bayesian and MLE implementation. See `fit.control` for details.

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). Also, a list called `marginal.quantiles` is produced, giving the quantiles for each marginal posterior distribution.

list

Functions

- `fitAbn.bayes()`: Internal function called by `fitAbn`.
- `fitAbn.mle()`: Internal function called by `fitAbn`.
- `regressionLoop()`: Internal function called by `fitAbn.mle()`.

Author(s)

Fraser Iain Lewis and Gilles Kratzer.

References

Kratzer, G., Lewis, F.I., Comin, A., Pittavino, M. and Furrer, R. (2019). "Additive Bayesian Network Modelling with the R Package abn". arXiv preprint arXiv:1911.09006.

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

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: r-bayesian-networks.org.

See Also

`buildScoreCache`

Other fitAbn: `fit.control()`

Other Bayes: `buildScoreCache()`, `calc.node.inla.glmm()`, `calc.node.inla.glm()`, `getmarginals()`

Examples

```
## 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=mydag.empty, data.df=mydat, data.dists=mydists)

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

## fit  using the formula statement
#  including the creation of the graph of the DAG via Rgraphviz
myres <- fitAbn(dag=~b1|b2+b2|p4:g1+g1|p2+b3|g1+b4|b1+p4|g1,
                data.df=mydat, data.dists=mydists)
print(myres$mlik) ## a much weaker fit than full independence DAG

plotAbn(dag=myres$abnDag$dag, data.dists=mydists, fitted.values=myres$modes)

## Or equivalentelly using the formula statement, with plotting
## Now repeat but include some dependencies first
mydag <- mydag.empty
mydag["b1","b2"] <- 1 # b1<-b2 and so on
mydag["b2","p4"] <- mydag["b2","g1"] <- mydag["g1","p2"] <- 1
mydag["b3","g1"] <- mydag["b4","b1"] <- mydag["p4","g1"] <- 1
myresAlt <- fitAbn(dag=mydag, data.df=mydat, data.dists=mydists)
plot(myresAlt)

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

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

## Print the output for mle first then for Bayes:
print(myres.mle)
print(myres)


## Not run:
  ## 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 50 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=mydag, data.df=mydat, data.dists=mydists,
                  compute.fixed=TRUE,
                  control=list(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=mydag, data.df=mydat, data.dists=mydists,
                     compute.fixed=TRUE,
                     control=list(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")

## An elementary mixed model example using built-in data specify DAG,
## only two variables using a 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=~b1|b2, data.df=ex3.dag.data[,c(1,2,14)], data.dists=mydists,
                  group.var="group", cor.vars=c("b1","b2"),
                  control=list(max.mode.error=0))
print(myres.c) ## show all the output

## compare mode for node b1 with glmer(), lme4::glmer is automatically attached.

## 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=~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=~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=~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,
#             control=list(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")

## End(Not run)
```

---

get.var.types *Create ordered vector with integers denoting the distribution*

Description

gaussian = 1, binomial = 2, poisson = 3, multinomial = 4

Usage

```
get.var.types(data.dists = NULL)
```

Arguments

data.dists	list specifying each columns distribution type. Names correspond to column names and values must be one of "gaussian", "binomial", "poisson", "multinomial".

Value

numeric encoding of distribution corresponding to its list element number in `data.dists`.

---

getmarginals *Internal function called by* `fitAbn.bayes`.

Description

Function for computing marginal posterior densities using C and is called from fit.dag() Only to be called internally.

Usage

```
getmarginals(
  res.list,
  data.df,
  dag.m,
  var.types,
  max.parents,
  mean,
  prec,
  loggam.shape,
  loggam.inv.scale,
  max.iters,
  epsabs,
  verbose,
  error.verbose,
  trace,
  grouped.vars,
  group.ids,
  epsabs.inner,
  max.iters.inner,
  finite.step.size,
```

```
  hessian.params,
  max.iters.hessian,
  min.pdf,
  marginal.node,
  marginal.param,
  variate.vec,
  n.grid,
  INLA.marginals,
  iter.max,
  max.hessian.error,
  factor.brent,
  maxiters.hessian.brent,
  num.intervals.brent
)
```

Arguments

res.list	rest of arguments as for call to C fitabn
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.
dag.m	adjacency matrix
var.types	distributions in terms of a numeric code
max.parents	max number of parents over all nodes in dag (different from other max.parents definitions).
mean	the prior mean for all the Gaussian additive terms for each node. INLA argument control.fixed=list(mean.intercept=...) and control.fixed=list(mean=...).
prec	the prior precision ($\tau = \frac{1}{\sigma^2}$) for all the Gaussian additive term for each node. INLA argument control.fixed=list(prec.intercept=...) and control.fixed=list(prec=...).
loggam.shape	the shape parameter in the Gamma distribution prior for the precision in a Gaussian node. INLA argument control.family=list(hyper = list(prec = list(prior="loggamma",pa loggam.inv.scale)))).
loggam.inv.scale	the inverse scale parameter in the Gamma distribution prior for the precision in a Gaussian node. INLA argument control.family=list(hyper = list(prec = list(prior="loggamma",param=c(loggam.shape, loggam.inv.scale)))).
max.iters	total number of iterations allowed when estimating the modes in Laplace approximation. passed to .Call("fit_single_node", ...).
epsabs	absolute error when estimating the modes in Laplace approximation for models with no random effects. Passed to .Call("fit_single_node", ...).
verbose	if TRUE then provides some additional output, in particular the code used to call INLA, if applicable.
error.verbose	logical, additional output in the case of errors occurring in the optimization. Passed to .Call("fit_single_node", ...).
trace	Non-negative integer. If positive, tracing information on the progress of the "L-BFGS-B" optimization is produced. Higher values may produce more tracing

information. (There are six levels of tracing. To understand exactly what these do see the source code.). Passed to `.Call("fit_single_node", ...)`.

`grouped.vars` result returned from `check.valid.groups`. Column indexes of all variables which are affected from grouping effect.

`group.ids` result returned from `check.valid.groups`. Vector of group allocation for each observation (row) in 'data.df'.

`epsabs.inner` absolute error in the maximization step in the (nested) Laplace approximation for each random effect term. Passed to `.Call("fit_single_node", ...)`.

`max.iters.inner`
total number of iterations in the maximization step in the nested Laplace approximation. Passed to `.Call("fit_single_node", ...)`.

`finite.step.size`
suggested step length used in finite difference estimation of the derivatives for the (outer) Laplace approximation when estimating modes. Passed to `.Call("fit_single_node", ...)`.

`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 initial guess, second entry absolute error. Passed to `.Call("fit_single_node", ...)`.

`max.iters.hessian`
integer, maximum number of iterations to use when determining an optimal finite difference approximation (Nelder-Mead). Passed to `.Call("fit_single_node", ...)`.

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

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

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

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

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

`INLA.marginals` vector - TRUE if INLA used false otherwise

`iter.max` same as `max.iters` in `fit.control`. Total number of iterations allowed when estimating the modes in Laplace approximation. Passed to .Call("fit_single_node", ...).

`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. Passed to `.Call("fit_single_node", ...)`.

`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 $h/factor.brent, h * factor.brent$). Passed to `.Call("fit_single_node", ...)`.

maxiters.hessian.brent
: maximum number of iterations allowed in the Brent-Dekker method. Passed to `.Call("fit_single_node", ...)`.

num.intervals.brent
: the number of initial different bracket segments to try in the Brent-Dekker method. Passed to `.Call("fit_single_node", ...)`.

See Also

Other Bayes: `buildScoreCache()`, `calc.node.inla.glmm()`, `calc.node.inla.glm()`, `fitAbn()`

`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 neighborhood average set size, the parent average set size and children average set size.

Usage

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

Arguments

`object`	an object of class `abnLearned`, `abnFit`. Alternatively, a matrix or a formula statement defining the network structure, a directed acyclic graph (DAG). Note that row names must be set up 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 neighborhood average set size, the parent average set size, and the children's average set size.

The dag 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 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 neighborhood average set size, the parent average set size and children average set size.

References

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

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)
plot(createAbnDag(dag))
```

---

linkStrength *Returns the strengths of the edge connections in a Bayesian Network learned from observational data*

Description

A flexible implementation of multiple proxy for strength measures useful for visualizing the edge connections in a Bayesian Network learned from 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 standard 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 plug-in 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) = H(Y|Z)-H(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 named matrix with the requested metric.

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).

Examples

```
# Gaussian
N <- 1000
mydists <- list(a="gaussian",
                b="gaussian",
                c="gaussian")
a <- rnorm(n = N, mean = 0, sd = 1)
b <- 1 + 2*rnorm(n = N, mean = 5, sd = 1)
c <- 2 + 1*a + 2*b + rnorm(n = N, mean = 2, sd = 1)
mydf <- data.frame("a" = a,
                   "b" = b,
                   "c" = c)
mycache.mle <- buildScoreCache(data.df = mydf,
                               data.dists = mydists,
                               method = "mle",
                               max.parents = 2)
mydag.mp <- mostProbable(score.cache = mycache.mle, verbose = FALSE)
```

```
linkstr <- linkStrength(dag = mydag.mp$dag,
                        data.df = mydf,
                        data.dists = mydists,
                        method = "ls",
                        discretization.method = "sturges")
```

`logit` *Logit of proportions*

Description

See also the C implementation `?abn::logit_cpp()`.

Usage

```
logit(x)
```

Arguments

`x`	numeric with values between `[0,1]`.

Value

numeric vector of same length as `x`.

`logit_cpp` *logit functions*

Description

transform `x` either via the logit, or expit.

Usage

```
logit_cpp(x)
```

Arguments

`x`	a numeric vector

logLik.abnFit *Print logLik of objects of class* abnFit

Description

Print logLik of objects of class abnFit

Usage

```
## S3 method for class 'abnFit'
logLik(object, digits = 3L, verbose = TRUE, ...)
```

Arguments

object	Object of class abnFit
digits	number of digits of the results.
verbose	print additional output.
...	additional parameters. Not used at the moment.

mb *Compute the Markov blanket*

Description

This function computes the Markov blanket of a set of nodes 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 returns 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 + 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 separtor between either children or parents, | separates children (left side) and parents (right side), + separates terms, . replaces all the variables in name.

Value

character vector of node names from the Markov blanket.

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")
mb(dag, node = c("b","e"))

mb(~a|b:c:e+b|c:d:f+e|f, node = "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.

References

Cover, Thomas M, and Joy A Thomas. (2012). "Elements of Information Theory". John Wiley & Sons.

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, score="bic", prior.choice=1, verbose=TRUE, ...)
```

Arguments

score.cache	object of class abnCache typically outputted by from buildScoreCache().
score	which score should be used to score the network. Possible choices are aic, bic, mdl, mlik.
prior.choice	an integer, 1 or 2, where 1 is a uniform structural prior and 2 uses a weighted prior, see details
verbose	if TRUE then provides some additional output.
...	further arguments passed to or from other methods.

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-modeling 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 or the files provided in the package directories inst/bootstrapping_example and inst/old_vignette for details.

The parameter prior.choice determines the prior used within each 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 favors 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., searchHillclimber or searchHeuristic.

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.

References

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

Examples

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

## This data comes with `abn` see ?ex1.dag.data
```

```
mydat <- ex1.dag.data[1:5000, c(1:7,10)]

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

## Parent limits, for speed purposes quite specific here:
max.par <- list("b1"=0,"p1"=0,"g1"=1,"b2"=1,"p2"=2,"b3"=3,"g2"=3,"g3"=2)
## Now build cache (no constraints in ban nor retain)
mycache <- buildScoreCache(data.df=mydat, data.dists=mydists, max.parents=max.par)

## Find the globally best DAG:
mp.dag <- mostProbable(score.cache=mycache)
myres <- fitAbn(object=mp.dag,create.graph=TRUE)
myres$mlik
plot(myres) # plot the best model

## last line is essentially equivalent to:
# plotAbn(dag=mp.dag$dag, data.dists=mydists, fitted.values=myres$modes)

## Fit the known true DAG (up to variables 'b4' and 'b5'):
true.dag <- matrix(data=0, ncol=8, nrow=8)
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["g3",c("g1","b2")] <- 1

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

  ###################################################################
  ## Example 2 - 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 and requires INLA:
  # 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)

  ## and 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)
```

nobs.abnFit *Print number of observations of objects of class* abnFit

Description

Print number of observations of objects of class abnFit

Usage

```
## S3 method for class 'abnFit'
nobs(object, ...)
```

Arguments

object	Object of class abnFit
...	additional parameters. Not used at the moment.

odds *Probability to odds*

Description

Probability to odds

Usage

```
odds(x)
```

Arguments

x	numeric vector of probabilities with values between [0,1].

Value

numeric vector of same length as x.

or — *Odds Ratio from a matrix*

Description

Compute the odds ratio from a contingency table or a matrix.

Usage

```
or(x)
```

Arguments

x	a 2x2 table or matrix.

Value

A real value.

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 grouping 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.

Usage

```
pigs.vienna
```

Format

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

- PCBinary.
- PTBinary.
- MSBinary.
- HSBinary.
- TAILBinary.
- AbscessBinary.

- PyaemiaBinary.
- EPcatBinary.
- PDcatBinary.
- plbinaryBinary.
- batchGroup variable, corresponding to the 500 different pig producers

Details

This dataset was used in an older version of the vignette. See also the files provided in the package directories `inst/bootstrapping_example` and `inst/old_vignette` give a detailed analysis of the dataset and provide more details for a bootstrapping example thereof.

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.

---

plot.abnDag *Plots DAG from an object of class* abnDag

Description

Plots DAG from an object of class abnDag

Usage

```
## S3 method for class 'abnDag'
plot(x, new = TRUE, ...)
```

Arguments

x	Object of class abnDag
new	defaults to TRUE for opening a new plot.
...	additional parameters. Not used at the moment.

Value

Rgraphviz::plot

Examples

```
mydag <- createAbnDag(dag = ~a+b|a, data.df = data.frame("a"=1, "b"=1))
plot(mydag)
```

plot.abnFit *Plot objects of class* abnFit

Description

Plot objects of class abnFit

Usage

```
## S3 method for class 'abnFit'
plot(x, which = "abnFit", ...)
```

Arguments

x	Object of class abnFit
which	defaults to "abnFit".
...	additional parameters. Not used at the moment.

plot.abnHeuristic *Plot objects of class* abnHeuristic

Description

Plot objects of class abnHeuristic

Usage

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

Arguments

x	Object of class abnHeuristic
...	additional parameters. Not used at the moment.

plot.abnHillClimber *Plot objects of class* abnHillClimber

Description

Plot objects of class abnHillClimber

Usage

```
## S3 method for class 'abnHillClimber'
plot(x, new = TRUE, ...)
```

Arguments

x	Object of class abnHillClimber
new	defaults to TRUE for opening a new plot.
...	additional parameters. Not used at the moment.

plot.abnMostprobable *Plot objects of class* abnMostprobable

Description

Plot objects of class abnMostprobable

Usage

```
## S3 method for class 'abnMostprobable'
plot(x, new = TRUE, ...)
```

Arguments

x	Object of class abnMostprobable
new	defaults to TRUE for opening a new plot.
...	additional parameters. Not used at the moment.

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, data.dists=NULL, markov.blanket.node=NULL, fitted.values=NULL,
          digits=2, edge.strength=NULL, edge.strength.lwd=5, edge.direction="pc",
               edge.color="black", edge.linetype="solid", edge.arrowsize=0.6,
               edge.fontsize=node.fontsize, node.fontsize=12,
               node.fillcolor=c("lightblue", "brown3", "chartreuse3"),
               node.fillcolor.list=NULL,
               node.shape=c("circle", "box", "ellipse", "diamond"),
               plot=TRUE , ...)
```

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.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 its Markov blanket.
fitted.values	modes or coefficents outputted from fitAbn.
digits	number of digits to display the fitted.values.
edge.strength	a named matrix containing evaluations of edge strength which will change the arcs width (could be Mutual information, p-values, number of bootstrap retrieve samples or the outcome of the linkStrength).
edge.strength.lwd	maximum line width for edge.strength.
edge.direction	character giving the direction in which arcs should be plotted, pc (parent to child) or cp (child to parent) or undirected.
edge.color	the colour of the edge.
edge.linetype	the linetype of the edge. Defaults to "solid". Valid values are the same as for the R's base graphic parameter lty.
edge.arrowsize	the thickness of the arrows. Not relevant if arc.strength is provided.
edge.fontsize	the font size of the arcs fitted values.
node.fontsize	the font size of the nodes names.
node.fillcolor	the colour of the node. Second and third element is used for the Markov blanket and node of the Markov blanket.

`node.fillcolor.list`
: the list of node that should be coloured.

`node.shape`
: the shape of the nodes according the Gaussian, binomial, Poisson and multinomial distributions.

`plot`
: logical variable, if set to `TRUE` then the graph is plotted.

`...`
: arguments passed to the plotting function.

Details

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

The dag can be provided using a formula statement (similar to glm). A typical formula is `~ node1|parent1:parent2 + node2:node3|parent3`.

The construction is based on the **graph** package. Properties of the graph can be changend after the construction, see 'Examples'.

Value

A rendered graph, if `plot=TRUE`. The `graphAM` object is returned invisibly.

See Also

`graphAM-class`, `edgeRenderInfo`

Examples

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

#Define a matrix formulation
edge.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(edge.strength) <- rownames(edge.strength) <- names(dist)

## Plot form a matrix
plotAbn(dag = edge.strength, data.dists = dist)

## Edge strength
plotAbn(dag = ~a|b:c:d:e+b|c:d:f+c|d:e+d|e+e|f, data.dists = dist, edge.strength = edge.strength)

## Plot from a formula for a different DAG!
plotAbn(dag = ~a|b:c:e+b|c:d:f+e|f, data.dists = dist)

## Markov blanket
```

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

## Change col for all edges
tmp <- plotAbn(dag = ~a|b:c:e+b|c:d:f+e|f, data.dists = dist, plot=FALSE)
graph::edgeRenderInfo(tmp) <- list(col="blue")
Rgraphviz::renderGraph(tmp)

## Change lty for individual ones. Named vector is necessary
tmp <- plotAbn(dag = ~a|b:c:e+b|c:d:f+e|f, data.dists = dist, plot=FALSE)
edgelty <- rep(c("solid","dotted"), c(6,1))
names(edgelty) <- names( graph::edgeRenderInfo(tmp, "col"))
graph::edgeRenderInfo(tmp) <- list(lty=edgelty)
Rgraphviz::renderGraph(tmp)
```

`print.abnCache` *Print objects of class* abnCache

Description

Print objects of class abnCache

Usage

```
## S3 method for class 'abnCache'
print(x, digits = 3, ...)
```

Arguments

x	Object of class abnCache
digits	number of digits of the results.
...	additional parameters. Not used at the moment.

Value

summary statement of the class of abnCache.

Examples

```
## Subset of the build-in dataset, see  ?ex0.dag.data
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 constraints
# 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 the cache of pre-computed scores accordingly to the structural constraints

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

---

print.abnDag *Print objects of class* abnDag

Description

Print objects of class abnDag

Usage

```
## S3 method for class 'abnDag'
print(x, digits = 3L, ...)
```

Arguments

x	Object of class abnDag
digits	number of digits of the adjacency matrix.
...	additional parameters. Not used at the moment.

Value

outputs adjacency matrix and statement of the class of x.

Examples

```
mydag <- createAbnDag(dag = ~a+b|a, data.df = data.frame("a"=1, "b"=1))
print(mydag)
```

print.abnFit *Print objects of class* abnFit

Description

Print objects of class abnFit

Usage

```
## S3 method for class 'abnFit'
print(x, digits = 3L, ...)
```

Arguments

x	Object of class abnFit
digits	number of digits of the results.
...	additional parameters. Not used at the moment.

print.abnHeuristic *Print objects of class* abnHeuristic

Description

Print objects of class abnHeuristic

Usage

```
## S3 method for class 'abnHeuristic'
print(x, digits = 2L, ...)
```

Arguments

x	Object of class abnHeuristic
digits	number of digits of the results.
...	additional parameters. Not used at the moment.

print.abnHillClimber *Print objects of class* abnHillClimber

Description

Print objects of class abnHillClimber

Usage

```
## S3 method for class 'abnHillClimber'
print(x, digits = 3L, ...)
```

Arguments

x	Object of class abnHillClimber
digits	number of digits of the results.
...	additional parameters. Not used at the moment.

print.abnMostprobable *Print objects of class* abnMostprobable

Description

Print objects of class abnMostprobable

Usage

```
## S3 method for class 'abnMostprobable'
print(x, digits = 3L, ...)
```

Arguments

x	Object of class abnMostprobable
digits	number of digits of the results.
...	additional parameters. Not used at the moment.

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

Description

This function computes the score's contribution of each observation to the total network score.

Usage

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

Arguments

object	an object of class 'abnLearned' produced by mostProbable, searchHeuristic or searchHillClimber.
dag	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 observation 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 function is an attempt to perform diagnostic for an ABN analysis.

Value

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

Examples

```
## Not run:
## 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)

## End(Not run)
```

---

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

Arguments

score.cache	output from buildScoreCache().
score	which score should be used to score the network. Possible choices are aic, bic, mdl, mlik.
num.searches	a positive integer giving the number of different search to run, see details.
seed	a non-negative integer which sets the seed.
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.
max.steps	a constant giving the number of search steps per search, see details.
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.
verbose	if TRUE then provides some additional output.
...	further arguments passed to or from other methods.

Details

This function is a flexible implementation of multiple greedy heuristic algorithms, particularly well adapted to the abn framework. It targets multi-random restarts heuristic algorithms. The user can select the num.searches and the maximum number of steps within by max.steps. The optimization algorithm within each search is relatively rudimentary.

The function searchHeuristic is different from the searchHillClimber in the sense that this function is fully written in R, whereas the searchHillClimber is written in C and thus expected to be faster. The function searchHillClimber at each hill-climbing step consider every information from the pre-computed scores cache while the function searchHeuristic performs a local stepwise optimization. This function chooses a structural move (or edge move) and compute the score's change. On this point, it is closer to the MCMCMC algorithm from Madigan and York (1995) and Giudici and Castelo (2003) with a single edge move.

If the user select random, then a random valid DAG is selected. The routine used favourise low density structure. The function implements three algorithm selected with the parameter algo: hc, tabu or sa.

If algo=hc: The Hill-climber algorithm (hc) is a single move algorithm. At each Hill-climbing step within a search an arc is attempted to be added. The new score is computed and compared to the previous network's score.

If algo=tabu: The same algorithm is as with hc is used, but a list of banned moves is computed. The parameter tabu.memory controls the length of the tabu list. The idea is that the classical Hill-climber algorithm is inefficient when it should cross low probability regions to unblock from a local maximum and reaching a higher score peak. By forcing the algorithm to choose some not already used moves, this will force the algorithm to escape the local maximum.

If algo=sa: This variant of the heuristic search algorithm is based on simulated annealing described by Metropolis et al. (1953). Some accepted moves could result in a decrease of the network score. The acceptance rate can be monitored with the parameter temperature.

Value

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

- dagsa list of DAGs
- scoresa vector giving the network score for the locally optimal network for each search
- detailed.scorea vector giving the evolution of the network score for the all the random restarts
- scorea number giving the network score for the locally optimal network
- score.cachethe pre-computed cache of scores
- num.searchesa numeric giving the number of random restart
- max.stepsa numeric giving the maximal number of optimization steps within each search
- algorithma character for indicating the algorithm used

References

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

(1995) "Bayesian graphical models for discrete data". International Statistical Review, 63:215232. Giudici, P. and Castelo, R. (2003). "Improving Markov chain Monte Carlo model search for data mining". Machine Learning, 50:127158. Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., & Teller, E. (1953). "Equation of state calculations by fast computing machines". The journal of chemical physics, 21(6), 1087-1092.

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 10 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, score = "mlik", num.searches = 1, seed = 42,
                      start.dag = NULL, support.threshold = 0.5, timing.on = TRUE,
                           dag.retained = NULL, verbose = FALSE, ...)
```

Arguments

score.cache	output from buildScoreCache().
score	character giving which network score should be used to select the structure. Currently 'mlik' only.
num.searches	number of times to run the search.
seed	non-negative integer which sets the seed in the GSL random number generator.
start.dag	a DAG given as a matrix, see details for format, which can be used to provide a starting point for the structural search explicitly.
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.
timing.on	extra output in terms of duration computation.
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.
verbose	extra output.
...	further arguments passed to or from other methods.

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 than 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 visualization; 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.scorea vector giving network score for initial network from which the search commenced
- final.scorea vector giving the network score for the locally optimal network
- init.daglist of matrices, initial DAGs
- final.daglist of matrices, locally optimal DAGs
- consensusa matrix holding a binary graph, not necessary a DAG!
- support.thresholdpercentage supported used to create consensus matrix

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).

`simulateAbn` *Simulate data from a fitted additive Bayesian network.*

Description

Simulate data from a fitted additive Bayesian network.

Usage

```
simulateAbn(
  object = NULL,
  run.simulation = TRUE,
  bugsfile = NULL,
  n.chains = 10L,
  n.adapt = 1000L,
  n.thin = 100L,
  n.iter = 10000L,
  seed = 42L,
  verbose = FALSE,
  debug = FALSE
)
```

Arguments

object	of type abnFit.
run.simulation	call JAGS to simulate data (default is TRUE).
bugsfile	A path to a valid file or NULL (default) to delete the bugs file after simulation.
n.chains	number of parallel chains for the model.
n.adapt	number of iteration for adaptation. If n.adapt is set to zero, then no adaptation takes place.
n.thin	thinning interval for monitors.
n.iter	number of iteration to monitor.
seed	by default set to 42.
verbose	if TRUE prints additional output
debug	if TRUE prints bug file content to stdout and does not run simulations.

Value

data.frame

See Also

makebugs

Examples

```
df <- FCV[, c(12:15)]
mydists <- list(Outdoor="binomial",
                Sex="multinomial",
                GroupSize="poisson",
                Age="gaussian")

## buildScoreCache -> mostProbable() -> fitAbn()
suppressWarnings({
  mycache.mle <- buildScoreCache(data.df = df, data.dists = mydists, method = "mle",
                                 adj.vars = NULL, cor.vars = NULL,
                                 dag.banned = NULL, dag.retained = NULL,
                                 max.parents = 1,
                                 which.nodes = NULL, defn.res = NULL)
}) # ignore non-convergence warnings
mp.dag.mle <- mostProbable(score.cache = mycache.mle, verbose = FALSE)
myres.mle <- fitAbn(object = mp.dag.mle, method = "mle")

myres.sim <- simulateAbn(object = myres.mle,
                             run.simulation = TRUE,
                             bugsfile = NULL,
                             verbose = FALSE)
str(myres.sim)
prop.table(table(myres.sim$Outdoor))
prop.table(table(df$Outdoor))
```

simulateDag *Simulate a DAG with with arbitrary arcs density*

Description

Provided with node names, returns an abnDAG. Arc density refers to the chance of a node being connected to the node before it.

Usage

```
simulateDag(node.name, data.dists = NULL, edge.density = 0.5, verbose = FALSE)
```

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 corresponding to the node.name specifying the distribution for each node. If not provided arbitrary distributions are assigned to the nodes.
edge.density	number in [0,1] specifying the edge probability in the dag.
verbose	print more information on the run.

Details

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

Value

object of class abnDag consisting of a named matrix, a named list giving the distribution for each node and an empty element for the data.

Examples

```
simdag <- simulateDag(node.name = c("a", "b", "c", "d"),
                      edge.density = 0.5,
                      data.dists = list(a = "gaussian",
                                        b = "binomial",
                                        c = "poisson",
                                        d = "multinomial"))

## 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, edge.density = 0.8)
plot(out)
```

skewness *Computes skewness of a distribution*

Description

Computes skewness of a distribution

Usage

```
skewness(x)
```

Arguments

x	a numeric vector

Value

integer

summary.abnDag *Prints summary statistics from an object of class* abnDag

Description

Prints summary statistics from an object of class abnDag

Usage

```
## S3 method for class 'abnDag'
summary(object, ...)
```

Arguments

object	an object of class abnLearned, abnFit. Alternatively, a matrix or a formula statement defining the network structure, a directed acyclic graph (DAG). Note that row names must be set up or given in node.names.
...	additional parameters. Not used at the moment.

Examples

```
mydag <- createAbnDag(dag = ~a+b|a, data.df = data.frame("a"=1, "b"=1))
summary(mydag)
```

summary.abnFit *Print summary of objects of class* abnFit

Description

Print summary of objects of class abnFit

Usage

```
## S3 method for class 'abnFit'
summary(object, digits = 3L, ...)
```

Arguments

object	Object of class abnFit
digits	number of digits of the results.
...	additional parameters. Not used at the moment.

summary.abnMostprobable *Print summary of objects of class* abnMostprobable

Description

Print summary of objects of class abnMostprobable

Usage

```
## S3 method for class 'abnMostprobable'
summary(object, ...)
```

Arguments

object	Object of class abnMostprobable
...	additional parameters. Not used at the moment.

`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,
                  data.df=NULL,
                  data.dists=NULL,
                  group.var=NULL,
                  outfile=NULL,
                  directed=TRUE,
                  verbose=FALSE)
```

Arguments

`dag`	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.
`verbose`	if TRUE more output is printed. If TRUE and 'outfile=NULL' the '.dot' file is printed to console.

Details

Graphviz (https://www.graphviz.org) is a 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. The R package, `Rgraphviz` (available through the Bioconductor project https://www.bioconductor.org/) interfaces R and the 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

Marta Pittavino

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
  outfile <- paste(tempdir(), "graph.dot", sep="/")
  toGraphviz(dag=mydag, data.df=mydat, data.dists=mydists, outfile=outfile)
  ## and then process using graphviz tools e.g. on linux
  # system(paste( "dot -Tpdf -o graph.pdf", outfile))
  # 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
  outfile <- paste0(tempdir(), "/graph.dot")
  toGraphviz(dag=mydag, data.df=ex3.dag.data[,c(1,2,14)], data.dists=mydists,
             group.var="group",
             outfile=outfile, directed=FALSE)
  ## and then process using graphviz tools e.g. on linux:
```

```
  # pdffile <- paste0(tempdir(), "/graph.pdf")"
  # system(paste("dot -Tpdf -o ", pdffile, outfile))
  # system(paste("evince ", pdffile, " &")   ## or some other viewer

## End(Not run)
```

---

validate_abnDag *Check for valid DAG of class* abnDag

Description

Beside some basic checks, this function also checks for square matrix with no undirected cycles (trivial cycles) and for no undirected cycles.

Usage

```
validate_abnDag(dag, data.df = NULL, returnDag = TRUE, ...)
```

Arguments

dag	dag is either a formula, a matrix or an object of class 'abnDag'
data.df	data frame
returnDag	if TRUE (default) returns DAG as matrix.
...	additional arguments.

Details

Similar to check.valid.dag().

Value

Either TRUE/FALSE or DAG as matrix.

---

validate_dists *Check for valid distribution*

Description

The distribution names must match inla() family=''. Similar to get.var.types(), mainly different in output.

Usage

```
validate_dists(data.dists, returnDists = TRUE, ...)
```

Arguments

data.dists	list of variable distributions.
returnDists	if TRUE (default) returns the same list as provided.
...	additional arguments.

Value

either TRUE/FALSE or list of variable distributions as provided as input.

---

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.

Usage

```
var33
```

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.

- v1Binary, independent.
- v2Gaussian, conditionally dependent upon v1.
- v3Binary, independent.
- v4Binary, conditionally dependent upon v3.
- v5Gaussian, conditionally dependent upon v6.
- v6Binary, conditionally dependent upon v4 and v7.
- v7Gaussian, conditionally dependent upon v8.
- v8Gaussian, conditionally dependent upon v9.
- v9Binary, conditionally dependent upon v10.
- v10Binary, independent.
- v11Binary, conditionally dependent upon v10, v12 and v19.
- v12Binary, independent.
- v13Gaussian, independent.
- v14Gaussian, conditionally dependent upon v13.
- v15Binary, conditionally dependent upon v14 and v21.

- v16Gaussian, independent.
- v17Gaussian, conditionally dependent upon v16 and v20.
- v18Binary, conditionally dependent upon v20.
- v19Binary, conditionally dependent upon v20.
- v20Binary, independent.
- v21Binary, conditionally dependent upon v20.
- v22Gaussian, conditionally dependent upon v21.
- v23Gaussian, conditionally dependent upon v21.
- v24Gaussian, conditionally dependent upon v23.
- v25Gaussian, conditionally dependent upon v23 and v26.
- v26Binary, conditionally dependent upon v20.
- v27Binary, independent.
- v28Binary, conditionally dependent upon v27, v29 and v31.
- v29Gaussian, independent.
- v30Gaussian, conditionally dependent upon v29.
- v31Gaussian, independent.
- v32Binary, conditionally dependent upon v21, v29 and v31.
- v33Gaussian, 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