Package ‘xergm’

August 7, 2014

Version 1.1

Date 2014-08-07

Title Extensions for Exponential Random Graph Models (ERGM)

Description
The xergm package implements temporal exponential random graph models (TERGM) by bootstrapped pseudolikelihood (``btergm'') and (temporal) network autocorrelation models (``tnam'').

Depends R (>= 2.14.0), ergm

Imports stats4, utils, methods, statnet, statnet.common, network, sna,texreg (>= 1.33), Matrix, parallel, boot, coda, stats, ROCR,igraph, vegan, lme4 (>= 1.0)

Suggests RSiena (>= 1.0.12.169)

License GPL (>= 2)

Author Philip Leifeld [aut, cre],Skyler J. Cranmer [aut],Bruce A. Desmarais [aut]

Maintainer Philip Leifeld <philip.leifeld@uni-konstanz.de>

Repository CRAN

Repository/R-Forge/Project xergm

Repository/R-Forge/Revision 66

Repository/R-Forge/DateTimeStamp 2014-08-06 21:32:16

Date/Publication 2014-08-07 14:22:48

NeedsCompilation no

R topics documented:

`xergm-package` *Extensions of Exponential Random Graph Models (ERGM)*

Description

Extensions of Exponential Random Graph Models (ERGM).

Details

The **xergm** package implements extensions of exponential random graph models, in particular bootstrapped temporal ERGMs (`btergm`) and generalized ERGMs (`gergm`). The `btergm` function estimates temporal exponential random graph models (TERGM) by bootstrapped pseudolikelihood and provides various goodness-of-fit and diagnostic measures and plots. GERGMs have not been implemented yet. To display citation information, type `citation("xergm")`.

Author(s)

Philip Leifeld (`http://www.philipleifeld.de`) Skyler J. Cranmer (`http://www.unc.edu/~skylerc/`) Bruce A. Desmarais (`http://people.umass.edu/bruced/`)

See Also

btergm preprocess simulate.btergm gof interpret btergm-class tnam tnam-terms

adjust — *Adjust the dimensions of a matrix to the dimensions of another matrix*

Description

Adjust the dimensions of a matrix to the dimensions of another matrix.

Usage

```
adjust(source, target, remove = TRUE, add = TRUE)
```

Arguments

`source`	A matrix, network, list or data.frame object or a vector which should be adjusted.
`target`	A matrix, network, list or data.frame object or a vector to which the source object is compared with regard to its labels.
`remove`	Should rows and columns that are not present in the target object be removed?
`add`	Should rows and columns that are present in the target object but not in the source object be added to the source object?

Details

An adjacency matrix (the `source` matrix) is compared to another adjacency matrix (the `target` matrix) by matching the row or column labels. If the target matrix contains rows/columns which are not present in the source matrix, new rows and columns with the corresponding labels and NA values in the cells are inserted into the source matrix. If the source matrix contains rows/columns which are not present in the target matrix, these rows and columns are removed from the source matrix. In addition to adjacency matrices, two-mode matrices, network objects (also with vertex attributes), data frames and vectors are supported.

See Also

xergm-package handleMissings preprocess

btergm — *TERGM by bootstrapped pseudolikelihood*

Description

TERGM by bootstrapped pseudolikelihood.

Usage

```
btergm(formula, R = 500, parallel = c("no", "multicore", "snow"),
    ncpus = getOption("boot.ncpus", 1L), cl = NULL, ...)
```

Arguments

`formula`	Formula for the TERGM. Model construction works like in the **ergm** package with the same model terms etc. (for a list of terms, see `help("ergm-terms")`). The networks to be modeled on the left-hand side of the equation must be given either as a list of network objects with more recent networks last (i.e., chronological order) or as a list of matrices with more recent matrices at the end. `dyadcov` and `edgecov` terms accept time-independent covariates (as `network` or `matrix` objects) or time-varying covariates (as a list of networks or matrices with the same length as the list of networks to be modeled).
`R`	Number of bootstrap replications. The higher the number of replications, the more accurate but also the slower is the estimation.
`parallel`	Use multiple cores in a computer or nodes in a cluster to speed up bootstrapping computations. The default value `"no"` means parallel computing is switched off. If `"multicore"` is used, the `mclapply` function from the **parallel** package (formerly in the **multicore** package) is used for parallelization. This should run on any kind of system except MS Windows because it is based on forking. It is usually the fastest type of parallelization. If `"snow"` is used, the `parLapply` function from the **parallel** package (formerly in the **snow** package) is used for parallelization. This should run on any kind of system including cluster systems and including MS Windows. It is slightly slower than the former alternative if the same number of cores is used. However, `"snow"` provides support for MPI clusters with a large amount of cores, which **multicore** does not offer (see also the `cl` argument). The backend for the bootstrapping procedure is the **boot** package.
`ncpus`	The number of CPU cores used for parallel computing (only if `parallel` is activated). If the number of cores should be detected automatically on the machine where the code is executed, one can set `ncpus = detectCores()`. On some HPC clusters, the number of available cores is saved as an environment variable; for example, if MOAB is used, the number of available cores can sometimes be accessed using `Sys.getenv("MOAB_PROCCOUNT")`, depending on the implementation.
`cl`	An optional **parallel** or **snow** cluster for use if `parallel = "snow"`. If not supplied, a cluster on the local machine is created temporarily.
`...`	Further arguments to be handed over to subroutines.

Details

The `btergm` function computes temporal exponential random graph models (TERGM) by bootstrapped pseudolikelihood, as described in Desmarais and Cranmer (2012).

References

Cranmer, Skyler J., Tobias Heinrich and Bruce A. Desmarais (2014): Reciprocity and the Structural Determinants of the International Sanctions Network. *Social Networks* 36(1): 5–22. `http://dx.doi.org/10.1016/j.socnet.2013.01.001`.

Desmarais, Bruce A. and Skyler J. Cranmer (2012): Statistical Mechanics of Networks: Estimation and Uncertainty. *Physica A* 391: 1865–1876. http://dx.doi.org/10.1016/j.physa.2011.10.018.

Desmarais, Bruce A. and Skyler J. Cranmer (2010): Consistent Confidence Intervals for Maximum Pseudolikelihood Estimators. *Neural Information Processing Systems 2010 Workshop on Computational Social Science and the Wisdom of Crowds.*

See Also

xergm-package simulate.btergm gof knecht btergm-class preprocess

Examples

```
# A simple toy example:

library("statnet")
set.seed(5)

networks <- list()
for(i in 1:10){            # create 10 random networks with 10 actors
  mat <- matrix(rbinom(100, 1, .25), nrow = 10, ncol = 10)
  diag(mat) <- 0           # loops are excluded
  nw <- network(mat)       # create network object
  networks[[i]] <- nw      # add network to the list
}

covariates <- list()
for (i in 1:10) {          # create 10 matrices as covariates
  mat <- matrix(rnorm(100), nrow = 10, ncol = 10)
  covariates[[i]] <- mat   # add matrix to the list
}

fit <- btergm(networks ~ edges + istar(2) +
    edgecov(covariates), R = 100)

summary(fit)               # show estimation results

# For an example with real data, see help("knecht").


# Examples for parallel processing:

# btergm and the gof functions work with MPI clusters via the parallel
# package. If no snow cluster is provided, btergm will try to create
# a temporary cluster:

## Not run:
require("parallel")
fit <- btergm(networks ~ edges + istar(2) + edgecov(covariates),
    R = 100, parallel = "snow", ncpus = 25)

## End(Not run)
```

```
# A snow/MPI cluster can be created and used manually as follows. In
# this example, a MOAB HPC server is used. It stores the number of
# available cores as a system option:

## Not run:
require("parallel")
cores <- as.numeric(Sys.getenv("MOAB_PROCCOUNT"))
cl <- makeCluster(cores)
fit <- btergm(networks ~ edges + istar(2) + edgecov(covariates),
    R = 100, parallel = "snow", ncpus = cores, cl = cl)
stopCluster(cl)

## End(Not run)

# In the following example, the Rmpi package is used to create a
# cluster. This may not work on all systems; contact your local
# support staff or the help files on your HPC server to find out how
# to create a cluster object on your system.

## Not run:
# snow/Rmpi start-up
if (!is.loaded("mpi_initialize")) {
    library("Rmpi")
}
library(snow);

mpirank <- mpi.comm.rank (0)
if (mpirank == 0) {
   invisible(makeMPIcluster())
} else {
  sink (file="/dev/null")
  invisible(slaveLoop (makeMPImaster()))
  mpi.finalize()
  q()
}
# End snow/Rmpi start-up

cl <- getMPIcluster()

fit <- btergm(networks ~ edges + istar(2) + edgecov(covariates),
    R = 100, parallel = "snow", ncpus = 25, cl = cl)

## End(Not run)
```

btergm-class *Class* "btergm"

Description

`btergm` objects result from the estimation of a bootstrapped TERGM via the `btergm` function in the **xergm** package. `btergm` objects contain the coefficients, the bootstrapping samples of the coefficients, the number of replications, the number of observations, the number of time steps, the original formula, and the response, effects and weights objects that were fed into the `glm` call for estimating the model.

Usage

```
## S4 method for signature 'btergm'
summary(object, level = 0.95, ...)

## S4 method for signature 'btergm'
show(object)

## S4 method for signature 'btergm'
nobs(object)

## S4 method for signature 'btergm'
coef(object, ...)

## S4 method for signature 'btergm'
confint(object, parm, level = 0.95, ...)

btergm.se(object, print = FALSE)

btergm.timesteps(object)
```

Arguments

`object`	A `btergm` object.
`level`	The significance level for computation of the confidence intervals. The default is `0.95` (that is, an alpha value of 0.05). Other common values include `0.999`, `0.99`, `0.9`, and `0.5`.
`parm`	Parameters (specified by integer position or character string).
`print`	Should the formatted coefficient table be printed to the R console along with significance stars (`print = TRUE`), or should the plain coefficient matrix be returned (`print = FALSE`)?
`...`	Further arguments to be handed over to subroutines.

Details

Various generic methods are available for `btergm` objects: The `coef` and `show` methods return the coefficients; the `summary` method gives a model summary. The `nobs` method returns the number of observations. The confint method returns confidence intervals from the bootstrap replications of `btergm` objects, and the user can specify the confidence level. The method returns a matrix with three columns: the estimate, the lower bound, and the upper bound of the confidence interval for each model term.

The `btergm.se` function computes standard errors and p values for `btergm` objects. It returns a matrix with four columns: the estimate, the standard error, the z value, and the p value for each model term. If the argument `print = TRUE` is used, the matrix is printed to the R console as a formatted coefficient matrix with significance stars instead. Note that confidence intervals are the preferred way of interpretation for bootstrapped TERGMs; standard errors are only accurate if the bootstrapped data are normally distributed, which is not always the case. Various methods for checking for normality for each model term are available, for example quantile-quantile plots (e.g., `qqnorm(x@bootsamp[, 1])` for the first model term in the `btergm` object called `x`).

The `btergm.timesteps` function extracts the number of time steps from a `btergm` object. The number of time steps is the number of networks being modeled on the left-hand side of the model formula.

Slots

`coef`**:** Object of class `"numeric"`. The coefficients.

`bootsamp`**:** Object of class `"matrix"`. The bootstrapping sample.

`R`**:** Object of class `"numeric"`. Number of replications.

`nobs`**:** Object of class `"numeric"`. Number of observations.

`time.steps`**:** Object of class `"numeric"`. Number of time steps.

`formula`**:** Object of class `"formula"`. The original model formula (without indices for the time steps).

`response`**:** Object of class `"integer"`. The response variable.

`effects`**:** Object of class `"data.frame"`. The effects that went into the `glm` call.

`weights`**:** Object of class `"integer"`. The weights of the observations.

References

Cranmer, Skyler J., Tobias Heinrich and Bruce A. Desmarais (2014): Reciprocity and the Structural Determinants of the International Sanctions Network. *Social Networks* 36(1): 5–22. `http://dx.doi.org/10.1016/j.socnet.2013.01.001`.

Desmarais, Bruce A. and Skyler J. Cranmer (2012): Statistical Mechanics of Networks: Estimation and Uncertainty. *Physica A* 391: 1865–1876. `http://dx.doi.org/10.1016/j.physa.2011.10.018`.

Desmarais, Bruce A. and Skyler J. Cranmer (2010): Consistent Confidence Intervals for Maximum Pseudolikelihood Estimators. *Neural Information Processing Systems 2010 Workshop on Computational Social Science and the Wisdom of Crowds*.

See Also

xergm-package btergm simulate.btergm gof knecht getformula interpret `getformula` *Extract the formula from a model.*

Description

Extract the model formula from ergm or btergm objects.

Usage

```
getformula(x)

## S4 method for signature 'ergm'
getformula(x)

## S4 method for signature 'btergm'
getformula(x)
```

Arguments

`x`	A model object, for example a `btergm` or an `ergm` object.

Details

Extract the model formula from ergm or btergm objects.

See Also

gof

`gof-methods` *Conduct Goodness-of-Fit Diagnostics on ERGMs, TERGMs, and SAOMs*

Description

Assess goodness of fit and degeneracy of btergm and other network models.

Usage

```
## S4 method for signature 'btergm'
gof(model, target = NULL,
    formula = getformula(model), nsim = 100, MCMC.interval = 10000,
    MCMC.burnin = 10000, parallel = c("no", "MPI", "SOCK"),
    ncpus = detectCores() - 1, cl = NULL, classicgof = TRUE,
    rocprgof = TRUE, checkdegeneracy = TRUE,
    dsp = TRUE, esp = TRUE, geodist = TRUE, degree = TRUE,
    idegree = TRUE, odegree = TRUE, kstar = TRUE, istar = TRUE,
    ostar = TRUE, pr.impute = "poly4", verbose = TRUE, ...)

## S4 method for signature 'ergm'
gof(model, target = NULL,
    formula = getformula(model), nsim = 100, MCMC.interval = 10000,
    MCMC.burnin = 10000, parallel = c("no", "MPI", "SOCK"),
    ncpus = detectCores() - 1, cl = NULL, classicgof = TRUE,
    rocprgof = TRUE, checkdegeneracy = TRUE,
    dsp = TRUE, esp = TRUE, geodist = TRUE, degree = TRUE,
    idegree = TRUE, odegree = TRUE, kstar = TRUE, istar = TRUE,
    ostar = TRUE, pr.impute = "poly4", verbose = TRUE, ...)

## S4 method for signature 'sienaAlgorithm'
gof(model, siena.data, siena.effects,
    predict.period = NULL, nsim = 50, parallel = c("no", "multicore",
    "snow"), ncpus = detectCores() - 1, cl = NULL, target.na = NA,
    target.na.method = "remove", target.structzero = 10,
    classicgof = TRUE, rocprgof = TRUE, dsp = TRUE, esp = TRUE,
    geodist = TRUE, degree = TRUE, idegree = TRUE, odegree = TRUE,
    kstar = TRUE, istar = TRUE, ostar = TRUE, pr.impute = "poly4",
    ...)

## S4 method for signature 'sienaModel'
gof(model, siena.data, siena.effects,
    predict.period = NULL, nsim = 50, parallel = c("no", "multicore",
    "snow"), ncpus = detectCores() - 1, cl = NULL, target.na = NA,
    target.na.method = "remove", target.structzero = 10,
    classicgof = TRUE, rocprgof = TRUE, dsp = TRUE, esp = TRUE,
    geodist = TRUE, degree = TRUE, idegree = TRUE, odegree = TRUE,
    kstar = TRUE, istar = TRUE, ostar = TRUE, pr.impute = "poly4",
    ...)
```

Arguments

`model`	A `btergm`, `ergm`, `sienaAlgorithm`, or `sienaModel` object.
`siena.data`	An object of the class `siena`, which is usually created using the `sienaDataCreate` function in the `RSiena` package.
`siena.effects`	An object of the class `sienaEffects`, which is usually created using the `getEffects()` and the `includeEffects()` function in the `RSiena` package.
`predict.period`	Which time period should be predicted? By default, the last time period is predicted based on the last simulation of the second-last time period. The time period can be provided as a numeric, e.g., `predict.period = 4` for predicting the fourth network.
`target`	A network or list of networks to which the simulations are compared. If left empty, the original networks from the `btergm` object `x` are used as observed networks.

`formula`	A model formula from which networks are simulated for comparison. By default, the formula from the `btergm` object `x` is used. It is possible to hand over a formula with only a single response network and/or dyad or edge covariates or with lists of response networks and/or covariates. It is also possible to use indices like `networks[[4]]` or `networks[3:5]` inside the formula.
`nsim`	The number of networks to be simulated at each time step. Example: If there are six time steps in the `formula` and `nsim = 100`, a total of 600 new networks is simulated.
`MCMC.interval`	Internally, this package uses the simulation facilities of the **ergm** package to create new networks against which to compare the original network(s) for goodness-of-fit assessment. This argument sets the MCMC interval to be passed over to the simulation command. The default value is `10000`, which means that every 10000th simulation outcome from the MCMC sequence is used. There is no general rule of thumb on the selection of this parameter, but if the results look suspicious (e.g., when the model fit is perfect), increasing this value may be helpful.
`MCMC.burnin`	Internally, this package uses the simulation facilities of the **ergm** package to create new networks against which to compare the original network(s) for goodness-of-fit assessment. This argument sets the MCMC burnin to be passed over to the simulation command. The default value is `10000`. There is no general rule of thumb on the selection of this parameter, but if the results look suspicious (e.g., when the model fit is perfect), increasing this value may be helpful.
`parallel`	Use multiple cores in a computer or nodes in a cluster to speed up the simulations. The default value `"no"` means parallel computing is switched off. If `"multicore"` is used (only available for `sienaAlgorithm` and `sienaModel` objects), the `mclapply` function from the **parallel** package (formerly in the **multicore** package) is used for parallelization. This should run on any kind of system except MS Windows because it is based on forking. It is usually the fastest type of parallelization. If `"snow"` is used (only available for `sienaAlgorithm` and `sienaModel` objects), the `parLapply` function from the **parallel** package (formerly in the **snow** package) is used for parallelization. This should run on any kind of system including cluster systems and including MS Windows. It is slightly slower than the former alternative if the same number of cores is used. However, `"snow"` provides support for MPI clusters with a large amount of cores, which **multicore** does not offer (see also the `cl` argument). If `"MPI"` is used (only available for `btergm` and `ergm` objects), MPI parallelization as implemented in the **ergm** package is used. And if `"SOCK"` is used (only available for `btergm` and `ergm` objects), a SOCK cluster as implemented in the **ergm** package (via the deprecated **snow** package) is used for parallelization. Note that `"multicore"` and `"SOCK"` will only work if all cores are on the same node. For example, if there are three nodes with eight cores each, a maximum of eight CPUs can be used.
`ncpus`	The number of CPU cores used for parallel simulations (only if `parallel` is activated). If the number of cores should be detected automatically on the machine where the code is executed, one can set `ncpus = detectCores()`. On some HPC clusters, the number of available cores is saved as an environment variable; for example, if MOAB is used, the number of available cores can sometimes be

	accessed using `Sys.getenv("MOAB_PROCCOUNT")`, depending on the implementation. Note that the maximum number of connections in a single R session (i.e., to other cores or for opening files etc.) is 128, so fewer than 128 cores should be used at a time.
`cl`	An optional **parallel** or **snow** cluster for use if `parallel = "snow"`. If not supplied, a cluster on the local machine is created temporarily.
`target.na`	Which value was used for missing data in the dependent variable?
`target.na.method`	How should missing data be handled when comparing the simulations to the empirical (= observed) network? Two options are possible: `remove` drops nodes with missing ties both from the simulations (after running the simulations) and from the observed network before the comparison. `fillmode` replaces missing values by the mode of the network matrix (usually `0`).
`target.structzero`	Which value was used for structural zeroes (usually nodes which have dropped out of the network or have not yet joined the network) in the dependent variable? These nodes are removed from the observed network and the simulations before comparison.
`classicgof`	If `classicgof = TRUE` is set, the classic statnet-style goodness-of-fit comparison is conducted. This means that shared-partner statistics, the geodesic distance distribution and the degree distribution are compared between observed and simulated networks. The results can be plotted as boxplots or printed as tables. Note that the `classicgof`, `rocprgof` and `checkdegeneracy` arguments can be used together. In that case, the resulting `btergmgof` object will contain all three types of GOF/degeneracy assessment.
`rocprgof`	If `rocprgof = TRUE` is set, the coordinates of ROC and PR curves as well as the AUC measure are stored in the resulting `btergmgof` object. The results can be plotted as curves or printed as tables. Note that the `classicgof`, `rocprgof` and `checkdegeneracy` arguments can be used together. In that case, the resulting `btergmgof` object will contain all three types of GOF/degeneracy assessment.
`checkdegeneracy`	If `checkdegeneracy = TRUE` is set, the global statistics of the observed and simulated networks are compared for each observed time step separately. Frequent significant deviations indicate degeneracy. The results can be printed as tables. Note that the `classicgof`, `rocprgof` and `checkdegeneracy` arguments can be used together. In that case, the resulting `btergmgof` object will contain all three types of GOF/degeneracy assessment.
`dsp`	Compute the dyad-wise shared partner statistic?
`esp`	Compute the edge-wise shared partner statistic?
`geodist`	Compute the geodesic distance statistic?
`degree`	Compute the degree statistic?
`idegree`	Compute the indegree statistic?
`odegree`	Compute the outdegree statistic?
`kstar`	Compute the kstar statistic?

`istar`	Compute the instar statistic?
`ostar`	Compute the outstar statistic?
`pr.impute`	In some cases, the first precision value of the precision-recall curve is undefined. The `pr.impute` argument serves to impute this missing value to ensure that the AUC-PR value is not severely biased. Possible values are `"no"` for no imputation, `"one"` for using a value of `1.0`, `"second"` for using the next (= adjacent) precision value, `"poly1"` for fitting a straight line through the remaining curve to predict the first value, `"poly2"` for fitting a second-order polynomial curve etc. until `"poly9"` Warning: this is a pragmatic solution. Please double-check whether the imputation makes sense. This can be checked by plotting the resulting `btergmgof` object and using the `pr.poly` argument to plot the predicted curve on top of the actual PR curve.
`verbose`	Print details?
`...`	Arbitrary further arguments are handed over to the `simulate.formula` function or the `siena07` function. For details, refer to the help page of these functions.

Details

The generic `gof` function provides goodness-of-fit measures and degeneracy checks for `btergm`, `ergm` and Siena-type models. Three different types of GOF/degeneracy assessment are possible with this function:

(1) Classic statnet-type GOF assessment by comparing summary statistics of observed and simulated networks. The `gof` function has six built-in statistics: dyad-wise shared partners (dsp), edge-wise shared partners (esp), degree (for undirected networks only), indegree (for directed networks only), outdegree (for directed networks only), and geodesic distances. The comparison can be plotted using boxplots for the simulations and lines for the observed network(s) or printed using t-tests (testing whether simulated and observed networks are significantly different for all values in the distributions of the summary statistics).

(2) An assessment of the classification performance using receiver operating characteristics (ROC) and precision-recall (PR) curves as well as the area under the curve (AUC) for the ROC curve.

(3) A degeneracy check by comparing the global statistics of simulated networks to those of the observed networks at each observed time step. If the global statistics differ significantly, this is indicated by small p values. If there are many significant results, this indicates degeneracy.

For all three types of GOF assessment, by default, in-sample predictive performance is assessed by comparing all observed networks to all simulations from the same networks (just like in the **ergm** package, but aggregated over several time steps). If an observed network or a list of observed networks is provided as the `target` argument, the simulations are compared to these networks instead. This is useful for out-of-sample prediction. If a formula is provided, the simulations are based on the networks and covariates specified in the formula. This is helpful in situations where complex out-of-sample predictions have to be evaluated. A usage scenario could be to simulate from a network at time `t` (provided through the `formula` argument) and compare to an observed network at time `t + 1` (the `target` argument). This can be done, for example, to assess predictive performance between time steps of the original networks, or to check whether the model performs well with regard to a newly measured network given the old data from the previous time step.

Predictive fit can also be assessed for stochastic actor-oriented models (SAOM) as implemented in the **RSiena** package. After compiling the usual objects (model, data, effects), one of the time steps can be predicted based on the previous time step and the SAOM using the `sienaAlgorithm` (for **RSiena** >= 1.1-227) or `sienaModel` (for **RSiena** < 1.1-227) method of the `gof` function.

See also the `plot.btergmgof` help page for details on the plotting and printing options for GOF assessment.

See Also

xergm-package btergm simulate.btergm simulate.formula

`handleMissings` *Handle missing data in matrices.*

Description

Handle missing data in matrices.

Usage

```
handleMissings(mat, na = NA, method = "remove", logical = FALSE)
```

Arguments

`mat`	A matrix object.
`na`	The value that missing data are coded as. Usually `NA`, sometimes `9` or `10`.
`method`	What should be done with the missing data? If `method = "remove"` is set, the function determines how many missing entries are in each row and column and iteratively removes rows or columns with the largest amount of missing data until no missing data are left in the matrix. If `method = "fillmode"` is set, the modal value of the matrix is identified (usually `0` in network matrices) and missing cells are imputed by filling in this modal value. `method = "zero"` replaces NAs by 0s.
`logical`	Return a matrix with logical values indicating which cells should be removed? By default the manipulated matrix is returned.

Details

This function deals with missing data in matrices or network objects used for inferential network analysis. It can either remove missing rows and/or columns iteratively (rows and columns with more NA values first, then successively rows and columns with fewer NA entries) or replace missing values by the modal value of the matrix or by 0. The function can return either the manipulated matrix or a matrix with logical values indicating which of the cells should be removed.

See Also

xergm-package adjust preprocess

`interpret` *Interpretation functions for ergm and btergm objects*

Description

Interpretation functions for ergm and btergm objects.

Usage

```
interpret(object, ...)

## S4 method for signature 'ergm'
interpret(object, formula = object$formula,
    coefficients = coef(object), network = eval(parse(text =
    deparse(formula[[2]]))), type = "tie", i, j, ...)

## S4 method for signature 'btergm'
interpret(object, formula = object@formula,
    coefficients = coef(object), network = eval(parse(text =
    deparse(formula[[2]]))), type = "tie", i, j,
    t = 1:length(network), ...)
```

Arguments

`object`	An `ergm` or `btergm` object.
`formula`	The formula to be used for computing probabilities. By default, the formula embedded in the model object is retrieved and used.
`coefficients`	The estimates on which probabilities should be based. By default, the coefficients from the model object are retrieved and used. Custom coefficients can be handed over, for example, in order to compare versions of the model where the reciprocity term is fixed at 0 versus versions of the model where the reciprocity term is left as in the empirical result. This is one of the examples described in Desmarais and Cranmer (2012).
`network`	The response network on which probabilities are based. Depending on whether the function is applied to an `ergm` or `btergm` object, this can be either a single network or a list of networks.
`type`	If `type = "tie"` is used, probabilities at the edge level are computed. For example, what is the probability of a specific node i to be connected to a specific node j given the rest of the network and given the model? If `type = "dyad"` is used, probabilities at the dyad level are computed. For example, what is the probability that node i is connected to node j but not vice-versa, or what is the probability that nodes i and j and mutually connected in a directed network? If `type = "node"` is used, probabilities at the node level are computed. For example, what is the probability that node i is connected to a set of three other j nodes given the rest of the network and the model?

`i`	A single (sender) node i or a set of (sender) nodes i. If `type = "node"` is used, this can be more than one node and should be provided as a vector. The `i` argument can be either provided as the index of the node in the sociomatrix (e.g., the fourth node would be `i = 4`) or the row name of the node in the sociomatrix (e.g., `i = "Peter"`). If more than one node is provided and `type = "node"`, there can be only one (receiver) node j. The `i` and `j` arguments are used to specify for which nodes probabilities should be computed. For example, what is the probability that `i = 4` is connected to `i = 7`?
`j`	A single (receiver) node j or a set of (receiver) nodes j. If `type = "node"` is used, this can be more than one node and should be provided as a vector. The `j` argument can be either provided as the index of the node in the sociomatrix (e.g., the fourth node would be `j = 4`) or the row name of the node in the sociomatrix (e.g., `j = "Mary"`). If more than one node is provided and `type = "node"`, there can be only one (sender) node i. The `i` and `j` arguments are used to specify for which nodes probabilities should be computed. For example, what is the probability that `i = 4` is connected to `i = 7`?
`t`	A vector of (numerical) time steps for which the probabilities should be computed. This only applies to `btergm` objects because `ergm` objects are by definition based on a single time step. By default, all available time steps are used. It is, for example, possible to compute probabilities only for a single time step by specifying, e.g., `t = 5` in order to compute probabilities for the fifth response network.
`...`	Further arguments to be handed over to subroutines.

Details

The `interpret` function facilitates interpretation of ERGMs and TERGMs at the micro level, as described in Desmarais and Cranmer (2012). There are generic methods which work with `ergm` objects and `btergm` objects. The function can be used to interpret these models at the tie or edge level, dyad level, and block level.

For example, what is the probability that two specific nodes i (the sender) and node j (the receiver) are connected given the rest of the network and given the model? Or what is the probability that any two nodes are tied at t = 2 if they were tied (or disconnected) at t = 1 (i.e., what is the amount of tie stability)? These tie- or edge-level questions can be answered if the `type = "tie"` argument is used.

Another example: What is the probability that node i has a tie to node j but not vice-versa? Or that i and j maintain a reciprocal tie? Or that they are disconnected? How much more or less likely are i and j reciprocally connected if the `mutual` term in the model is fixed at 0 (compared to the model that includes the estimated parameter for reciprocity)? See example below. These dyad-level questions can be answered if the `type = "dyad"` argument is used.

Or what is the probability that a specific node i is connected to nodes j1 and j2 but not to j5 and j7? And how likely is any node i to be connected to exactly four j nodes? These node-level questions (focusing on the ties of node i or node j) can be answered by using the `type = "node"` argument.

References

Desmarais, Bruce A. and Skyler J. Cranmer (2012): Micro-Level Interpretation of Exponential Random Graph Models with Application to Estuary Networks. *The Policy Studies Journal* 40(3):

402–434.

See Also

xergm-package btergm btergm.timesteps

Examples

```
##### The following example is a TERGM adaptation of the #####
##### dyad-level example provided in figure 5(c) on page #####
##### 424 of Desmarais and Cranmer (2012) in the PSJ. At #####
##### each time step, it compares dyadic probabilities   #####
##### (no tie, unidirectional tie, and reciprocal tie    #####
##### probability) between a fitted model and a model    #####
##### where the reciprocity effect is fixed at 0 based   #####
##### on 20 randomly selected dyads per time step. The   #####
##### results are visualized using a grouped bar plot.   #####

## Not run:
# create toy dataset and fit a model
networks <- list()
for (i in 1:3) {           # create 3 random networks with 10 actors
  mat <- matrix(rbinom(100, 1, 0.25), nrow = 10, ncol = 10)
  diag(mat) <- 0           # loops are excluded
  nw <- network(mat)       # create network object
  networks[[i]] <- nw      # add network to the list
}
fit <- btergm(networks ~ edges + istar(2) + mutual, R = 200)

# extract coefficients and create null hypothesis vector
null <- coef(fit)  # estimated coefs
null[3] <- 0       # set mutual term = 0

# sample 20 dyads per time step and compute probability ratios
probabilities <- matrix(nrow = 9, ncol = length(networks))
# nrow = 9 because three probabilities + upper and lower CIs
colnames(probabilities) <- paste("t =", 1:length(networks))
for (t in 1:length(networks)) {
  d <- dim(as.matrix(networks[[t]]))  # how many row and column nodes?
  size <- d[1] * d[2]                 # size of the matrix
  nw <- matrix(1:size, nrow = d[1], ncol = d[2])
  nw <- nw[lower.tri(nw)]             # sample only from lower triangle b/c
  samp <- sample(nw, 20)              # dyadic probabilities are symmetric
  prob.est.00 <- numeric(0)
  prob.est.01 <- numeric(0)
  prob.est.11 <- numeric(0)
  prob.null.00 <- numeric(0)
  prob.null.01 <- numeric(0)
  prob.null.11 <- numeric(0)
  for (k in 1:20) {
    i <- arrayInd(samp[k], d)[1, 1]   # recover 'i's and 'j's from sample
    j <- arrayInd(samp[k], d)[1, 2]
    # run interpretation function with estimated coefs and mutual = 0:
```

```
    int.est <- interpret(fit, type = "dyad", i = i, j = j, t = t)
    int.null <- interpret(fit, coefficients = null, type = "dyad",
        i = i, j = j, t = t)
    prob.est.00 <- c(prob.est.00, int.est[[1]][1, 1])
    prob.est.11 <- c(prob.est.11, int.est[[1]][2, 2])
    mean.est.01 <- (int.est[[1]][1, 2] + int.est[[1]][2, 1]) / 2
    prob.est.01 <- c(prob.est.01, mean.est.01)
    prob.null.00 <- c(prob.null.00, int.null[[1]][1, 1])
    prob.null.11 <- c(prob.null.11, int.null[[1]][2, 2])
    mean.null.01 <- (int.null[[1]][1, 2] + int.null[[1]][2, 1]) / 2
    prob.null.01 <- c(prob.null.01, mean.null.01)
  }
  prob.ratio.00 <- prob.est.00 / prob.null.00  # ratio of est. and null hyp
  prob.ratio.01 <- prob.est.01 / prob.null.01
  prob.ratio.11 <- prob.est.11 / prob.null.11
  probabilities[1, t] <- mean(prob.ratio.00)   # mean estimated 00 tie prob
  probabilities[2, t] <- mean(prob.ratio.01)   # mean estimated 01 tie prob
  probabilities[3, t] <- mean(prob.ratio.11)   # mean estimated 11 tie prob
  ci.00 <- t.test(prob.ratio.00, conf.level = 0.99)$conf.int
  ci.01 <- t.test(prob.ratio.01, conf.level = 0.99)$conf.int
  ci.11 <- t.test(prob.ratio.11, conf.level = 0.99)$conf.int
  probabilities[4, t] <- ci.00[1]              # lower 00 conf. interval
  probabilities[5, t] <- ci.01[1]              # lower 01 conf. interval
  probabilities[6, t] <- ci.11[1]              # lower 11 conf. interval
  probabilities[7, t] <- ci.00[2]              # upper 00 conf. interval
  probabilities[8, t] <- ci.01[2]              # upper 01 conf. interval
  probabilities[9, t] <- ci.11[2]              # upper 11 conf. interval
}

# create barplots from probability ratios and CIs
require("gplots")
bp <- barplot2(probabilities[1:3, ], beside = TRUE, plot.ci = TRUE,
    ci.l = probabilities[4:6, ], ci.u = probabilities[7:9, ],
    col = c("tan", "tan2", "tan3"), ci.col = "grey40",
    xlab = "Dyadic tie values", ylab = "Estimated Prob./Null Prob.")
mtext(1, at = bp, text = c("(0,0)", "(0,1)", "(1,1)"), line = 0, cex = 0.5)

## End(Not run)
```

knecht *Longitudinal classroom friendship network and behavior (Andrea Knecht)*

Description

The Knecht dataset contains the friendship network of 26 pupils in a Dutch school class measured at four time points along with several demographic and behavioral covariates like age, sex, ethnicity, religion, delinquency, alcohol consumption, primary school co-attendance, and school advice. Some of these covariates are constant while others vary over time.

The full dataset (see Knecht 2006 and 2008) contains a large number of classrooms while the dataset presented here is an excerpt based on one single classroom. This excerpt was first used in a tutorial for the software **Siena** and the corresponding R package **RSiena** (Snijders, Steglich and van de Bunt 2010). The following description was largely copied from the original data description provided on the homepage of the **Siena** project (see below for the URL).

The data were collected between September 2003 and June 2004 by Andrea Knecht, supervised by Chris Baerveldt, at the Department of Sociology of the University of Utrecht (NL). The entire study is reported in Knecht (2008). The project was funded by the Netherlands Organisation for Scientific Research NWO, grant 401-01-554. The 26 students were followed over their first year at secondary school during which friendship networks as well as other data were assessed at four time points at intervals of three months. There were 17 girls and 9 boys in the class, aged 11–13 at the beginning of the school year. Network data were assessed by asking students to indicate up to twelve classmates which they considered good friends. Delinquency is defined as a rounded average over four types of minor delinquency (stealing, vandalism, graffiti, and fighting), measured in each of the four waves of data collection. The five-point scale ranged from ‘never’ to ‘more than 10 times’, and the distribution is highly skewed. In a range of 1–5, the mode was 1 at all four waves, the average rose over time from 1.4 to 2.0, and the value 5 was never observed.

Usage

```
data(knecht)
```

Format

Note: the data have to be transformed before they can be used with the `btergm` function from the **xergm** package (see examples below).

`friendship` is a list of adjacency matrices at four time points, containing friendship nominations of the column node by the row node. The following values are used: `0` = no, `1` = yes, `NA` = missing, `10` = not a member of the classroom (structural zero).

`demographics` is a data frame with 26 rows (the pupils) and four demographic variables about the pupils:

- `sex` (`1` = girl, `2` = boy)
- `age` (in years)
- `ethnicity` (`1` = Dutch, `2` = other, `0` = missing)
- `religion` (`1` = Christian, `2` = non-religious, `3` = non-Christian religion, `0` = missing)

`primary` is a 26 x 26 matrix indicating whether two pupils attended the same primary school. `0` = no, `1` = yes.

`delinquency` is a data frame with 26 rows (the pupils) and four columns (the four time steps). It contains the rounded average of four items (stealing, vandalizing, fighting, graffiti). Categories: frequency over last three months, `1` = never, `2` = once, `3` = 2–4 times, `4` = 5–10 times, `5` = more than 10 times; `0` = missing.

`alcohol` is a data frame with 26 rows (the pupils) and 3 columns (waves 2, 3, and 4). It contains data on alcohol use (“How often did you drink alcohol with friends in the last three months?”). Categories: `1` = never, `2` = once, `3` = 2–4 times, `4` = 5–10 times, `5` = more than 10 times; `0` = missing.

`advice` is a data frame with one variable, “school advice”, the assessment given at the end of primary school about the school capabilities of the pupil (`4` = low, `8` = high, `0` = missing)

Source

The data were gathered by Andrea Knecht, as part of her PhD research, building on methods developed by Chris Baerveldt, initiator and supervisor of the project. The project is funded by the Netherlands Organisation for Scientific Research NWO, grant 401-01-554, and is part of the research program "Dynamics of Networks and Behavior" with principle investigator Tom A. B. Snijders.

- Complete original data: `https://easy.dans.knaw.nl/ui/datasets/id/easy-dataset:48665`
- This excerpt in Siena format: `http://www.stats.ox.ac.uk/~snijders/siena/klas12b.zip`
- Siena dataset description: `http://www.stats.ox.ac.uk/~snijders/siena/tutorial2010_data.htm`

Permission to redistribute this dataset along with the **xergm** package was granted by Andrea Knecht on April 17, 2014. Questions about the data or the original study should be directed to her.

References

Knecht, Andrea (2006): *Networks and Actor Attributes in Early Adolescence* [2003/04]. Utrecht, The Netherlands Research School ICS, Department of Sociology, Utrecht University. (ICS-Codebook no. 61).

Knecht, Andrea (2008): *Friendship Selection and Friends' Influence. Dynamics of Networks and Actor Attributes in Early Adolescence*. PhD Dissertation, University of Utrecht. `http://igitur-archive.library.uu.nl/dissertations/2008-0125-200537/`.

Knecht, Andrea, Tom A. B. Snijders, Chris Baerveldt, Christian E. G. Steglich, and Werner Raub (2010): Friendship and Delinquency: Selection and Influence Processes in Early Adolescence. *Social Development* 19(3): 494–514. `http://dx.doi.org/10.1111/j.1467-9507.2009.00564.x`.

Leifeld, Philip and Skyler J. Cranmer (2014): Comparing the Temporal Exponential Random Graph Model (TERGM) and the Stochastic Actor-Oriented Model (SAOM). Paper presented at the 2014 Annual Meeting of the Swiss Political Science Association (SVPW), Bern, Switzerland, January 30.

Snijders, Tom A. B., Christian E. G. Steglich, and Gerhard G. van de Bunt (2010): Introduction to Actor-Based Models for Network Dynamics. *Social Networks* 32: 44–60. `http://dx.doi.org/10.1016/j.socnet.2009.02.004`.

Steglich, Christian E. G. and Andrea Knecht (2009): Die statistische Analyse dynamischer Netzwerkdaten. In: Stegbauer, Christian and Roger Haeussling (editors), *Handbuch der Netzwerkforschung*, Wiesbaden: Verlag fuer Sozialwissenschaften.

Examples

```
## Not run:
# for more details, see the package vignette -- run vignette("xergm")

require("texreg")
require("sna")
require("xergm")
require("RSiena")
data("knecht")

# what do the networks look like? plot them!
par(mfrow = c(2, 2), mar = c(0, 0, 1, 0))
for (i in 1:length(friendship)) {
  plot(network(friendship[[i]]), main = paste("t =", i),
      usearrows = FALSE, edge.col = "grey50")
}

# ===================================================================
# Preprocess data and estimate TERGMs using the btergm function
# ===================================================================

# first, estimate a temporally pooled ERGM

# the Knecht network has the same dimensions at all time steps; "10"
# values indicate composition change; NA values are missings

# step 1: make sure the network matrices have node labels
for (i in 1:length(friendship)) {
  rownames(friendship[[i]]) <- 1:nrow(friendship[[i]])
  colnames(friendship[[i]]) <- 1:ncol(friendship[[i]])
}
rownames(primary) <- rownames(friendship[[1]])
colnames(primary) <- colnames(friendship[[1]])

# step 2: preprocess dep. NW, remove struct. zeros, impute NA with 0:
# get rid of missing data in the friendship networks (the "dependent
# variable") and adjust the dimensions temporally

dep <- preprocess(friendship, primary, demographics$sex, lag = FALSE,
    covariate = FALSE, na = NA, na.method = "fillmode",
    structzero = 10, structzero.method = "remove")
    # this has to be repeated for the covariates if they have NAs

length(dep)              # make sure there are still four time steps
sapply(friendship, dim)  # network dimensions: 26-26-26-26
sapply(dep, dim)         # after the adjustment: 26-26-25-25
rownames(dep[[3]])       # node 21 (an NA value) was removed

# step 3: preprocess covariates; adjust them to the "dep" dimensions

primary.cov <- preprocess(primary, dep, demographics$sex,
    lag = FALSE, covariate = TRUE)
sex.cov <- preprocess(demographics$sex, primary.cov, dep,
    lag = FALSE, covariate = TRUE)

sapply(primary.cov, dim)      # 26-26-25-25
sapply(sex.cov, length)       # 26-26-25-25

# step 4: add nodal covariates to the networks
```

```
for (i in 1:length(dep)) {
  dep[[i]] <- network(dep[[i]])
  odegsqrt <- sqrt(degree(dep[[i]], cmode = "outdegree"))
  idegsqrt <- sqrt(degree(dep[[i]], cmode = "indegree"))
  dep[[i]] <- set.vertex.attribute(dep[[i]], "odegsqrt", odegsqrt)
  dep[[i]] <- set.vertex.attribute(dep[[i]], "idegsqrt", idegsqrt)
  dep[[i]] <- set.vertex.attribute(dep[[i]], "sex",
      sex.cov[[i]])
}

# step 5: estimate the TERGM without any lagged terms
model1 <- btergm(dep ~ edges + mutual + ttriple + transitiveties +
    ctriple + nodeicov("idegsqrt") + nodeicov("odegsqrt") +
    nodeocov("odegsqrt") + nodeofactor("sex") + nodeifactor("sex") +
    nodematch("sex") + edgecov(primary.cov), R = 100)

summary(model1, level = 0.95)  # look at the results of the estimation

# step 5: plot goodness of fit (see ?gof.btergm and ?plot.btergmgof)
gof1 <- gof(model1, nsim = 25)
plot(gof1)  # display boxplot diagram


# ====================================================================
# Add cross-temporal dynamics
# ====================================================================

# we now include cross-temporal dynamics

# step 1: preprocess data; this time with a lag
# dep contains t = 2 to t = 4
# lag contains t = 1 to t = 3
# mem indicates edge stability between 1-2, 2-3, and 3-4
dep <- preprocess(friendship, primary, demographics$sex, lag = TRUE,
    covariate = FALSE, na = NA, na.method = "fillmode",
    structzero = 10, structzero.method = "remove")
lag <- preprocess(friendship, primary, demographics$sex, lag = TRUE,
    covariate = TRUE, na = NA, na.method = "fillmode",
    structzero = 10, structzero.method = "remove")
mem <- preprocess(friendship, primary, demographics$sex, lag = TRUE,
    covariate = TRUE, memory = "stability", na = NA,
    na.method = "fillmode", structzero = 10,
    structzero.method = "remove")
primary.cov <- preprocess(primary, dep, demographics$sex,
    lag = FALSE, covariate = TRUE)
sex.cov <- preprocess(demographics$sex, primary.cov, dep,
    lag = FALSE, covariate = TRUE)

length(dep)       # now there are only three time steps
sapply(dep, dim)  # after the adjustment: 26-25-25
sapply(lag, dim)  # 26-25-25
sapply(mem, dim)  # 26-25-25 (stability between t and t - 1)
```

```
# delayed reciprocity: are ties from t - 1 reciprocated at t?
delrecip <- lapply(friendship, t)  # transpose friendship matrices
delrecip <- preprocess(delrecip, primary, friendship, lag = TRUE,
    covariate = TRUE, na = NA, na.method = "fillmode",
    structzero = 10, structzero.method = "remove")
sapply(delrecip, dim)

# step 3: add nodal covariates to the networks
for (i in 1:length(dep)) {
  dep[[i]] <- network(dep[[i]])
  odegsqrt <- sqrt(degree(dep[[i]], cmode = "outdegree"))
  idegsqrt <- sqrt(degree(dep[[i]], cmode = "indegree"))
  dep[[i]] <- set.vertex.attribute(dep[[i]], "odegsqrt", odegsqrt)
  dep[[i]] <- set.vertex.attribute(dep[[i]], "idegsqrt", idegsqrt)
  dep[[i]] <- set.vertex.attribute(dep[[i]], "sex", sex.cov[[i]])
}

# step 4: estimate the TERGM with cross-temporal model terms
model2 <- btergm(dep ~ edges + mutual + ttriple + transitiveties +
    ctriple + nodeicov("idegsqrt") + nodeicov("odegsqrt") +
    nodeocov("odegsqrt") + nodeofactor("sex") + nodeifactor("sex") +
    nodematch("sex") + edgecov(primary.cov) + edgecov(delrecip) +
    edgecov(mem), R = 100)

# look at the results (first using tables, then visually)
summary(model2)
screenreg(list(model1, model2))  # compare the two models directly
plotreg(model2, custom.model.names = "Model 2", custom.coef.names =
    c("Edges", "Reciprocity", "Transitive triples",
    "Transitive ties", "Cyclic triples", "Indegree popularity",
    "Outdegree popularity", "Outdegree activity", "Ego = male",
    "Alter = male", "Both nodes = male", "Same primary school",
    "Delayed reciprocity", "Memory term (edge stability)"),
    omit.coef = "Edges", file="coefs.pdf")

# step 5: plot goodness of fit (see ?gof.btergm and ?plot.btergmgof)
gof2 <- gof(model2, nsim = 25)
plot(gof2)  # display boxplot diagram; model fit is much better now!


# ====================================================================
# assess out-of-sample predictive performance of the model
# ====================================================================

# we also want to try to predict the network at t = 4 based on a model
# estimated for t = 1 through t = 3
model3 <- btergm(dep[1:2] ~ edges + mutual + ttriple + transitiveties +
    ctriple + nodeicov("idegsqrt") + nodeicov("odegsqrt") +
    nodeocov("odegsqrt") + nodeofactor("sex") + nodeifactor("sex") +
    nodematch("sex") + edgecov(primary.cov[1:2]) + edgecov(delrecip[1:2]) +
    edgecov(mem[1:2]), R = 100)

screenreg(list(model1, model2, model3))  # similar results as before
```

```
# simulate 100 networks from t = 3 and compare to t = 4
gof3 <- gof(model3, nsim = 100, target = dep[[3]], formula = dep[[3]] ~
    edges + mutual + ttriple + transitiveties + ctriple +
    nodeicov("idegsqrt") + nodeicov("odegsqrt") +
    nodeocov("odegsqrt") + nodeofactor("sex") + nodeifactor("sex") +
    nodematch("sex") + edgecov(primary.cov[[3]]) +
    edgecov(delrecip[[3]]) + edgecov(mem[[3]]))

# display goodness of fit
plot(gof3, roc = FALSE, pr = FALSE)  # predictive fit (boxplots)
gof3  # display goodness of fit tables
pdf("rocpr.pdf")
plot(gof3, boxplot = FALSE, pr = FALSE, roc = TRUE,
    roc.random = TRUE, pr.random = FALSE, ylab = "TPR/PPV",
    xlab = "FPR/TPR", roc.main = "ROC and PR curves")  # ROC curve
plot(gof3, boxplot = FALSE, roc = FALSE, pr = TRUE,
    pr.random = TRUE, rocpr.add = TRUE)  # add PR curve
dev.off()

# ====================================================================
# assess predictive goodness of fit of a SAOM (estimated using RSiena)
# ====================================================================

# determine composition change
comp <- rep(list(c(1, 4)), 26)  # actors are present from t=1 to t=4
comp[[21]] <- c(1, 2)  # actor 21 drops out after the second time step
changes <- sienaCompositionChange(comp)

# prepare networks and covariate model terms
siena.nets <- sienaNet(array(c(friendship[[1]], friendship[[2]],
    friendship[[3]], friendship[[4]]), dim = c(26, 26, 4)))
primaryCov <- coDyadCovar(primary)
sexCov <- coCovar(demographics$sex)
ageCov <- coCovar(demographics$age)
ethnicityCov <- coCovar(demographics$ethnicity)
religionCov <- coCovar(demographics$religion)

mymodel <- sienaModelCreate(useStdInits = FALSE, projname = "myproject")

mydata <- sienaDataCreate(siena.nets, primaryCov, sexCov, ageCov,
    ethnicityCov, religionCov, changes)

# add effects
myeff <- getEffects(mydata)
myeff <- includeEffects(myeff, transTrip)
myeff <- includeEffects(myeff, transTies)
myeff <- includeEffects(myeff, cycle3)
myeff <- includeEffects(myeff, outPopSqrt)
myeff <- includeEffects(myeff, egoX, interaction1 = "sexCov")
myeff <- includeEffects(myeff, altX, interaction1 = "sexCov")
myeff <- includeEffects(myeff, sameX, interaction1 = "sexCov")
myeff <- includeEffects(myeff, X, interaction1 = "primaryCov")
myeff <- includeEffects(myeff, inPopSqrt)
myeff <- includeEffects(myeff, outActSqrt)

# estimate the SIENA model
ans <- siena07(mymodel, data = mydata, effects = myeff, batch = TRUE,
    verbose = FALSE)
ans  # display results

# finally, assess predictive performance of the model using the xergm
# package; this should take up to 10 minutes on a recent PC
siena.gof <- gof(
    mymodel,                     # hand over the SIENA model
    siena.data = mydata,         # ... and the SIENA data
    siena.effects = myeff,       # ... and the SIENA effects
    nsim = 3,                    # number of estimation replications
                                 # (should be 100, but is very slow!)
    target.na = NA,              # how are NA values denoted?
    target.na.method = "remove", # remove NAs in comparison network
    target.structzero = 10,      # how are structural zeros denoted?
    parallel = "no",             # adjust this for parallel processing
    ncpus = 1                    # number of CPU cores
)

siena.gof  # display goodness of fit results

# plot the goodness of fit (see ?gof.btergmgof for details)
plot(siena.gof, roc = FALSE, pr = FALSE)  # display boxplot diagram
plot(siena.gof, boxplot = FALSE, pr = FALSE)  # display ROC curve
plot(siena.gof, boxplot = FALSE, roc = FALSE)  # display PR curve

# compare ROC and PR curves for TERGM and SAOM
plot(siena.gof, boxplot = FALSE, pr = FALSE, roc.col = "red1",
    ylab = "", xlab = "", roc.main = "SAOM versus TERGM prediction")
plot(siena.gof, boxplot = FALSE, roc = FALSE, pr.col = "steelblue1",
    rocpr.add = TRUE)
plot(gof3, boxplot = FALSE, pr = FALSE, roc.col = "red4",
    rocpr.add = TRUE)
plot(gof3, boxplot = FALSE, roc = FALSE, pr.col = "steelblue4",
    rocpr.add = TRUE)
color <- c("red1", "red4", "steelblue1", "steelblue4")
label <- c("SAOM ROC", "TERGM ROC", "SAOM PR", "TERGM PR")
legend("bottom", legend = label, col = color, lty = 1, lwd = 3)

# compare area under the curve
ROC <- c(SAOM = siena.gof$auc.roc, TERGM = gof3$auc.roc)
PR <- c(SAOM = siena.gof$auc.pr, TERGM = gof3$auc.pr)
barplot(cbind(ROC, PR), beside = TRUE, col = color)
legend("topright", legend = label, pch = 15, col = color)


## End(Not run)
```

`plot.btergmgof` *Plot or print btergmgof objects*

Description

Plot or print formatted goodness-of-fit statistics and degeneracy checks from btergmgof objects.

Usage

```
## S3 method for class 'btergmgof'
plot(x, boxplot = TRUE, boxplot.mfrow = TRUE,
    boxplot.dsp = TRUE, boxplot.esp = TRUE, boxplot.geodist = TRUE,
    boxplot.degree = TRUE, boxplot.idegree = TRUE,
    boxplot.odegree = TRUE, boxplot.kstar = TRUE, boxplot.istar = TRUE,
    boxplot.ostar = TRUE, boxplot.dsp.max = NULL,
    boxplot.esp.max = NULL, boxplot.geodist.max = NULL,
    boxplot.degree.max = NULL, boxplot.idegree.max = NULL,
    boxplot.odegree.max = NULL, boxplot.kstar.max = NULL,
    boxplot.istar.max = NULL, boxplot.ostar.max = NULL,
    boxplot.transform = function(x) x, boxplot.border = "darkgray",
    boxplot.mean.col = "black", boxplot.median.col = "black",
    boxplot.lwd = 0.8, boxplot.outline = FALSE, boxplot.ylab = "Frequency",
    boxplot.main = NULL, boxplot.ylim = NULL, roc = TRUE, pr = TRUE,
    rocpr.add = FALSE, rocpr.avg = c("none", "horizontal", "vertical",
    "threshold"), rocpr.spread = c("boxplot", "stderror", "stddev"),
    rocpr.lwd = 3, roc.main = NULL, roc.random = FALSE, roc.col = "#bd0017",
    roc.random.col = "#bd001744", pr.main = NULL, pr.random = FALSE,
    pr.col = "#5886be", pr.random.col = "#5886be44", pr.poly = 0, ...)

## S3 method for class 'btergmgof'
print(x, classicgof = TRUE,
    rocprgof = TRUE, degeneracy = TRUE, ...)

## S3 method for class 'btergmgof'
summary(object, classicgof = TRUE,
    rocprgof = TRUE, degeneracy = TRUE, ...)
```

Arguments

`x`	A `btergmgof` object created by calling the gof.btergm method.
`boxplot`	Create a plot where simulated statistics are summarized as boxplots and observed networks as lines (= classic statnet-style GOF plots)?
`boxplot.mfrow`	Should the classic GOF plots come out separately (`boxplot.mfrow = FALSE`), or should all statistics be aligned in a single diagram (`boxplot.mfrow = TRUE`)? Returning the plots separately can be helpful if the output is redirected to a multipage PDF or TIFF file.

`boxplot.dsp`	Plot a GOF diagram for dyad-wise shared partners?
`boxplot.esp`	Plot a GOF diagram for edge-wise shared partners?
`boxplot.geodist`	Plot a GOF diagram for geodesic distances?
`boxplot.degree`	Plot a GOF diagram for the degree distribution?
`boxplot.idegree`	Plot a GOF diagram for the indegree distribution?
`boxplot.odegree`	Plot a GOF diagram for the outdegree distribution?
`boxplot.kstar`	Plot a GOF diagram for the kstar distribution?
`boxplot.istar`	Plot a GOF diagram for the instar distribution?
`boxplot.ostar`	Plot a GOF diagram for the outstar distribution?
`boxplot.dsp.max`	Upper bound of dyad-wise shared partners to report.
`boxplot.esp.max`	Upper bound of edge-wise shared partners to report.
`boxplot.geodist.max`	Upper bound of geodesic distances to report (excluding `Inf` values).
`boxplot.degree.max`	Upper bound of the degree statistic to report.
`boxplot.idegree.max`	Upper bound of the indegree statistic to report.
`boxplot.odegree.max`	Upper bound of the outdegree statistic to report.
`boxplot.kstar.max`	Upper bound of the kstar statistic to report.
`boxplot.istar.max`	Upper bound of the istar statistic to report.
`boxplot.ostar.max`	Upper bound of the ostar statistic to report.
`boxplot.transform`	A function which transforms the `y` values used for the boxplots. For example, if some of the values become very large and make the output illegible, `boxplot.transform = function(x) x^0.1` or a similar transformation of the values can be used. Note that logarithmic transformations often produce infinite values because `log(0) = -Inf`, so one should rather use something like `boxplot.transform = function(x) log1p` to avoid infinite values.
`boxplot.border`	Color of the borders of the boxplots.
`boxplot.mean.col`	Color of the mean of the observed network statistic.
`boxplot.median.col`	Color of the median of the observed network statistic.
`boxplot.lwd`	Scaling factor for the line width of the boxplots and of the line for the observed network statistics.

`boxplot.outline`	Print outliers in the boxplots?
`boxplot.ylab`	Label of the y-axis of a GOF plot.
`boxplot.main`	Main title of a GOF plot. Note: The same main title is used for all resulting diagrams. To obtain diagrams with varying main titles, plot each statistic separately using a new call of the `plot.btergm` method.
`boxplot.ylim`	Vertical limit of the boxplots. Only the maximum value must be provided because the minimum is always 0. Providing a vertical limit may be sensible if multiple models have to be compared side by side.
`roc`	Should the ROC curve(s) be plotted?
`pr`	Should the precision-recall curve(s) be plotted?
`rocpr.add`	Add the ROC and/or PR curve to an existing plot?
`rocpr.avg`	Averaging pattern for the ROC and PR curve(s) if multiple target time steps were used. Allowed values are `"none"` (plot all curves separately), `"horizontal"` (horizontal averaging), `"vertical"` (vertical averaging), and `"threshold"` (threshold (= cutoff) averaging). Note that while threshold averaging is always feasible, vertical and horizontal averaging are not well-defined if the graph cannot be represented as a function x->y and y->x, respectively. More information can be obtained from the help pages of the **ROCR** package, the functions of which are employed here.
`rocpr.spread`	When multiple target time steps are used and curve averaging is enabled, the variation around the average curve can be visualized as standard error bars (`"stderror"`), standard deviation bars (`"stddev"`), or by using box plots (`"boxplot"`). Note that the function plotCI, which is used internally by the **ROCR** package to draw error bars, might raise a warning if the spread of the curves at certain positions is 0. More details can be found in the documentation of the **ROCR** package, the functions of which are employed here.
`rocpr.lwd`	Line width of the ROC and PR curve(s).
`roc.main`	Main title of the ROC plot.
`roc.random`	Draw an additional ROC curve for random graphs with the same tie probability (as a reference value)?
`roc.col`	Color of the ROC curve(s). By default, a dark red color is used.
`roc.random.col`	Color of the random graph ROC curve.
`pr.main`	Main title of the ROC plot.
`pr.random`	Draw an additional PR curve for random graphs with the same tie probability (as a reference value)?
`pr.col`	Color of the precision-recall curve(s). By default, a blue color is used.
`pr.random.col`	Color of the random graph PR curve.
`pr.poly`	If a value of `0` is set, nothing special happens. If a value of `1` is set, a straight line is fitted through the PR curve and displayed. Values between `2` and `9` fit higher-order polynomial curves through the PR curve and display the resulting curve. This argument allows to check whether the imputation of the first precision value in the PR curve yielded a reasonable result (in case the value had to be imputed).

`classicgof`	Print tables with t-tests comparing observed and simulated networks, as in statnet?
`rocprgof`	Print tables with the area under the curve (AUC) for the PR and ROC curves at each time step of the observed network(s)?
`degeneracy`	Print tables with t-tests comparing the global statistics of observed and simulated networks for each observed time step? This is essentially a degeneracy check.
`object`	A `btergmgof` object created by calling the gof.btergm method.
`...`	Further arguments. These arguments are passed on to the plot command. This facilitates customization of the plots.

Details

These methods plot or print nicely formatted goodness-of-fit statistics from `btergmgof` objects to the R console. The typical workflow is to estimate a TERGM using the btergm function and save it as an object, then use the gof function to produce a `btergmgof` object, and finally use the functions and methods described here to show the output of this resulting object to assess the goodness of fit.

The `plot` method plots (1) boxplots of network statistics and draws the observed statistics as lines (classic statnet-like GOF boxplots) and (2) receiver operating characteristics (ROC) and precision recall (PR) curves.

The `print` and `summary` methods show (1) observed versus simulated network statistics (classic statnet-like GOF tables), (2) the area under the ROC curve (AUC) for each observed network, and (3) degeneracy checks by comparing global statistics of simulated versus observed networks.

See Also

xergm-package btergm simulate.btergm gof

`preprocess` *Preprocess lists of network matrices for use with btergm*

Description

Remove NAs and structural zeroes, create lagged covariates, and create memory terms.

Usage

```
preprocess(object, ..., lag = FALSE, covariate = FALSE,
    memory = c("no", "autoregression", "stability",
    "innovation"), na = NA, na.method = "fillmode",
    structzero = NA, structzero.method = "remove",
    verbose = FALSE)
```

Arguments

`object`	The object of interest that should be manipulated. The object can be a matrix, vector, data.frame, or a list of several objects of this kind. Usually, this is a list of network matrices (the dependent variable for the `btergm` function) or a list of covariates that needs to be adjusted given the missing observations at previous or future time steps. The name of the object is not allowed to match any of the argument names of this function (e.g., handing over an object with the name `memory` will cause an error message).
`...`	One or multiple other objects, separated by commas. These are used for comparison. If data are missing in these objects but not in the object of interest (given as the `object` argument), the missing data are also removed from the `object` of interest. The names of these objects are not allowed to match any of the argument names of this function (e.g., handing over an object with the name `memory` will cause an error message).
`lag`	Take into account missing or unobserved data at the previous or next time step? Use this argument in conjunction with the `covariate` argument. If `lag = TRUE` and `covariate = TRUE`, the first item in the list is adjusted to the dimensions of the second item, the second item is adjusted to the dimensions of the third item, and so forth, and the last item is omitted because it cannot be used as a covariate anyway. If `lag = TRUE` and `covariate = FALSE`, the first item is omitted, the second item is adjusted to the dimensions of the first item, the third item is adjusted to the dimensions of the second item, and so forth. If `lag = FALSE` (the `covariate` argument does not matter in this case), the first item is adjusted to the dimensions of the first item, the second item is adjusted to the dimensions of the second item etc. In all cases, missing data are first removed cross-sectionally across objects.
`covariate`	Is the object of interest a covariate or a dependent variable/network? This governs whether forward- or backward-looking missing data adjustment is used (when `lag = TRUE` is active). See the entry for `lag` for more details.
`memory`	Create a memory term which can be used as an edge covariate. Memory terms describe the temporal stability of the network. Several types of memory terms can be created. The default value `memory = "no"` will not create a memory term. `memory = "stability"` creates a term that captures dyadic stability between one time step and the next. `memory = "autoregression"` creates a lagged dependent network and is equivalent to `memory = "no"`. The result is a positive autoregression term, which is the lagged dependent network. This captures the stability of edges (but not of non-edges). `memory = "innovation"` measures the extent to which new edges are introduced over time ("edge innovation"). The `memory` argument only works when `lag = TRUE` and `covariate = TRUE` are also set.
`na`	A vector of values that are identified as missing data. These values can be removed or replaced (see the `handleMissings` function).
`na.method`	What should be done with missing data? Valid values are `remove` and `fillmode`. See the `handleMissings` function for details.
`structzero`	A vector of values that are identified as structural zeroes. These values can be removed or replaced (see the `handleMissings` function).

`structzero.method`	What should be done with structural zeroes? Valid values are `remove` and `fillmode`. See the `handleMissings` function for details.
`verbose`	Report the amount of missing data that were removed or replaced in each object?

Details

For use with the `btergm` function, lists of network matrices or covariates can be preprocessed using the `preprocess` function. The first object that is provided as an argument is adjusted to all remaining objects that are handed over via the `...` argument. The preprocess function can deal with constant or time-varying `matrix`, `vector` and `data.frame` objects. The user can specify whether the object of interest is a dependent network or a covariate and whether NA and structural zero removal should take into account missing data at previous or following time steps (`lag = TRUE`).

The preprocessing procedure consists of four steps: (1) remove or impute missing data and/or structural zeroes in each object at each time step; (2) cross-sectional adjustment of matrix dimensions (e.g., if the object of interest has more observations than another object at the current time step, the observations are dropped from the object of interest); (3) backward-looking adjustment of matrix dimensions (if the object of interest is a dependent variable/network and the `lag = TRUE` argument is used); and (4) forward-looking adjustment of matrix dimensions (if the object of interest is a covariate and the `lag = TRUE` argument is used). The last two steps can be helpful for building lagged or delayed covariates which have to take into account missing data at other time steps as well.

For example, if 26 nodes are present during the first time step, 25 during the second, 23 during the third, and 25 during the fourth time step, the arguments `covariate = TRUE` and `lag = TRUE` return a list of three objects with 25, 23 and 25 objects, respectively, because the covariate is assumed to lag one step behind the other objects provided after the first object.

If the `memory` argument is specified (see below for details on the allowed values), the function will not return a lagged network, but will attempt to create a memory term. A memory term is a dyadic covariate which captures the temporal process.

See Also

xergm-package handleMissings adjust

Examples

```
## Not run:
# This example illustrates the usefulness of the preprocess function.

# first network: nodes a to j present
mat1 <- rbinom(100, 1, 0.1)
mat1 <- matrix(mat1, nrow = 10)  # has 10 nodes
rownames(mat1) <- letters[1:10]
colnames(mat1) <- letters[1:10]

# second network: nodes c to n present
mat2 <- rbinom(144, 1, 0.1)
mat2 <- matrix(mat2, nrow = 12)  # has 12 nodes
rownames(mat2) <- letters[3:14]
colnames(mat2) <- letters[3:14]
```

```
# third network: nodes a and d to k present
mat3 <- rbinom(81, 1, 0.1)
mat3 <- matrix(mat3, nrow = 9)  # has 9 nodes
rownames(mat3) <- letters[c(1, 4:11)]
colnames(mat3) <- letters[c(1, 4:11)]

# fourth network: same as second matrix
mat4 <- mat2

networks <- list(mat1, mat2, mat3, mat4)

# btergm without cross-temporal dependencies:
model.1 <- btergm(networks ~ edges + mutual)
summary(model.1)

# When cross-temporal dependencies are specified, the dimensions
# of the matrices do not match. This would cause a problem for btergm:

\dontrun{
btergm(networks[2:4] ~ edges + mutual + edgecov(networks[1:3])) # ERROR!
}

# This is because the first network in the dependent network and the
# first network in the lagged covariate are expected to have the same
# dimensions (and also at the second and third time step, of course).

# Therefore, missing nodes in the covariate (here: {k, l, m, n} at t=1,
# {a} at t=2, and {c, l, m, n} at t=3) must be removed from the
# dependent network at t=2, t=3 and t=4 as well:

dep <- preprocess(networks, lag = TRUE, covariate = FALSE)

# This reduces the size of dep from 12 to 8 at t=2, from 9 to 8 at
# t=3, and from 12 to 8 at t=4, and it removes the first network from
# the list. Moreover, some nodes are present in the lagged covariate
# but not in the dependent network (that is, at the next time step).
# Therefore, node sets {a, b}, {c, l, m, n}, and {a} must be removed
# from the lagged covariate at t=1, t=2, and t=3, respectively, to make
# the dimensions compatible:

lag <- preprocess(networks, lag = TRUE, covariate = TRUE)

# To compare the dimensions of the original versus preprocessed
# dependent networks and covariates, try the following code:

cbind(
    "original_dep" = lapply(networks[2:4], nrow),
    "original_lag" = lapply(networks[1:3], nrow),
    "new_dep" = lapply(dep, nrow),
    "new_lag" = lapply(lag, nrow)
)
```

```
# The dependent networks were reduced from 12, 9 and 12 to 8, 8 and
# 8 nodes, and the lagged networks were reduced from 10, 12 and 9 to
# 8, 8 and 8 nodes, respectively. The lagged node sets are now
# compatible. To see this:

cbind(rownames(dep[[1]]), rownames(lag[[1]]))
cbind(rownames(dep[[2]]), rownames(lag[[2]]))
cbind(rownames(dep[[3]]), rownames(lag[[3]]))

# Note, however, that the composition still changes within each list
# across some of the time steps:

cbind(rownames(dep[[1]]), rownames(dep[[2]]), rownames(dep[[3]]))
cbind(rownames(lag[[1]]), rownames(lag[[2]]), rownames(lag[[3]]))

# We can now use the btergm function on the preprocessed lists:

model.2 <- btergm(dep ~ edges + mutual + edgecov(lag))
summary(model.2)

# The model can now be estimated because the current and lagged networks
# have the same node sets at each time step. The disadvantage of this
# approach is that some observations are lost. The advantage, however,
# is that cross-temporal theories can be tested.

# However, since the node sets still differ across time steps, ROC and
# PR curves cannot be estimated. This is true because a simulation from
# nodes {c ... j} cannot be compared to a target network with nodes
# {d ... k}. Therefore, the following command would compare the wrong
# sets of nodes to estimate prediction performance:

\dontrun{
gof.2 <- gof(model.2, classicgof = FALSE, rocprgof = TRUE) # PROBLEM!
}

# To solve this problem, the most obvious approach is to estimate the
# model at earlier time steps and compute the out-of-sample predictive
# performance only for the last network:

model.3 <- btergm(dep[1:2] ~ edges + mutual + edgecov(lag[1:2]))
gof.3 <- gof(model.3, target = dep[[3]], formula = dep[[3]] ~ edges +
    mutual + edgecov(lag[[3]]), classicgof = FALSE, rocprgof = TRUE)

# This models time steps 2 and 3 as a function of the lagged network
# at time steps 1 and 2, uses the resulting coefficients to predict
# the network at time step 4, and compares network 4 to simulations
# based on the coefficients from the previous time steps and the
# lagged network at the third time step. As the matrices within the
# third list item have identical node sets, predictive performance
# could be computed. The resulting ROC and PR curves can be plotted
# as follows:

plot(gof.3, boxplot = FALSE, pr = FALSE, roc.random = TRUE,
    ylab = "TPR/PPV", xlab = "FPR/TPR", roc.main = "ROC and PR")
plot(gof.3, boxplot = FALSE, roc = FALSE, pr.random = TRUE,
    rocpr.add = TRUE)
legend("right", legend = c("ROC", "ROC random graph", "PR",
    "PR random graph"), col = c("#bd0017", "#bd001744", "#5886be",
    "#5886be44"), lty = 1, lwd = 3)

# For another example with real-world data, see vignette("knecht")

## End(Not run)
```

`simulate.btergm` *Simulate new networks from btergm objects*

Description

Simulate new networks from btergm objects.

Usage

```
## S3 method for class 'btergm'
simulate(object, nsim = 1, seed = NULL,
    formula = object@formula, index = NULL,
    coef = object@coef, verbose = FALSE, ...)
```

Arguments

`object`	A `btergm` object, resulting from a call of the `btergm` function.
`nsim`	The number of networks to be simulated. Note that for values greater than one, a `network.list` object is returned, which can be indexed just like a `list` object, for example `mynetworks[[1]]` for the first simulated network in the object `mynetworks`.
`seed`	Random number integer seed. See set.seed.
`formula`	A model formula from which the new network(s) should be simulated. By default, the formula is taken from the btergm `object`.
`index`	Index of the network from which the new network(s) should be simulated. The index refers to the list of response networks on the left-hand side of the model formula. Note that more recent networks are located at the end of the list. By default, the first (= oldest) network is used.
`coef`	A vector of parameter estimates. By default, the coefficients are extracted from the given btergm `object`.
`verbose`	Print additional details while running the simulations?
`...`	Arbitrary further arguments are handed over to the `simulate.formula` function. For details, refer to the help page of the `simulate.formula` function.

Details

The `simulate.btergm` function is a wrapper for the `simulate.formula` function in the **ergm** package (see `help("simulate.formula")`). It can be used to simulate new networks from a `btergm` object. The `index` argument specifies from which of the original networks the new network(s) should be simulated. For example, if `object` is an estimation based on cosponsorship networks from the 99th to the 107th Congress (as in Desmarais and Cranmer 2012), and the cosponsorship network in the 108th Congress should be predicted using the `simulate.btergm` function, then the argument `index = 9` should be passed to the function because the network should be based on the 9th network in the list (that is, the latest network, which is the cosponsorship network for the 107th Congress). Note that all relevant objects (the networks and the covariates) must be present in the workspace (as was the case during the estimation of the model).

References

Desmarais, Bruce A. and Skyler J. Cranmer (2012): Statistical Mechanics of Networks: Estimation and Uncertainty. *Physica A* 391: 1865–1876.

See Also

xergm-package btergm gof

`tnam` *Fit (temporal) network autocorrelation models*

Description

Fit (temporal) network autocorrelation models.

Usage

```
tnam(formula, family = gaussian, re.node = FALSE,
    re.time = FALSE, time.linear = FALSE, time.quadratic = FALSE,
    center.y = FALSE, na.action = na.omit, ...)
```

Arguments

`formula`	A formula where the left-hand side specifies either a vector containing the outcome variable (for a cross-sectional model) or a list of such vectors (for modeling the outcome at multiple time steps) or a data frame with one time step per column (also for longitudinal models of behavior). The right-hand side of the formula consists of tnam-specific model terms like `netlag`, `structsim` and other terms which are described on the help page of tnam-terms.
`family`	The link function for fitting the generalized linear model or the mixed effects model, for example `gaussian` or `binomial`. The options are the same as in the `glm` and `glmer` functions. For details on the `family` argument, see the `family` help page.

`re.node`	If multiple time steps are present: should a random effect for the nodes be added to the model? This results in the estimation of a mixed effects model.
`re.time`	If multiple time steps are present: should a random effect for the time steps be added to the model? This results in the estimation of a mixed effects model.
`time.linear`	If multiple time steps are present: should a linear effect for time be added to the model? This can be estimated in the standard GLM framework.
`time.quadratic`	If multiple time steps are present: should a squared effect for time be added to the model? This can be estimated in the standard GLM framework.
`center.y`	Center the dependent variable by subtracting the mean from the actual value within each time step?
`na.action`	How should missings value be treated? By default, they are omitted. See the `na.omit` help page for details.
`...`	Further arguments that should be passed to the `glm`, `lmer`, or `glmer` function, which is used under the hood for estimating the model.

Details

The `tnam` function serves to estimate temporal or cross-sectional network autocorrelation models. Model terms such as spatial lags, temporal lags, spatio-temporal lags, centrality etc. can be specified in the `formula` argument. Details on the model terms can be found on the `tnam-terms` help page.

See Also

xergm-package tnam-terms preprocess knecht

Examples

```
# The following example models delinquency among adolescents at
# multiple time steps as a function of (1) their nodal attributes
# like sex or religion, (2) their peers' delinquency levels, (3)
# their own and their peers' past delinquency behavior, and (4)
# their structural position in the network. See ?knecht for
# details on the dataset.

library("statnet")
library("xergm")
data("knecht")

# prepare the dependent variable y
delinquency <- as.data.frame(delinquency)
rownames(delinquency) <- letters

# replace structural zeros (denoted as 10) and add node labels
friendship[[3]][friendship[[3]] == 10] <- NA
friendship[[4]][friendship[[4]] == 10] <- NA
for (i in 1:length(friendship)) {
  rownames(friendship[[i]]) <- letters
  colnames(friendship[[i]]) <- letters
}
```

```
# prepare the sex variable (can be a list of vectors)
sex <- as.numeric(demographics$sex)
names(sex) <- letters
sex <- list(sex, sex, sex, sex)

# prepare the religion variable (can also be a data frame)
rel <- as.numeric(demographics$religion)
rel <- data.frame(rel, rel, rel, rel)
rownames(rel) <- letters

# Estimate the model. The first term is the sex of the respondent,
# the second term is the religion of the respondent, the third
# term is the previous delinquency behavior of the respondent,
# the fourth term is the delinquency behavior of direct friends,
# the fifth term is the delinquency behavior of indirect friends
# at a path distance of 2, the sixth effect is the past delinquency
# of direct friends, the seventh term indicates whether the
# respondent has any contacts at all, and the last term captures
# the effect of the betweenness centrality of the respondent on
# his or her behavior. Apparently, previous behavior, being an
# isolate, and religion seem to have an effect on delinquency in
# this dataset. There is also a slight positive trend over time,
# and direct friends exert a minor effect (not significant).
# Note that a linear model may not be the best specification for
# modeling the ordered categorical delinquency variable, but it
# suffice here for illustration purposes.

model1 <- tnam(
    delinquency ~
    covariate(sex, coefname = "sex") +
    covariate(rel, coefname = "religion") +
    covariate(delinquency, lag = 1, exponent = 1) +
    netlag(delinquency, friendship) +
    netlag(delinquency, friendship, pathdist = 2, decay = 1) +
    netlag(delinquency, friendship, lag = 1) +
    degreedummy(friendship, deg = 0, reverse = TRUE) +
    centrality(friendship, type = "betweenness"),
    re.node = TRUE, time.linear = TRUE
)
summary(model1)

# for nice table output, use the texreg package
library("texreg")
screenreg(model1, naive = TRUE)
```

tnam-terms *Terms used in (Temporal) Network Autocorrelation Models (tnam)*

Description

The function tnam is used to fit (temporal) network autocorrelation models.

The function tnamdata can be used alternatively to create a data frame containing all the data ready for estimation. This may be useful when a non-standard model should be estimated, like a tobit model or a model with zero inflation, for example.

Both functions accept a formula containing several model terms. The model terms are themselves functions which can be called separately. For example, one model term is called `netlag`. This model term can be part of the formula handed over to the `tnam` function, or `netlag` can be called directly in order to create a single variable.

This help page describes the different model terms available in (temporal) network autocorrelation models. See the tnam help page for details on the model.

Usage

```
attribsim(y, attribute, match = FALSE, lag = 0,
    normalization = c("no", "row", "column"), center = FALSE,
    coefname = NULL)

centrality(networks, type = c("indegree", "outdegree", "freeman",
    "betweenness", "flow", "closeness", "eigenvector",
    "information", "load"), directed = TRUE, lag = 0,
    rescale = FALSE, center = FALSE, coefname = NULL, ...)

clustering(networks, directed = TRUE, lag = 0, center = FALSE,
    coefname = NULL, ...)

covariate(y, lag = 0, exponent = 1, center = FALSE,
    coefname = NULL)

degreedummy(networks, deg = 0, type = c("indegree", "outdegree",
    "freeman"), reverse = FALSE, directed = TRUE, lag = 0,
    center = FALSE, coefname = NULL, ...)

interact(x, y, lag = 0, center = FALSE, coefname = NULL)

netlag(y, networks, lag = 0, pathdist = 1, decay = pathdist^-1,
    normalization = c("no", "row", "column"),
    reciprocal = FALSE, center = FALSE, coefname = NULL, ...)

structsim(y, networks, lag = 0, method = c("euclidean",
    "minkowski", "jaccard", "binary", "hamming"), center = FALSE,
    coefname = NULL, ...)

weightlag(y, networks, lag = 0, normalization = c("no", "row",
    "column"), center = FALSE, coefname = NULL)
```

Arguments

`attribute`	A vector, list of vectors or data frame with the same dimensions as `y`. Based on this attribute, the similarity between nodes i and j will be calculated, and the resulting similarity matrix is used to weight the `y` variable.
`center`	Should the model term be centered? That is, should the mean of the variable be subtracted from the actual value at each time step?
`coefname`	An additional name that is used as part of the coefficient label for easier identification in the summary output of the model.
`decay`	For each value in `pathdist`, the `decay` argument specifies the relative importance. By default, a geometric decay is used, that is, the behavior of nodes at path distance 2 is counted only half as much as the behavior of adjacent nodes. Alternatively, if both are equally important, it is possible to write `pathdist = c(1, 2)` and `decay = c(1, 1)`.
`deg`	The degree (e.g., `deg = 2`) or degree range (e.g., `deg = 1:3`).
`directed`	Is the input matrix or network a directed network?
`exponent`	The exponent of a covariate. For example, `exponent = 2` creates a squared variable. This may be helpful for modeling non-linear effects or for modeling a quadratic behavior shape.
`lag`	The temporal lag. The default value 0 means there is no lag. A value of 1 would specify a single-period lag, that is, current behavior is modeled conditional on previous influence. A value of 2 would specify a two-period lag, that is, current behavior is modeled conditional on pre-previous influence, etc.
`match`	If `match = FALSE`, a similarity matrix is computed by subtracting node j's attribute value from node i's attribute value, standardizing the resulting distance between 0 and 1, and converting it into a similarity by subtracting it from 1. This similarity matrix is used as a weight matrix to compute a spatial lag. If `match = TRUE` is specified, the weight matrix contains values of 1 whenever node i and j have the same attribute value and 0 otherwise.
`method`	The distance function used for computing structural similarity. Possible values are `"euclidean"`, `"minkowski"`, `"jaccard"`, `"binary"`, and `"hamming"`.
`networks`	The network(s) for computing the peer influence, also known as the weight matrix. This can be a matrix or a network object (for a single time step) or a list of matrices or network objects (for multiple time steps).
`normalization`	Possible values: `"no"` for switching off normalization, `"row"` for row normalization of the weight matrix, and `"column"` for column normalization of the weight matrix.
`pathdist`	An integer or a vector of integers. For example, if `pathdist = 1` is used, this computes the sum of the behavior of adjacent nodes. If `pathdist = 2` is specified, this computes the effect of indirect paths of length 2 ("friends of friends"). If `pathdist = 1:2` is set, both directly connected nodes' behavior and the behavior of nodes at a path distance of 2 from the focal node are counted. See also the `decay` argument.
`reciprocal`	If `reciprocal = TRUE` is specified, only the behavior of nodes to which a reciprocal relation exists is counted (that is, a link in both directions).

`rescale`	Should the centrality index be rescaled between 0 and 1?
`reverse`	Reverse the selection of degrees. For example, when `deg = 0` and `reverse = FALSE` are specified, resulting values of 1 indicate that a node has no connections, whereas the combination `deg = 0` and `reverse = TRUE` results in the value 1 representing nodes which have a degree of at least 1.
`type`	The type of centrality measure. Possible values are `"indegree"`, `"outdegree"`, `"freeman"`, `"betweenness"`, `"flow"`, `"closeness"`, `"eigenvector"`, `"information"`, and `"load"`.
`x`	A variable that should be interacted with `y`. Either a vector or a list of vectors or another model term (this is the preferred way).
`y`	The outcome or behavior variable. Either a vector (for a single time step) or a list of vectors with named elements in each vector (for multiple time steps) or a data frame with row names where each column is one time step (for multiple time steps).
`...`	Additional arguments to be handed over to subroutines.

Model terms for `tnam`

`attribsim` *Spatial lag based on attribute similarity* The `attribsim` model term computes a similarity matrix based on the `attribute` argument and uses this similarity matrix to construct a spatial lag by multiplying the similarity matrix and the outcome vector `y`. The intuition behind this model term is that node i's behavior may be influenced by node j's behavior if nodes i and j are similar on another dimension. For example, if i and j both smoke while k does not smoke, j's alcohol consumption may affect i's alcohol consumption to a larger extent than node k's alcohol consumption. In this example, the `y` outcome variable is alcohol consumption and the `attribute` argument is smoking. If `match = FALSE`, the absolute similarity between i and j is computed by subtracting j's attribute value from i's attribute value and taking the absolute value to construct the similarity matrix. If `match = TRUE`, the function computes a matrix containing values of 1 if i and j have the same attribute value and 0 otherwise. A scenario where the `attribsim` model term makes sense is degree assortativity: if i and j have the same degree centrality, they may be inclined to learn from each other's behavior, even in the absence of a direct connection between them.

`centrality` *Node centrality* The `centrality` model term computes a centrality index for the nodes in a network or matrix. This can capture important structural effects because being central often implies certain constraints or opportunities more peripheral nodes do not have. For example, central nodes in a network of employees might be able to perform better.

clustering *Local clustering coefficient*, or *transitivity* The `clustering` model term computes the local clustering coefficient, which is also known as transitivity. This index has high values if the direct neighborhood of a node is densely interconnected. For example, if one's friends are friends with each other, this may have repercussions on ego's behavior.

covariate *Exogenous nodal covariate* The `covariate` model term adds an exogenous nodal covariate to the model. For example, when performance of employees is modeled, a covariate could be seniority of these employees. It is possible to add lagged covariates to model the effect of past nodal attributes on current behavior. Similarly, this model term can be used to add autoregressive terms, that is, the effect of previous behavior on current behavior.

degreedummy *Dummy variable for degree centrality values* The `degreedummy` model term controls for specific degree centralities or ranges of degree centrality. For example, do nodes with a degree of 0 (isolates) show different behavior than nodes who are connected? Or do nodes with a degree centrality larger than three exert different behavior?

interact *Interactions between other model terms* The `interact` model term adds an interaction effect between two other model terms by multiplying the result vectors of these two model terms. When using interaction terms, centering the result is recommended. Note that only the interaction term is created; the main effects must be introduced to the model using the other model terms.

netlag *Spatial network lag* The `netlag` model term captures the autocorrelation inherent in networks. For example, when political actors are members of a policy network, their success of achieving policy outcomes is not independent from each other. Most likely, being connected to policy winners increases the success rate. In many settings, indirect effects may be important as well: how does the behavior of my friends' friends affect my own behavior? In some contexts, spatio-temporal lags are useful: how does the past behavior of my friends affect my current behavior? The `netlag` model term is designed for binary networks because things like indirect effects, restriction to reciprocal dyads, decay of indirect relations etc. is possible. For weighted networks, the `weightlag` term is recommended. If no other arguments are specified and loops are absent and a binary matrix is used, both model terms produce the same results.

structsim *Structural similarity* The `structsim` model term computes the structural similarity with other nodes in the network and multiplies this similarity matrix with the outcome variable. The intuition is that behavior is sometimes affected by comparison with structurally similar nodes. For example, a worker may be impressed by the performance of other workers who are embedded in the same team or who report to the same bosses. As with the other model terms, temporal lags are possible.

weightlag *Weighted spatial lag* The `weightlag` model term captures spatial autocorrelation in weighted networks. For example, the GDP per capita of a country may be affected by the GDP of proximate other countries or by the GDP of trade partners. In these cases, indirect contacts etc. do not make any sense, therefore the distinction between the `weightlag` and the `netlag` model term. The weight matrix is multiplied by the outcome variable, possibly after row or column normalization.

See Also

xergm-package tnam tnamdata preprocess knecht

Index