Package 'btergm'

July 22, 2025

Version 1.11.1

Date 2025-03-19

Title Temporal Exponential Random Graph Models by Bootstrapped Pseudolikelihood

Description Temporal Exponential Random Graph Models (TERGM) estimated by maximum pseudolikelihood with bootstrapped confidence intervals or Markov Chain Monte Carlo maximum likelihood. Goodness of fit assessment for ERGMs, TERGMs, and SAOMs. Micro-level interpretation of ERGMs and TERGMs. The methods are described in Leifeld, Cranmer and Desmarais (2018), JStatSoft <doi:10.18637/jss.v083.i06>.

URL https://github.com/leifeld/btergm

Encoding UTF-8

Imports stats, utils, methods, graphics, network (>= 1.19.0), sna (>= 2.8), ergm (>= 4.8.1), parallel, Matrix (>= 1.3.2), boot (>= 1.3.17), coda (>= 0.18.1), ROCR (>= 1.0.7), igraph (>= 2.1.2), statnet.common (>= 4.11.0)

Suggests fastglm (>= 0.0.1), speedglm (>= 0.3.1), testthat, Bergm (>= 5.0.7), RSiena (>= 1.0.12.232), ggplot2 (>= 2.0.0)

Depends R (>= 3.5)

License GPL (>= 2)

RoxygenNote 7.3.2

NeedsCompilation no

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

Maintainer Philip Leifeld <philip.leifeld@manchester.ac.uk>

Repository CRAN

Date/Publication 2025-03-19 03:20:02 UTC

Contents

`btergm-package` *Temporal Exponential Random Graph Models by Bootstrapped Pseudolikelihood*

Description

Temporal Exponential Random Graph Models by Bootstrapped Pseudolikelihood.

Details

Temporal Exponential Random Graph Models (TERGM) estimated by maximum pseudolikelihood with bootstrapped confidence intervals, Markov Chain Monte Carlo maximum likelihood, or Bayesian estimation. Goodness of fit assessment for ERGMs, TERGMs, and SAOMs. Micro-level interpretation of ERGMs and TERGMs.

The **btergm** package implements TERGMs with MPLE and bootstrapped confidence intervals (`btergm` function), MCMC MLE (`mtergm` function), or Bayesian estimation (`tbergm` function).

Goodness of fit assessment for ERGMs, TERGMs, SAOMs, and dyadic independence models is possible with the generic `gof` function and its various methods defined here in the **btergm** package. New networks can be simulated from TERGMs using the `simulate.btergm` function. The package also implements micro-level interpretation for ERGMs and TERGMs using the `interpret` function. Furthermore, the package contains the `chemnet` and `knecht` (T)ERGM datasets. To display citation information, type `citation("btergm")`.

Author(s)

Philip Leifeld, Skyler J. Cranmer, Bruce A. Desmarais

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. doi: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. doi: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*.

Leifeld, Philip, Skyler J. Cranmer and Bruce A. Desmarais (2018): Temporal Exponential Random Graph Models with btergm: Estimation and Bootstrap Confidence Intervals. *Journal of Statistical Software* 83(6): 1–36. doi:10.18637/jss.v083.i06.

See Also

Useful links:

- `https://github.com/leifeld/btergm`

`adjust` *Adjust the dimensions of a source object to the dimensions of a target object*

Description

Adjust the dimensions of a source object to the dimensions of a target object.

Usage

```
adjust(
  source,
  target,
  remove = TRUE,
  add = TRUE,
  value = NA,
  returnlabels = FALSE
)
```

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?
value	The value to be inserted if a new row or column is added. By default, new cells are filled with NA values, but other sensible values may include -Inf or 0.
returnlabels	Return a list of added and removed row and column labels rather than the actual matrix, vector, or network 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), and vectors are supported.

Note that it is not necessary to use this function to preprocess any data before estimating a TERGM. The estimation functions in the **btergm** package call this function repeatedly to mutually adjust all data as needed.

See Also

handleMissings

Examples

```
# create sociomatrix a with 13 vertices a to m
vertices <- letters[1:13]
a <- matrix(rbinom(length(vertices)^2, 1, 0.1), nrow = length(vertices))
rownames(a) <- colnames(a) <- vertices

# create matrix b with the same vertices except f and k, but additional n
vertices <- c(vertices[-c(6, 11)], "n")
b <- matrix(rbinom(length(vertices)^2, 1, 0.1), nrow = length(vertices))
rownames(b) <- colnames(b) <- vertices

# check dimensions
dim(a)  # 13 x 13
dim(b)  # 12 x 12

# adjust a to b: add n and fill up with NAs; remove f and k
adjust(a, b, add = TRUE, remove = TRUE)
```

```
## Not run:
# more complex example with additional attributes stored in the network
# object; convert a to network object with additional vertex and network
# attributes
nw <- network(a)
vertices <- letters[1:13]
nwattrib1 <- matrix(rbinom(length(vertices)^2, 1, 0.1),
                    nrow = length(vertices))
nwattrib2 <- nwattrib1
rownames(nwattrib1) <- colnames(nwattrib1) <- vertices
set.network.attribute(nw, "nwattrib1", nwattrib1)
set.network.attribute(nw, "nwattrib2", nwattrib2)
set.vertex.attribute(nw, "vattrib", 1:length(vertices))

# check presence of the two attributes
list.network.attributes(nw)  # nwattrib1 and nwattrib2 are listed
get.network.attribute(nw, "nwattrib1")  # returns sociomatrix with labels
get.network.attribute(nw, "nwattrib2")  # returns sociomatrix without labels
list.vertex.attributes(nw)  # vattrib is listed
get.vertex.attribute(nw, "vattrib")  # returns numeric vector 1:13

# adjust the network including the two attributes
nw.adjusted <- adjust(nw, b, add = TRUE, remove = TRUE)
as.matrix(nw.adjusted)  # note that the order of nodes may have changed
get.network.attribute(nw.adjusted, "nwattrib1")  # returns adjusted matrix
get.network.attribute(nw.adjusted, "nwattrib2")  # returns adjusted matrix
get.vertex.attribute(nw.adjusted, "vattrib")  # returns adjusted vector

## End(Not run)
```

`alliances` *Longitudinal international defense alliance network, 1981–2000*

Description

Longitudinal international defense alliance network, 1981–2000.

Format

`allyNet` is a list of network objects at 20 time points, 1981–2000, containing undirected defense alliance networks. In addition to the alliance ties, each network object contains three vertex attributes. `cinc` is the "CINC" or Composite Index of National Capability score (see https://correlatesofwar.org/data-sets/national-material-capabilities/). `polity` is the "polity score" of each country in the respective year. Quoting the online description, "the Polity Score captures this regime authority spectrum on a 21-point scale ranging from -10 (hereditary monarchy) to +10 (consolidated democracy)," (see https://www.systemicpeace.org/polityproject.html). `year` is simply the year recorded as a vertex attribute.

`contigMat` is a 164 x 164 binary matrix in which a 1 indicates that two countries share a border.

`lNet` is a list of 20 matrices. Each element is the adjacency matrix from the previous year. This is used to model memory in the ties.

`LSP` is a list of 20 matrices. Each element is a matrix recording the number of shared partners between countries in the alliance network from the previous year.

`warNet` is a list of 20 matrices. Each element is a binary matrix that indicates whether two states were in a militarized interstate dispute in the respective year.

Details

The alliances dataset contains the international defense alliance network among 164 countries, covering the years 1981–2000. In addition to the yearly defense alliance network, it contains data on military capabilities, governing regime type, geographic contiguity and international conflict. This is an excerpt from a dataset that has been used in two published analyses. The full dataset (Cranmer, Desmarais and Menninga 2012; Cranmer, Desmarais and Kirkland 2012) contains a large number of countries and a much longer time series.

Source

The data were gathered by Skyler Cranmer and Bruce Desmarais in the process of writing Cranmer, Desmarais and Menninga (2012) and Cranmer, Desmarais and Kirkland (2012).

Permission to redistribute this dataset along with this package was granted by Skyler Cranmer and Bruce Desmarais on December 15, 2015. Questions about the data should be directed to them.

References

Cranmer, Skyler J., Bruce A. Desmarais, and Justin H. Kirkland (2012): Toward a Network Theory of Alliance Formation. *International Interactions* 38(3): 295–324. doi:10.1080/03050629.2012.677741.

Cranmer, Skyler J., Bruce A. Desmarais, and Elizabeth Menninga (2012): Complex Dependencies in the Alliance Network. *International Interactions* 29(3): 279–313. doi:10.1177/0738894212443446.

Examples

```
## Not run:
data("alliances")

# btergm formulas look very similar to ERGM formulas.
# Note the R argument; usually want R > 1000.
# Here it is set to 50 to limit computation time.
# First, set the seed for replicability.
set.seed(123)
model <- btergm(allyNet ~ edges + gwesp(0, fixed = TRUE)
    + edgecov(lNet) + edgecov(LSP) + edgecov(warNet)
    + nodecov("polity") + nodecov("cinc") + absdiff("polity")
    + absdiff("cinc") + edgecov(contigMat) + nodecov("year"),
    R = 50)

# View estimates and confidence intervals.
summary(model)

# Evaluate model fit. Simulate 100 networks for each time point.
```

```
# Calculate edgewise shared partners, degree and geodesic distance
# distance distributions.
alliance_gof <- gof(model, statistics = c(deg, esp, geodesic))

# Plot goodness of fit.
plot(alliance_gof)

## End(Not run)
```

`btergm` *Estimate a TERGM by MPLE with temporal bootstrapping*

Description

Estimate a TERGM by MPLE with temporal bootstrapping.

Usage

```
btergm(
  formula,
  R = 500,
  offset = FALSE,
  returndata = FALSE,
  parallel = c("no", "multicore", "snow"),
  ncpus = 1,
  cl = NULL,
  control.ergm = NULL,
  usefastglm = FALSE,
  verbose = TRUE,
  ...
)
```

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.
`offset`	If `offset = TRUE` is set, a list of offset matrices (one for each time step) with structural zeros is handed over to the pseudolikelihood preparation routine. The offset matrices contain structural zeros where either the dependent networks or

any of the covariates have missing nodes (if `auto.adjust = TRUE` is used). All matrices and network objects are inflated to the dimensions of the largest object, and the offset matrices inform the estimation preparation routine which dyads are constrained to be absent. After MPLE data preparation, the dyads with these structural zeros are removed before the GLM is estimated. If `offset = FALSE` is set, all nodes that are not present across all covariates and networks within a time step are removed completely from the respective object(s) before the change statistics are computed for all remaining dyads.

`returndata` Return the processed input data instead of estimating and returning the model? In the `btergm` case, this will return a data frame with the dyads of the dependent variable/network and the change statistics for all covariates. In the `mtergm` case, this will return a list object with the blockdiagonal network object for the dependent variable and blockdiagonal matrices for all dyadic covariates and the offset matrix for the structural zeros.

`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()` after loading the **parallel** package. 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 PSOCK cluster is created temporarily on the local machine.

`control.ergm` ergm controls for `ergmMPLE` calls. See `control.ergm` for details.

`usefastglm` Controls whether to use the `fastglm` estimation routine from the **fastglm** package with `method = 3`. Defaults to `FALSE` (and then uses `speedglm.wfit` instead if available).

`verbose` Print details about data preprocessing and estimation settings.

`...` Further arguments to be handed over to the `boot` function.

Details

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

MLE but only asymptotically unbiased the longer the time series of networks because it uses temporal bootstrapping to correct the standard errors.

Author(s)

Philip Leifeld, Skyler J. Cranmer, Bruce A. Desmarais

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. doi: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. doi: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*.

Leifeld, Philip and Skyler J. Cranmer (2019): A Theoretical and Empirical Comparison of the Temporal Exponential Random Graph Model and the Stochastic Actor-Oriented Model. *Network Science* 7(1): 20-51. doi:10.1017/nws.2018.26.

Leifeld, Philip, Skyler J. Cranmer and Bruce A. Desmarais (2017): Temporal Exponential Random Graph Models with btergm: Estimation and Bootstrap Confidence Intervals. *Journal of Statistical Software* 83(6): 1-36. doi:10.18637/jss.v083.i06.

See Also

mtergm tbergm

Examples

```
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::network(mat)  # create network object
  networks[[i]] <- nw          # add network to the list
}

covariates <- list()
for (i in 1:10) {              # create 10 matrices as covariate
  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 examples with real data, see help("knecht") or help("alliances").
```

```
# Examples for parallel processing:

# Some preliminaries:
# - "Forking" means running the code on multiple cores in the same
#   computer. It's fast but consumes a lot of memory because all
#   objects are copied for each node. It's also restricted to
#   cores within a physical computer, i.e. no distribution over a
#   network or cluster. Forking does not work on Windows systems.
# - "MPI" is a protocol for distributing computations over many
#   cores, often across multiple physical computers/nodes. MPI
#   is fast and can distribute the work across hundreds of nodes
#   (but remember that R can handle a maximum of 128 connections,
#   which includes file access and parallel connections). However,
#   it requires that the Rmpi package is installed and that an MPI
#   server is running (e.g., OpenMPI).
# - "PSOCK" is a TCP-based protocol. It can also distribute the
#   work to many cores across nodes (like MPI). The advantage of
#   PSOCK is that it can as well make use of multiple nodes within
#   the same node or desktop computer (as with forking) but without
#   consuming too much additional memory. However, the drawback is
#   that it is not as fast as MPI or forking.
# The following code provides examples for these three scenarios.

# btergm works with clusters via the parallel package. That is, the
# user can create a cluster object (of type "PSOCK", "MPI", or
# "FORK") and supply it to the 'cl' argument of the 'btergm'
# function. If no cluster object is provided, btergm will try to
# create a temporary PSOCK cluster (if parallel = "snow") or it
# will use forking (if parallel = "multicore").

## Not run:
# To use a PSOCK cluster without providing an explicit cluster
# object:
require("parallel")
fit <- btergm(networks ~ edges + istar(2) + edgecov(covariates),
              R = 100, parallel = "snow", ncpus = 25)

# Equivalently, a PSOCK cluster can be provided as follows:
require("parallel")
cores <- 25
cl <- makeCluster(cores, type = "PSOCK")
fit <- btergm(networks ~ edges + istar(2) + edgecov(covariates),
              R = 100, parallel = "snow", ncpus = cores, cl = cl)
stopCluster(cl)

# Forking (without supplying a cluster object) can be used as
# follows.
require("parallel")
cores <- 25
fit <- btergm(networks ~ edges + istar(2) + edgecov(covariates),
              R = 100, parallel = "multicore", ncpus = cores)
```

```
stopCluster(cl)

# Forking (by providing a cluster object) works as follows:
require("parallel")
cores <- 25
cl <- makeCluster(cores, type = "FORK")
fit <- btergm(networks ~ edges + istar(2) + edgecov(covariates),
              R = 100, parallel = "snow", ncpus = cores, cl = cl)
stopCluster(cl)

# To use MPI, a cluster object MUST be created beforehand. In
# this example, a MOAB HPC server is used. It stores the number of
# available cores as a system option:
require("parallel")
cores <- as.numeric(Sys.getenv("MOAB_PROCCOUNT"))
cl <- makeCluster(cores, type = "MPI")
fit <- btergm(networks ~ edges + istar(2) + edgecov(covariates),
              R = 100, parallel = "snow", ncpus = cores, cl = cl)
stopCluster(cl)

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

# 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 *An S4 class to represent a fitted TERGM by bootstrapped MPLE*

Description

An S4 class to represent a fitted TERGM by bootstrapped MPLE.

Show the coefficients of a `btergm` object.

Usage

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

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

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

btergm.se(object, print = FALSE)

## S4 method for signature 'btergm'
confint(object, parm, level = 0.95, type = "perc", invlogit = FALSE, ...)

timesteps.btergm(object)

## S4 method for signature 'btergm'
summary(object, level = 0.95, type = "perc", invlogit = FALSE, ...)
```

Arguments

`object`	A `btergm` object.
`invlogit`	Apply inverse logit transformation to the estimates and/or confidence intervals? That is, $\frac{1}{1+\exp(-x)}$, where x is the respective value.
`...`	Further arguments to be passed through to the `confint` function.
`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`)?
`parm`	Parameters (specified by integer position or character string).
`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`, and `0.9`.
`type`	Type of confidence interval, e.g., basic bootstrap interval (`type = "basic"`), percentile-based interval (`type = "perc"`, which is the default option), or bias-adjusted and accelerated confidence interval (`type = "bca"`). All options from the `type` argument of the boot.ci function in the boot package can be used to generate confidence intervals.

Details

btergm objects result from the estimation of a bootstrapped TERGM via the btergm function. 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.

Functions

- coef(btergm): Return the coefficients of a btergm object.
- nobs(btergm): Return the number of observations saved in a btergm object.
- btergm.se(): Create a coefficient table from a btergm object

  Create a coefficient matrix with standard errors and p-values.

  This function can create a coefficient matrix with coefficients, standard errors, z-scores, and p-values, based on a fitted btergm object. 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@boot$t[, 1]) for the first model term in the btergm object called x).
- confint(btergm): Return the confidence intervals for estimates in a btergm object.
- timesteps.btergm(): Return the number of time steps saved in a btergm object.
- summary(btergm): Summary of a fitted btergm object.

Slots

coef Object of class "numeric". The coefficients.

boot 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).

formula2 The revised formula with the object references after applying the tergmprepare function.

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.

auto.adjust Object of class "logical". Indicates whether automatic adjustment of dimensions was done before estimation.

offset Object of class "logical". Indicates whether an offset matrix with structural zeros was used.

directed Object of class "logical". Are the dependent networks directed?

`bipartite` Object of class `"logical"`. Are the dependent networks bipartite?

`nvertices` Number of vertices.

`data` The data after processing by the `tergmprepare` function.

See Also

Other tergm-classes: `createBtergm()`, `createMtergm()`, `createTbergm()`, `mtergm-class`, `tbergm-class`

`checkdegeneracy` *Check for degeneracy in fitted TERGMs*

Description

Check for degeneracy in fitted TERGMs.

Usage

```
checkdegeneracy(object, ...)

## S4 method for signature 'mtergm'
checkdegeneracy(object, ...)

## S4 method for signature 'btergm'
checkdegeneracy(
  object,
  nsim = 1000,
  MCMC.interval = 1000,
  MCMC.burnin = 10000,
  verbose = FALSE
)

## S3 method for class 'degeneracy'
print(
  x,
  center = FALSE,
  t = 1:length(x$sim),
  terms = 1:length(x$target.stats[[1]]),
  ...
)

## S3 method for class 'degeneracy'
plot(
  x,
  center = TRUE,
  t = 1:length(x$sim),
  terms = 1:length(x$target.stats[[1]]),
```

```
  vbar = TRUE,
  main = NULL,
  xlab = NULL,
  target.col = "red",
  target.lwd = 3,
  ...
)
```

Arguments

object	A btergm or mtergm object, as estimated using the btergm or mtergm function.
...	Arbitrary further arguments for subroutines.
nsim	The number of networks to be simulated at each time step. This number should be sufficiently large for a meaningful comparison. If possible, much more than 1,000 simulations.
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 1000, which means that every 1000th 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.
verbose	Print details?
x	A degeneracy object created by the checkdegeneracy function.
center	If TRUE, print/plot the simulated minus the target statistics, with an expected value of 0 in a non-degenerate model. If FALSE, print/plot the distribution of simulated statistics and show the target statistic separately.
t	Time indices to include, e.g., t = 2:4 for time steps 2 to 4.
terms	Indices of the model terms to include, e.g., terms = 1:3 includes the first three statistics.
vbar	Show vertical bar for target statistic in histogram.
main	Main title of the plot.
xlab	Label on the x-axis. Defaults to the name of the statistic.
target.col	Color of the vertical bar for the target statistic. Defaults to red.
target.lwd	Line width of the vertical bar for the target statistic. Defaults to 3.

Details

The methods for the generic `degeneracy` function implement a degeneracy check for `btergm` and `mtergm` objects. For `btergm`, this works 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 `mtergm`, the `mcmc.diagnostics` function from the **ergm** package is used.

Value

A list with target statistics and simulations.

References

Hanneke, Steve, Wenjie Fu and Eric P. Xing (2010): Discrete Temporal Models of Social Networks. *Electronic Journal of Statistics* 4: 585–605. doi:10.1214/09EJS548.

Leifeld, Philip, Skyler J. Cranmer and Bruce A. Desmarais (2018): Temporal Exponential Random Graph Models with btergm: Estimation and Bootstrap Confidence Intervals. *Journal of Statistical Software* 83(6): 1-36. doi:10.18637/jss.v083.i06.

`chemnet` *German Toxic Chemicals Policy Network in the 1980s (Volker Schneider)*

Description

German Toxic Chemicals Policy Network in the 1980s (Volker Schneider).

Format

`pol` is a directed 30 x 30 adjancency matrix indicating which row actor sends political/strategic information to which column actor. `1` indicates an information exchange tie, and `0` indicates the absence of a network tie.

`scito` is a directed 30 x 30 adjacency matrix indicating which row actor sends technical/scientific information to which column actor. `1` indicates an information exchange tie, and `0` indicates the absence of a network tie. In contrast to political/strategic information exchange, two separate survey questions were asked about technical/scientific information exchange: sending information, and receiving information. The two matrices contain the same relation but one time from the sender's perspective and one time from the receiver's perspective. By combining the two matrices, one can create a "confirmed" technical/scientific information exchange relation. The `scito` matrix contains ties from the sender's perspective.

`scifrom` is a directed 30 x 30 adjacency matrix indicating which row actor receives technical/scientific information from which column actor. `1` indicates an information exchange tie, and `0` indicates the absence of a network tie. In contrast to political/strategic information exchange, two separate survey questions were asked about technical/scientific information exchange: sending information, and receiving information. The two matrices contain the same relation but one

time from the sender's perspective and one time from the receiver's perspective. By combining the two matrices, one can create a "confirmed" technical/scientific information exchange relation. The `scifrom` matrix contains ties from the receiver's perspective.

`infrep` is a directed 30 x 30 adjancency matrix indicating which row actor deems which column actor "particularly influential". `1` indicates such a tie, and `0` indicates the absence of an influence attribution tie.

`committee` is a 30 x 20 two-mode (bipartite) network matrix indicating which row actor is a member of which policy committee/forum (as indicated by the column labels). `1` indicates a membership tie, and `0` indicates non-membership.

`types` is a one-column data.frame where the `type` variable contains the actor type of each node. The following values are possible:

- `gov` (government actor, e.g., a federal ministry)
- `ig` (interest group)
- `io` (international organization)
- `par` (political party)
- `sci` (scientific organization)

`intpos` is a 30 x 6 matrix containing the interest positions of the 30 political actors on the six most salient political issues related to a pending new chemicals law. `-1` indicates a negative stance, i.e., the actor rejects the proposal; `1` indicates a positive stance, i.e., the actor supports the proposal; and `0` indicates a neutral or absent opinion.

Details

The chemnet dataset contains network and attribute data and for the 30 most influential political actors with regard to toxic chemicals regulation in Germany in 1983/1984. While the original dataset contains up to 47 actors, this dataset contains the "complete influence core" of mutually relevant actors. The data are cross-sectional. There are no missing data; the response rate was 100 percent. Volker Schneider (University of Konstanz) collected this dataset for his dissertation (Schneider 1988). The dataset was later re-used for a journal publication on information exchange in policy networks (Leifeld and Schneider 2012).

The chemnet dataset contains network relations on political/strategic and technical/scientific information exchange, influence attribution, and membership in policy committees/forums, as well as nodal attributes on the actor type and opinions about the six most salient issues related to the political process that was leading to a new chemicals law at the time being.

Source

The data were collected using paper-based questionnaires. The questionnaires were administered in personal interviews (PAPI). Further information, including the actual survey, data on additional actors, the full names of the policy committees/forums, and the full list of unabbreviated actor names can be found online at doi:10.7910/DVN/ONDFVJ in the replication archive of Leifeld and Schneider (2012).

- Replication archive: doi:10.7910/DVN/ONDFVJ
- AJPS publication: doi:10.1111/j.15405907.2011.00580.x

The dataset is publicly available. Questions about the data or the original study should be directed to Volker Schneider <volker.schneider@uni-konstanz.de>, the author of the original study and person who collected the data.

References

Leifeld, Philip and Volker Schneider (2012): Information Exchange in Policy Networks. *American Journal of Political Science* 53(3): 731–744. doi:10.1111/j.15405907.2011.00580.x.

Schneider, Volker (1988): *Politiknetzwerke der Chemikalienkontrolle. Eine Analyse einer transnationalen Politikentwicklung*. Walter de Gruyter: Berlin/New York.

Schneider, Volker and Philip Leifeld (2009): Ueberzeugungssysteme, Diskursnetzwerke und politische Kommunikation: Ein zweiter Blick auf die deutsche Chemikalienkontrolle der 1980er Jahre. In: Volker Schneider, Frank Janning, Philip Leifeld and Thomas Malang (editors): *Politiknetzwerke. Modelle, Anwendungen und Visualisierungen*. Pages 139–158. Wiesbaden: VS Verlag fuer Sozialwissenschaften.

Examples

```
## Not run:
# Replication code for Leifeld and Schneider (2012), AJPS.
# Note that the estimates can only be reproduced approximately
# due to internal changes in the statnet package.

# preparatory steps
library("network")
library("sna")
library("ergm")
library("btergm")
library("texreg")
seed <- 12345
set.seed(seed)
data("chemnet")

# create confirmed network relation
sci <- scito * t(scifrom)  # equation 1 in the AJPS paper
prefsim <- dist(intpos, method = "euclidean")  # equation 2
prefsim <- max(prefsim) - prefsim  # equation 3
prefsim <- as.matrix(prefsim)
committee <- committee %*% t(committee)  # equation 4
diag(committee) <- 0 # the diagonal has no meaning
types <- types[, 1]  # convert to vector

# create network objects and store attributes
nw.pol <- network(pol) # political/stratgic information exchange
set.vertex.attribute(nw.pol, "orgtype", types)
set.vertex.attribute(nw.pol, "betweenness",
    betweenness(nw.pol)) # centrality

nw.sci <- network(sci) # technical/scientific information exchange
set.vertex.attribute(nw.sci, "orgtype", types)
set.vertex.attribute(nw.sci, "betweenness",
```

```
    betweenness(nw.sci)) # centrality

# ERGM: model 1 in the AJPS paper; only preference similarity
model1 <- ergm(nw.pol ~ edges + edgecov(prefsim),
    control = control.ergm(seed = seed))
summary(model1)

# ERGM: model 2 in the AJPS paper; complete model
model2 <- ergm(nw.pol ~
    edges +
    edgecov(prefsim) +
    mutual +
    nodemix("orgtype", base = -7) +
    nodeifactor("orgtype", base = -1) +
    nodeofactor("orgtype", base = -5) +
    edgecov(committee) +
    edgecov(nw.sci) +
    edgecov(infrep) +
    gwesp(0.1, fixed = TRUE) +
    gwdsp(0.1, fixed = TRUE),
    control = control.ergm(seed = seed)
)
summary(model2)

# ERGM: model 3 in the AJPS paper; only preference similarity
model3 <- ergm(nw.sci ~ edges + edgecov(prefsim),
    control = control.ergm(seed = seed))
summary(model3)

# ERGM: model 4 in the AJPS paper; complete model
model4 <- ergm(nw.sci ~
    edges +
    edgecov(prefsim) +
    mutual +
    nodemix("orgtype", base = -7) +
    nodeifactor("orgtype", base = -1) +
    nodeofactor("orgtype", base = -5) +
    edgecov(committee) +
    edgecov(nw.pol) +
    edgecov(infrep) +
    gwesp(0.1, fixed = TRUE) +
    gwdsp(0.1, fixed = TRUE),
    control = control.ergm(seed = seed)
)
summary(model4)

# regression table using the texreg package
screenreg(list(model1, model2, model3, model4))

# goodness of fit using the btergm package
gof2 <- gof(model2, roc = FALSE, pr = FALSE)
gof2  # print gof output
plot(gof2)  # visual inspection of GOF
```

```
gof4 <- gof(model4, roc = FALSE, pr = FALSE)
gof4
plot(gof4)

# MCMC diagnostics
pdf("diagnostics2.pdf")
mcmc.diagnostics(model2)
dev.off()

pdf("diagnostics4.pdf")
mcmc.diagnostics(model4)
dev.off()

## End(Not run)
```

---

coauthor *Swiss political science co-authorship network 2013*

Description

Swiss political science co-authorship network 2013

Format

`ch_coaut` is an undirected, weighted 156 x 156 adjancency matrix indicating how many publications political scientist in Switzerland shared with each other as reported in late 2013, including only postdoctoral and professorial political scientists affiliated with research institutes or universities. The exact edge weight should be treated with caution because some publications were counted multiple times because they were reported by multiple co-authors. The diagonal contains the number of publications of the respective author. Leifeld and Ingold (2016) describe the data collection process in more detail.

`ch_nodeattr` is a data frame with node attributes/variables for the 156 researchers, in the same alphabetical row order as the network matrix. The first twelve columns with column labels starting with "inst_" are affiliations with different institutions (1 = affiliated; 0 = no affiliation). The next seven columns with column labels starting with "city_" contain the locations of the researchers' institutional affiliations. The "phdyear" column contains the self-reported year of obtaining the PhD, and the "birthyear" column contains the self-reported or publicly available year of birth; these two variables contain many missing values. The "status" column indicates whether a researcher was listed as a professor or as having postdoctoral or other non-professorial status at the time. "chairtitle" is the name of the chair or research group the researcher reported to be a member of. "num_publications" is the total number of publications, "num_articles" the number of journal articles among them, "num_books" the number of books among them, "share_articles" the percentage of journal articles among the publications, and "share_books" the percentage of monographs and edited volumes among the publications. The four columns with column names starting with "lang_" contain the relative shares of English, French, German, Italian, and other languages among the publications of the researcher. The column "share_en_articles" contains the percentage of English journal articles among all

publications of the researcher. "male" is a dummy variable indicating whether the author is male (1) or female (0). The variables contained here are described in Leifeld (2018).

`ch_dist100km` is a 156 x 156 matrix containing the geographical distance between any two researchers measured in units of 100km (for a reasonable scaling of coefficients in a statistical model), computed over the latitude and longitude of their main institutional affiliations. The measure is included in Leifeld (2018).

`ch_en_article_sim` is a 156 x 156 matrix containing the similarity between any two researchers in terms of the share of their work that is published in English and as journal articles. Values closer to 1.0 indicate that two researchers were similar in their language and publication type portfolio while values closer to 0 indicate that they were relatively dissimilar. Only extra-dyadic publications were counted in establishing this similarity. I.e., if researcher A and B co-authored, their joint publications were not included in establishing their English article share similarity. This was done to reduce endogeneity/reverse causality when modeling co-authorship as a function of English article share similarity. The measure is described in Leifeld (2018).

`ch_topicsim` is a 156 x 156 topic similarity matrix for the researchers. Topic similarities were computed by taking into account all words in the publication titles of any two researchers, excluding the publications they published as co-authors (i.e., only extra-dyadic publications, to reduce endogeneity/reverse causality in modeling co-authorship ties as a function of topic similarity). Topic similarity was established by computing the cosine similarity between the tf-idf scores for the title words of any two researchers (i.e., a vector space model). Leifeld (2018) contains more details on this procedure.

Details

The Swiss political science co-authorship network 2013 dataset contains the co-authorship network of all political scientists at Swiss universities and research institutes in late 2013. The data are described in Leifeld and Ingold (2016) and Leifeld (2018). The data contained here include post-doctoral and professorial researchers but not PhD students, as in Leifeld (2018), without the PhD researchers included in Leifeld and Ingold (2016). For the full dataset, see the replication archive at DOI doi:10.7910/DVN/85SK1M.

Leifeld and Ingold (2016) summarize the data collection strategy as follows: *"Data gathering took place between July and December 2013. A single coder pursued a three-step coding procedure: he first created a list of all relevant university departments and research institutes that host political scientists in Switzerland, then he browsed the websites of these institutes and entered all researchers along with several details about them into a database, including their seniority status (predoctoral, postdoctoral, or professor) and the URL of their publication list (either the CV, the institutional website, a private homepage, or several of those items in order to get a complete publication profile of each person). After entering all researchers of an institute, the coder went through the researchers' publication lists and entered the following pieces of information for each publication into the database: the reporting author, the names of all co-authors, the title of the publication, the year, the name of the journal or book in which the publication appeared (if applicable), the names of all editors (if applicable), and a classification of the type of publication (academic journal, book chapter, monograph, edited volume, other). Most publications are relatively recent, but the earliest publications in the database date back to the 1960s. After completing these three steps, data entered at the beginning was double-checked in order to avoid bias due to new publications that may have shown up during the coding time period. This procedure is the best one can do in terms of completeness, but it should be clear that it crucially depends on the accuracy of the*

self-reported bibliographic information. For example, if a researcher did not update his or her CV or list of publications for the previous six months, those most recent publications only had a chance to enter the database if the co-authors listed the publication on their website. In some relatively rare cases, all authors failed to report recent updates, and this may cause minor inaccuracies in the network dataset, mostly affecting very recent publications in 2013 because there is, on average, a reporting lag."

Based on the collected publication data, a co-authorship network matrix with 156 nodes was created. In addition to this matrix, the dataset here contains node attribute data (institutional affiliations, location, demographics, language shares, publication type shares) and relational covariates (geographical distance, similarity in terms of the share of English articles, and topic similarity) as described in Leifeld (2018). The dataset can be used to replicate Leifeld (2018), but only approximately due to changes in the estimation routine in the **ergm** package since the article was published.

Source

The data were collected from public information online. The full data collection details are described in Leifeld and Ingold (2016).

References

Leifeld, Philip (2018): Polarization in the Social Sciences: Assortative Mixing in Social Science Collaboration Networks is Resilient to Interventions. *Physica A: Statistical Mechanics and its Applications* 507: 510–523. doi:10.1016/j.physa.2018.05.109. Full replication data: doi:10.7910/DVN/85SK1M.

Leifeld, Philip and Karin Ingold (2016): Co-authorship Networks in Swiss Political Research. *Swiss Political Science Review* 22(2): 264–287. doi:10.1111/spsr.12193.

Examples

```
## Not run:
# Replication code for the full Swiss co-authorship ERGM in Leifeld (2018).
# Note that the estimates can only be reproduced approximately due to
# internal changes in the ergm package.

library("network")
library("ergm")

data("ch_coauthor")

# set up network object with node attributes
ch_nw <- network(ch_coaut, directed = FALSE)
set.vertex.attribute(ch_nw, "frequency", ch_nodeattr$num_publications)
set.vertex.attribute(ch_nw, "status", as.character(ch_nodeattr$status))
set.vertex.attribute(ch_nw, "male", ch_nodeattr$male)
set.vertex.attribute(ch_nw, "share_en_articles",
                     ch_nodeattr$share_en_articles)

# create same affiliation matrix
ch_inst_indices <- which(grepl("^inst_.+", colnames(ch_nodeattr)))
ch_same_affiliation <- as.matrix(ch_nodeattr[, ch_inst_indices]) %*%
```

```
  t(ch_nodeattr[, ch_inst_indices])

# create same chair matrix
ch_nodeattr$chairtitle[ch_nodeattr$chairtitle == ""] <- NA
ch_same_chair <- matrix(0, nrow = nrow(ch_same_affiliation),
                        ncol = ncol(ch_same_affiliation))
for (i in 1:length(ch_nodeattr$chairtitle)) {
  for (j in 1:length(ch_nodeattr$chairtitle)) {
    if (i != j &&
        !is.na(ch_nodeattr$chairtitle[i]) &&
        !is.na(ch_nodeattr$chairtitle[j]) &&
        ch_nodeattr$chairtitle[i] == ch_nodeattr$chairtitle[j] &&
        ch_same_affiliation[i, j] == TRUE) {
      ch_same_chair[i, j] <- 1
    }
  }
}
rownames(ch_same_chair) <- rownames(ch_same_affiliation)
colnames(ch_same_chair) <- colnames(ch_same_affiliation)

# create supervision matrix (same chair + affiliation + mixed seniority)
ch_supervision <- ch_same_affiliation *
  ch_same_chair *
  matrix(ch_nodeattr$status == "professor",
         nrow = nrow(ch_same_chair),
         ncol = ncol(ch_same_chair),
         byrow = FALSE) *
  matrix(ch_nodeattr$status != "professor",
         nrow = nrow(ch_same_chair),
         ncol = ncol(ch_same_chair),
         byrow = TRUE)

# ERGM estimation
ch_model <- ergm(ch_nw ~
                   edges +
                   gwesp(0.3, fixed = TRUE) +
                   gwdegree(0.4, fixed = TRUE) +
                   nodecov("frequency") +
                   nodefactor("status") +
                   nodefactor("male") +
                   nodematch("male") +
                   edgecov(ch_dist100km) +
                   edgecov(ch_same_affiliation) +
                   edgecov(ch_same_chair) +
                   edgecov(ch_supervision) +
                   edgecov(ch_topicsim) +
                   nodecov("share_en_articles") +
                   edgecov(ch_en_article_sim),
                 control = control.ergm(MCMLE.termination = "Hummel",
                                        MCMLE.effectiveSize = NULL))
summary(ch_model)  # corresponds Column 1 in Table 3 in Leifeld (2018)

## End(Not run)
```

createBtergm *Constructor for btergm objects*

Description

Constructor for btergm objects.

Usage

```
createBtergm(
  coef,
  boot,
  R,
  nobs,
  time.steps,
  formula,
  formula2,
  response,
  effects,
  weights,
  auto.adjust,
  offset,
  directed,
  bipartite,
  nvertices,
  data
)
```

Arguments

coef	Object of class "numeric". The coefficients.
boot	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).
formula2	The revised formula with the object references after applying the tergmprepare function.
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.
auto.adjust	Object of class "logical". Indicates whether automatic adjustment of dimensions was done before estimation.

offset	Object of class "logical". Indicates whether an offset matrix with structural zeros was used.
directed	Object of class "logical". Are the dependent networks directed?
bipartite	Object of class "logical". Are the dependent networks bipartite?
nvertices	Number of vertices.
data	The data after processing by the tergmprepare function.

Details

Create an S4 btergm object using this constructor function.

Author(s)

Philip Leifeld

See Also

Other tergm-classes: btergm-class, createMtergm(), createTbergm(), mtergm-class, tbergm-class

---

createMtergm *Constructor for mtergm objects*

Description

Constructor for mtergm objects.

Usage

```
createMtergm(
  coef,
  se,
  pval,
  nobs,
  time.steps,
  formula,
  formula2,
  auto.adjust,
  offset,
  directed,
  bipartite,
  estimate,
  loglik,
  aic,
  bic,
  ergm,
  nvertices,
  data
)
```

Arguments

coef	Object of class "numeric". The coefficients.
se	Standard errors.
pval	The p-values.
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).
formula2	The revised formula with the object references after applying the tergmprepare function.
auto.adjust	Object of class "logical". Indicates whether automatic adjustment of dimensions was done before estimation.
offset	Object of class "logical". Indicates whether an offset matrix with structural zeros was used.
directed	Object of class "logical". Are the dependent networks directed?
bipartite	Object of class "logical". Are the dependent networks bipartite?
estimate	Estimate: either MCMC MLE or MPLE.
loglik	Log likelihood of the MLE.
aic	Akaike's Information Criterion.
bic	Bayesian Information Criterion.
ergm	The original ergm object as estimated by the ergm function in the **ergm** package.
nvertices	Number of vertices.
data	The data after processing by the tergmprepare function.

Details

Create an S4 mtergm object using this constructor function.

Author(s)

Philip Leifeld

See Also

Other tergm-classes: btergm-class, createBtergm(), createTbergm(), mtergm-class, tbergm-class

createTbergm *Constructor for tbergm objects*

Description

Constructor for tbergm objects.

Usage

```
createTbergm(
  time.steps,
  formula,
  formula2,
  auto.adjust,
  offset,
  directed,
  bipartite,
  estimate,
  bergm,
  nvertices,
  data
)
```

Arguments

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).
formula2	The revised formula with the object references after applying the tergmprepare function.
auto.adjust	Object of class "logical". Indicates whether automatic adjustment of dimensions was done before estimation.
offset	Object of class "logical". Indicates whether an offset matrix with structural zeros was used.
directed	Object of class "logical". Are the dependent networks directed?
bipartite	Object of class "logical". Are the dependent networks bipartite?
estimate	Estimate: "bergm" for Bayesian estimation.
bergm	The original bergm object as estimated by the bergm function in the **Bergm** package.
nvertices	Number of vertices.
data	The data after processing by the tergmprepare function.

Details

Create an S4 tbergm object using this constructor function.

Author(s)

Philip Leifeld

See Also

Other tergm-classes: `btergm-class`, `createBtergm()`, `createMtergm()`, `mtergm-class`, `tbergm-class`

`edgeprob` *Create all predicted tie probabilities using MPLE*

Description

Create all predicted tie probabilities using MPLE.

Usage

```
edgeprob(object, verbose = FALSE)
```

Arguments

`object`	An `ergm`, `btergm`, or `mtergm` object.
`verbose`	Print details?

Details

For a given (T)ERGM, return a data frame with all predicted edge probabilities along with the design matrix of the MPLE logit model, based on the estimated coefficients and the design matrix, for all time points, along with `i`, `j`, and `t` variables indicating where the respective dyad is located.

`edgeprob` is a convenience function that creates a data frame with all dyads in the ERGM or TERGM along with their edge probabilities and their predictor values (i.e., change statistics). This is useful for creating marginal effects plots or contrasting multiple groups of dyads. This function works faster than the `interpret` function.

Value

The first variable in the resulting data frame contains the edge value (i.e., the dependent variable, which is usually binary). The next variables contain all the predictors from the ERGM or TERGM (i.e., the change statistics). The next five variables contain the indices of the sender (i), the receiver (j), the time step (t), the vertex id of i (i.name), and the vertex id of j (j.name). These five variables serve to identify the dyad. The last variable contains the computed edge probabilities.

See Also

Other interpretation: `interpret()`, `marginalplot()`

`getformula` *Extract the formula from a model*

Description

Extract the model formula from a fitted object.

Usage

```
getformula(x)

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

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

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

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

Arguments

x	A fitted model.

Details

The `getformula` function will extract the formula from a fitted model.

Methods (by class)

- `getformula(ergm)`: Extract the formula from an `ergm` object.
- `getformula(btergm)`: Extract the formula from a `btergm` object.
- `getformula(mtergm)`: Extract the formula from an `mtergm` object.
- `getformula(tbergm)`: Extract the formula from a `tbergm` object.

gof *Goodness-of-fit diagnostics for ERGMs, TERGMs, SAOMs, and logit models*

Description

Assess goodness of fit of btergm and other network models.

Usage

```
gof(object, ...)

createGOF(
  simulations,
  target,
  statistics = c(dsp, esp, deg, ideg, geodesic, rocpr, walktrap.modularity),
  parallel = "no",
  ncpus = 1,
  cl = NULL,
  verbose = TRUE,
  ...
)

## S4 method for signature 'btergm'
gof(
  object,
  target = NULL,
  formula = getformula(object),
  nsim = 100,
  MCMC.interval = 1000,
  MCMC.burnin = 10000,
  parallel = c("no", "multicore", "snow"),
  ncpus = 1,
  cl = NULL,
  statistics = c(dsp, esp, deg, ideg, geodesic, rocpr, walktrap.modularity),
  verbose = TRUE,
  ...
)

## S4 method for signature 'ergm'
gof(
  object,
  target = NULL,
  formula = getformula(object),
  nsim = 100,
  MCMC.interval = 1000,
  MCMC.burnin = 10000,
```

```
  parallel = c("no", "multicore", "snow"),
  ncpus = 1,
  cl = NULL,
  statistics = c(dsp, esp, deg, ideg, geodesic, rocpr, walktrap.modularity),
  verbose = TRUE,
  ...
)

## S4 method for signature 'mtergm'
gof(
  object,
  target = NULL,
  formula = getformula(object),
  nsim = 100,
  MCMC.interval = 1000,
  MCMC.burnin = 10000,
  parallel = c("no", "multicore", "snow"),
  ncpus = 1,
  cl = NULL,
  statistics = c(dsp, esp, deg, ideg, geodesic, rocpr, walktrap.modularity),
  verbose = TRUE,
  ...
)

## S4 method for signature 'tbergm'
gof(
  object,
  target = NULL,
  formula = getformula(object),
  nsim = 100,
  MCMC.interval = 1000,
  MCMC.burnin = 10000,
  parallel = c("no", "multicore", "snow"),
  ncpus = 1,
  cl = NULL,
  statistics = c(dsp, esp, deg, ideg, geodesic, rocpr, walktrap.modularity),
  verbose = TRUE,
  ...
)

## S4 method for signature 'sienaFit'
gof(
  object,
  period = NULL,
  parallel = c("no", "multicore", "snow"),
  ncpus = 1,
  cl = NULL,
  structzero = 10,
```

```
  statistics = c(esp, deg, ideg, geodesic, rocpr, walktrap.modularity),
  groupName = object$f$groupNames[[1]],
  varName = NULL,
  outofsample = FALSE,
  sienaData = NULL,
  sienaEffects = NULL,
  nsim = NULL,
  verbose = TRUE,
  ...
)

## S4 method for signature 'network'
gof(
  object,
  covariates,
  coef,
  target = NULL,
  nsim = 100,
  mcmc = FALSE,
  MCMC.interval = 1000,
  MCMC.burnin = 10000,
  parallel = c("no", "multicore", "snow"),
  ncpus = 1,
  cl = NULL,
  statistics = c(dsp, esp, deg, ideg, geodesic, rocpr, walktrap.modularity),
  verbose = TRUE,
  ...
)

## S4 method for signature 'matrix'
gof(
  object,
  covariates,
  coef,
  target = NULL,
  nsim = 100,
  mcmc = FALSE,
  MCMC.interval = 1000,
  MCMC.burnin = 10000,
  parallel = c("no", "multicore", "snow"),
  ncpus = 1,
  cl = NULL,
  statistics = c(dsp, esp, deg, ideg, geodesic, rocpr, walktrap.modularity),
  verbose = TRUE,
  ...
)
```

Arguments

object	A btergm, ergm, or sienaFit object (for the btergm, ergm, and sienaFit methods, respectively). Or a network object or matrix (for the network and matrix methods, respectively).
...	Arbitrary further arguments to be passed on to the statistics. See also the help page for the gof-statistics.
simulations	A list of network objects or sparse matrices (generated using the **Matrix** package) representing simulated networks.
target	In the gof function: 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. In the createGOF function: a list of sparse matrices (generated using the **Matrix** package) or a list of network objects (generated using the **network** package). The simulations are compared against these target networks.
statistics	A list of functions used for comparison of observed and simulated networks. Note that the list should contain the actual functions, not a character representation of them. See gof-statistics for details.
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, 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). Note that "multicore" 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. Parallel computing is described in more detail on the help page of btergm.
ncpus	The number of CPU cores used for parallel GOF assessment (only if parallel is activated). If the number of cores should be detected automatically on the machine where the code is executed, one can try the detectCores() function from the **parallel** package. 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.
verbose	Print details?
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. The comparison between simulated and observed networks is only done within time steps. For example, the first 100 simulations are compared with the first observed network, simulations 101-200 with the second observed network etc.

`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 `1000`, which means that every 1000th 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.

`period` Which transition between time periods should be used for GOF assessment? By default, all transitions between all time periods are used. For example, if there are three consecutive networks, this will extract simulations from the transitions between 1 and 2 and between 2 and 3, respectively, and these simulations will be compared to the networks at time steps 2 and 3, respectively. The time period can be provided as a numeric, e.g., `period = 4` for extracting the simulations between time steps 4 and 5 (= the fourth transition) and predicting the fifth network. Values lower than 1 or larger than the number of consecutive networks minus 1 are therefore not permitted. This argument is only used if out-of-sample prediction is switched off.

`structzero` Which value was used for structural zeros (usually nodes that have dropped out of the network or have not yet joined the network) in the dependent variable/network? These nodes are removed from the observed network and the simulations before comparison. Usually, the value `10` is used for structural zeros in Siena.

`groupName` The group name used in the Siena model.

`varName` The variable name that denotes the dependent networks in the Siena model.

`outofsample` Should out-of-sample prediction be attempted? If so, some additional arguments must be provided: `sienaData`, `sienaEffects`, and `nsim`. The `sienaData` object must contain a base and a target network for out-of-sample prediction. The `sienaEffects` must contain the effects to be used for the simulations. The estimates will be taken from the estimated `object`, and they will be injected into a new SAOM and fixed during the sampling procedure. `nsim` determines how many simulations are used for the out-of-sample comparison.

`sienaData` An object of the class `siena`, which is usually created using the `sienaDataCreate` function in the **RSiena** package. This argument is only used for out-of-sample prediction. The object must be based on a `sienaDependent` object that contains two networks: the base network from which to simulate forward, and the target network which you want to predict out-of-sample. The object can contain further objects for storing covariates etc. that are necessary for estimating new networks. The best practice is to create an object that is identical to the `siena` object used for estimating the model, except that it contains the base and the target network instead of the dependent variable/networks.

`sienaEffects` An object of the class `sienaEffects`, which is usually created using the `getEffects()` and the `includeEffects()` functions in the `RSiena` package. The best practice is to provide a `sienaEffects` object that is identical to the object used to create the original model (that is, it should contain the same effects), except that it should be based on the `siena` object provided through the `sienaData` argument. In other words, the `sienaEffects` object should be based on the base and target network used for out-of-sample prediction, and it should contain the same effects as those used for the original estimation. This argument is used only for out-of-sample prediction.

`covariates` A list of matrices or network objects that serve as covariates for the dependent network. The covariates in this list are automatically added to the formula as `edgecov` terms.

`coef` A vector of coefficients.

`mcmc` Should statnet's MCMC methods be used for simulating new networks? If `mcmc = FALSE`, new networks are simulated based on predicted tie probabilities of the regression equation.

Details

The generic `gof` function provides goodness-of-fit measures and degeneracy checks for `btergm`, `mtergm`, `tbergm`, `ergm`, `sienaFit`, and custom dyadic-independent models. The user can provide a list of network statistics for comparing simulated networks based on the estimated model with the observed network(s). See `gof-statistics`. The objects created by these methods can be displayed using various plot and print methods (see `gof-plot`).

In-sample GOF assessment is the default, which means that the same time steps are used for creating simulations and for comparison with the observed network(s). It is possible to do out-of-sample prediction by specifying a (list of) target network(s) using the `target` argument. 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 `sienaFit` method of the `gof` function. By default, however, within-sample fit is used for SAOMs, just like for (T)ERGMs.

The gof methods for networks and matrices serve to assess the goodness of fit of a dyadic-independence model. To do this, the method requires a vector of coefficients (one coefficient for the intercept or edges term and one coefficient for each covariate), a list of covariates (in matrix or network shape), and a dependent network or matrix. This is useful for assessing the goodness of fit of QAP-adjusted logistic regression models (as implemented in the netlogit function in the **sna** package) or other dyadic-independence models, such as models fitted using glm. Note that this method only works with cross-sectional models and does not accept lists of networks as input data.

The createGOF function is used internally by the gof function in order to create a gof object from a list of simulated networks and a list of target networks to compare against. It can also be used directly by the end user if the user wants to supply lists of simulated and target networks from other sources.

References

Leifeld, Philip, Skyler J. Cranmer and Bruce A. Desmarais (2018): Temporal Exponential Random Graph Models with btergm: Estimation and Bootstrap Confidence Intervals. *Journal of Statistical Software* 83(6): 1–36. doi:10.18637/jss.v083.i06.

Leifeld, Philip and Skyler J. Cranmer (2019): A Theoretical and Empirical Comparison of the Temporal Exponential Random Graph Model and the Stochastic Actor-Oriented Model. Network Science 7(1): 20–51. doi:10.1017/nws.2018.26.

---

gof-plot *Plot and print methods for GOF output*

Description

Plot and print methods for goodness-of-fit output for network models.

Usage

```
## S3 method for class 'boxplot'
print(x, ...)

## S3 method for class 'roc'
print(x, ...)

## S3 method for class 'pr'
print(x, ...)

## S3 method for class 'rocpr'
print(x, ...)

## S3 method for class 'univariate'
print(x, ...)

## S3 method for class 'gof'
```

```
print(x, ...)

## S3 method for class 'gof'
plot(x, mfrow = TRUE, ...)

## S3 method for class 'boxplot'
plot(
  x,
  relative = TRUE,
  transform = function(x) x,
  xlim = NULL,
  main = x$label,
  xlab = x$label,
  ylab = "Frequency",
  border = "darkgray",
  boxplot.lwd = 0.8,
  outline = FALSE,
  median = TRUE,
  median.col = "black",
  median.lty = "solid",
  median.lwd = 2,
  mean = TRUE,
  mean.col = "black",
  mean.lty = "dashed",
  mean.lwd = 1,
  ...
)

## S3 method for class 'roc'
plot(
  x,
  add = FALSE,
  main = x$label,
  avg = c("none", "horizontal", "vertical", "threshold"),
  spread.estimate = c("boxplot", "stderror", "stddev"),
  lwd = 3,
  rgraph = FALSE,
  col = "#bd0017",
  random.col = "#bd001744",
  ...
)

## S3 method for class 'pr'
plot(
  x,
  add = FALSE,
  main = x$label,
  avg = c("none", "horizontal", "vertical", "threshold"),
```

```
  spread.estimate = c("boxplot", "stderror", "stddev"),
  lwd = 3,
  rgraph = FALSE,
  col = "#5886be",
  random.col = "#5886be44",
  pr.poly = 0,
  ...
)

## S3 method for class 'rocpr'
plot(
  x,
  main = x$label,
  roc.avg = c("none", "horizontal", "vertical", "threshold"),
  roc.spread.estimate = c("boxplot", "stderror", "stddev"),
  roc.lwd = 3,
  roc.rgraph = FALSE,
  roc.col = "#bd0017",
  roc.random.col = "#bd001744",
  pr.avg = c("none", "horizontal", "vertical", "threshold"),
  pr.spread.estimate = c("boxplot", "stderror", "stddev"),
  pr.lwd = 3,
  pr.rgraph = FALSE,
  pr.col = "#5886be",
  pr.random.col = "#5886be44",
  pr.poly = 0,
  ...
)

## S3 method for class 'univariate'
plot(
  x,
  main = x$label,
  sim.hist = TRUE,
  sim.bar = TRUE,
  sim.density = TRUE,
  obs.hist = FALSE,
  obs.bar = TRUE,
  obs.density = TRUE,
  sim.adjust = 1,
  obs.adjust = 1,
  sim.lwd = 2,
  obs.lwd = 2,
  sim.col = "black",
  obs.col = "red",
  ...
)
```

Arguments

x	An object created by one of the gof methods.
...	Arbitrary further arguments.
mfrow	Should the GOF plots come out separately (mfrow = FALSE), or should all statistics be aligned in a single diagram (mfrow = TRUE)? Returning the plots separately can be helpful if the output is redirected to a multipage PDF or TIFF file.
relative	Print relative frequencies (as opposed to absolute frequencies) of a statistic on the y axis?
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, 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 transform = function(x) log1p to avoid infinite values.
xlim	Horizontal limit of the boxplots. Only the maximum value must be provided, e.g., xlim = 8.
main	Main title of a GOF plot.
xlab	Label of the x-axis of a GOF plot.
ylab	Label of the y-axis of a GOF plot.
border	Color of the borders of the boxplots.
boxplot.lwd	Line width of boxplot.
outline	Print outliers in the boxplots?
median	Plot the median curve for the observed network?
median.col	Color of the median of the observed network statistic.
median.lty	Line type of median line. For example "dashed" or "solid".
median.lwd	Line width of median line.
mean	Plot the mean curve for the observed network?
mean.col	Color of the mean of the observed network statistic.
mean.lty	Line type of mean line. For example "dashed" or "solid".
mean.lwd	Line width of mean line.
add	Add the ROC and/or PR curve to an existing plot?
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.

`spread.estimate`
: 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.

`lwd`
: Line width.

`rgraph`
: Should an ROC or PR curve also be drawn for a random graph? This serves as a baseline against which to compare the actual ROC or PR curve.

`col`
: Color of the ROC or PR curve.

`random.col`
: Color of the ROC or PR curve of the random graph prediction.

`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).

`roc.avg`
: Averaging pattern for the ROC 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.

`roc.spread.estimate`
: 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.

`roc.lwd`
: Line width.

`roc.rgraph`
: Should an ROC curve also be drawn for a random graph? This serves as a baseline against which to compare the actual ROC curve.

`roc.col`
: Color of the ROC curve.

`roc.random.col`
: Color of the ROC curve of the random graph prediction.

`pr.avg`
: Averaging pattern for the 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.

`pr.spread.estimate`
: 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.

`pr.lwd`
: Line width.

`pr.rgraph`
: Should an PR curve also be drawn for a random graph? This serves as a baseline against which to compare the actual PR curve.

`pr.col`
: Color of the PR curve.

`pr.random.col`
: Color of the PR curve of the random graph prediction.

`sim.hist`
: Draw a histogram for the simulated networks?

`sim.bar`
: Draw a bar for the median of the statistic for the simulated networks?

`sim.density`
: Draw a density curve fot the statistic for the simulated networks?

`obs.hist`
: Draw a histogram for the observed networks?

`obs.bar`
: Draw a bar for the median of the statistic for the observed networks?

`obs.density`
: Draw a density curve fot the statistic for the observed networks?

`sim.adjust`
: Bandwidth adjustment parameter for the density curve.

`obs.adjust`
: Bandwidth adjustment parameter for the density curve.

`sim.lwd`
: Line width for the simulated networks.

`obs.lwd`
: Line width for the observed network(s).

`sim.col`
: Color for the simulated networks.

`obs.col`
: Color for the observed network(s).

Details

These plot and print methods serve to display the output generated by the `gof` function and its methods. See the help page of `gof-methods` for details on how to compute goodness-of-fit statistics.

References

Leifeld, Philip, Skyler J. Cranmer and Bruce A. Desmarais (2018): Temporal Exponential Random Graph Models with btergm: Estimation and Bootstrap Confidence Intervals. *Journal of Statistical Software* 83(6): 1–36. doi:10.18637/jss.v083.i06.

gof-statistics *Statistics for goodness-of-fit assessment of network models*

Description

Statistics for goodness-of-fit assessment of network models.

Usage

```
dsp(mat, ...)

esp(mat, ...)

nsp(mat, ...)

deg(mat, ...)

b1deg(mat, ...)

b2deg(mat, ...)

odeg(mat, ...)

ideg(mat, ...)

kstar(mat, ...)

b1star(mat, ...)

b2star(mat, ...)

ostar(mat, ...)

istar(mat, ...)

kcycle(mat, ...)

geodesic(mat, ...)

triad.directed(mat, ...)

triad.undirected(mat, ...)

comemb(vec)

walktrap.modularity(mat, ...)
```

```
walktrap.roc(sim, obs, ...)

walktrap.pr(sim, obs, ...)

fastgreedy.modularity(mat, ...)

fastgreedy.roc(sim, obs, ...)

fastgreedy.pr(sim, obs, ...)

louvain.modularity(mat, ...)

louvain.roc(sim, obs, ...)

louvain.pr(sim, obs, ...)

maxmod.modularity(mat, ...)

maxmod.roc(sim, obs, ...)

maxmod.pr(sim, obs, ...)

edgebetweenness.modularity(mat, ...)

edgebetweenness.roc(sim, obs, ...)

edgebetweenness.pr(sim, obs, ...)

spinglass.modularity(mat, ...)

spinglass.roc(sim, obs, ...)

spinglass.pr(sim, obs, ...)

rocpr(sim, obs, roc = TRUE, pr = TRUE, joint = TRUE, pr.impute = "poly4", ...)
```

Arguments

mat	A sparse network matrix as created by the `Matrix` function in the **Matrix** package.
...	Additional arguments. This must be present in all auxiliary GOF statistics.
vec	A vector of community memberships in order to create a community co-membership matrix.
sim	A list of simulated networks. Each element in the list should be a sparse matrix as created by the `Matrix` function in the **Matrix** package.
obs	A list of observed (= target) networks. Each element in the list should be a sparse matrix as created by the `Matrix` function in the **Matrix** package.

`roc`	Compute receiver-operating characteristics (ROC)?
`pr`	Compute precision-recall curve (PR)?
`joint`	Merge all time steps into a single big prediction task and compute predictive fit (instead of computing GOF for all time steps separately)?
`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 object and using the `pr.poly` argument to plot the predicted curve on top of the actual PR curve.

Details

These functions can be plugged into the `statistics` argument of the `gof` methods in order to compare observed with simulated networks (see the gof-methods help page). There are three types of statistics:

1. Univariate statistics, which aggregate a network into a single quantity. For example, modularity measures or density. The distribution of statistics can be displayed using histograms, density plots, and median bars. Univariate statistics take a sparse matrix (`mat`) as an argument and return a single numeric value that summarize a network matrix.
2. Multivariate statistics, which aggregate a network into a vector of quantities. For example, the distribution of geodesic distances, edgewise shared partners, or indegree. These statistics typically have multiple values, e.g., esp(1), esp(2), esp(3) etc. The results can be displayed using multiple boxplots for simulated networks and a black curve for the observed network(s). Multivariate statistics take a sparse matrix (`mat`) as an argument and return a vector of numeric values that summarize a network matrix.
3. Tie prediction statistics, which predict dyad states the observed network(s) by the dyad states in the simulated networks. For example, receiver operating characteristics (ROC) or precision-recall curves (PR) of simulated networks based on the model, or ROC or PR predictions of community co-membership matrices of the simulated vs. the observed network(s). Tie prediction statistics take a list of simulated sparse network matrices and another list of observed sparse network matrices (possibly containing only a single sparse matrix) as arguments and return a `rocpr`, `roc`, or `pr` object (as created by the rocpr function).

Users can create their own statistics for use with the `gof` methods. To do so, one needs to write a function that accepts and returns the respective objects described in the enumeration above. It is advisable to look at the definitions of some of the existing functions to add custom functions. It is also possible to add an attribute called `label` to the return object, which describes what is being returned by the function. This label will be used as a descriptive label in the plot and for verbose output during computations. The examples section contains an example of a custom user statistic. Note that all statistics *must* contain the `...` argument to ensure that custom arguments of other statistics do not cause an error.

To aid the development of custom statistics, the helper function `comemb` is available: it accepts a vector of community memberships and converts it to a co-membership matrix. This function is also used internally by statistics like `walktrap.roc` and others.

Functions

- `dsp()`: Multivariate GOF statistic: dyad-wise shared partner distribution
- `esp()`: Multivariate GOF statistic: edge-wise shared partner distribution
- `nsp()`: Multivariate GOF statistic: non-edge-wise shared partner distribution
- `deg()`: Multivariate GOF statistic: degree distribution
- `b1deg()`: Multivariate GOF statistic: degree distribution for the first mode
- `b2deg()`: Multivariate GOF statistic: degree distribution for the second mode
- `odeg()`: Multivariate GOF statistic: outdegree distribution
- `ideg()`: Multivariate GOF statistic: indegree distribution
- `kstar()`: Multivariate GOF statistic: k-star distribution
- `b1star()`: Multivariate GOF statistic: k-star distribution for the first mode
- `b2star()`: Multivariate GOF statistic: k-star distribution for the second mode
- `ostar()`: Multivariate GOF statistic: outgoing k-star distribution
- `istar()`: Multivariate GOF statistic: incoming k-star distribution
- `kcycle()`: Multivariate GOF statistic: k-cycle distribution
- `geodesic()`: Multivariate GOF statistic: geodesic distance distribution
- `triad.directed()`: Multivariate GOF statistic: triad census in directed networks
- `triad.undirected()`: Multivariate GOF statistic: triad census in undirected networks
- `comemb()`: Helper function: create community co-membership matrix
- `walktrap.modularity()`: Univariate GOF statistic: Walktrap modularity distribution
- `walktrap.roc()`: Tie prediction GOF statistic: ROC of Walktrap community detection. Receiver-operating characteristics of predicting the community structure in the observed network(s) by the community structure in the simulated networks, as computed by the Walktrap algorithm.
- `walktrap.pr()`: Tie prediction GOF statistic: PR of Walktrap community detection. Precision-recall curve for predicting the community structure in the observed network(s) by the community structure in the simulated networks, as computed by the Walktrap algorithm.
- `fastgreedy.modularity()`: Univariate GOF statistic: fast and greedy modularity distribution
- `fastgreedy.roc()`: Tie prediction GOF statistic: ROC of fast and greedy community detection. Receiver-operating characteristics of predicting the community structure in the observed network(s) by the community structure in the simulated networks, as computed by the fast and greedy algorithm. Only sensible with undirected networks.
- `fastgreedy.pr()`: Tie prediction GOF statistic: PR of fast and greedy community detection. Precision-recall curve for predicting the community structure in the observed network(s) by the community structure in the simulated networks, as computed by the fast and greedy algorithm. Only sensible with undirected networks.

- `louvain.modularity()`: Univariate GOF statistic: Louvain clustering modularity distribution
- `louvain.roc()`: Tie prediction GOF statistic: ROC of Louvain community detection. Receiver-operating characteristics of predicting the community structure in the observed network(s) by the community structure in the simulated networks, as computed by the Louvain algorithm.
- `louvain.pr()`: Tie prediction GOF statistic: PR of Louvain community detection. Precision-recall curve for predicting the community structure in the observed network(s) by the community structure in the simulated networks, as computed by the Louvain algorithm.
- `maxmod.modularity()`: Univariate GOF statistic: maximal modularity distribution
- `maxmod.roc()`: Tie prediction GOF statistic: ROC of maximal modularity community detection. Receiver-operating characteristics of predicting the community structure in the observed network(s) by the community structure in the simulated networks, as computed by the modularity maximization algorithm.
- `maxmod.pr()`: Tie prediction GOF statistic: PR of maximal modularity community detection. Precision-recall curve for predicting the community structure in the observed network(s) by the community structure in the simulated networks, as computed by the modularity maximization algorithm.
- `edgebetweenness.modularity()`: Univariate GOF statistic: edge betweenness modularity distribution
- `edgebetweenness.roc()`: Tie prediction GOF statistic: ROC of edge betweenness community detection. Receiver-operating characteristics of predicting the community structure in the observed network(s) by the community structure in the simulated networks, as computed by the Girvan-Newman edge betweenness community detection method.
- `edgebetweenness.pr()`: Tie prediction GOF statistic: PR of edge betweenness community detection. Precision-recall curve for predicting the community structure in the observed network(s) by the community structure in the simulated networks, as computed by the Girvan-Newman edge betweenness community detection method.
- `spinglass.modularity()`: Univariate GOF statistic: spinglass modularity distribution
- `spinglass.roc()`: Tie prediction GOF statistic: ROC of spinglass community detection. Receiver-operating characteristics of predicting the community structure in the observed network(s) by the community structure in the simulated networks, as computed by the Spinglass algorithm.
- `spinglass.pr()`: Tie prediction GOF statistic: PR of spinglass community detection. Precision-recall curve for predicting the community structure in the observed network(s) by the community structure in the simulated networks, as computed by the Spinglass algorithm.
- `rocpr()`: Tie prediction GOF statistic: ROC and PR curves. Receiver-operating characteristics (ROC) and precision-recall curve (PR). Prediction of the dyad states of the observed network(s) by the dyad states of the simulated networks.

References

Leifeld, Philip, Skyler J. Cranmer and Bruce A. Desmarais (2018): Temporal Exponential Random Graph Models with btergm: Estimation and Bootstrap Confidence Intervals. *Journal of Statistical Software* 83(6): 1–36. doi:10.18637/jss.v083.i06.

Examples

```
# To see how these statistics are used, look at the examples section of
# ?"gof-methods". The following example illustrates how custom
# statistics can be created. Suppose one is interested in the density
# of a network. Then a univariate statistic can be created as follows.

dens <- function(mat, ...) {        # univariate: one argument
  mat <- as.matrix(mat)             # sparse matrix -> normal matrix
  d <- sna::gden(mat)               # compute the actual statistic
  attributes(d)$label <- "Density"  # add a descriptive label
  return(d)                         # return the statistic
}

# Note that the '...' argument must be present in all statistics.
# Now the statistic can be used in the statistics argument of one of
# the gof methods.

# For illustrative purposes, let us consider an existing statistic, the
# indegree distribution, a multivariate statistic. It also accepts a
# single argument. Note that the sparse matrix is converted to a
# normal matrix object when it is used. First, statnet's summary
# method is used to compute the statistic. Names are attached to the
# resulting vector for the different indegree values. Then the vector
# is returned.

ideg <- function(mat, ...) {
  d <- summary(mat ~ idegree(0:(nrow(mat) - 1)))
  names(d) <- 0:(length(d) - 1)
  attributes(d)$label <- "Indegree"
  return(d)
}

# See the gofstatistics.R file in the package for more complex examples.
```

---

handleMissings *Handle missing data in matrices*

Description

Process NA values (= remove nodes with NAs iteratively).

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.

Value

Either a matrix in which missing data were taken care of or a matrix indicating where missing data are located.

See Also

adjust

---

interpret *Micro-Level Interpretation of (T)ERGMs*

Description

Micro-level interpretation of (T)ERGMs.

Usage

```
interpret(object, ...)

## S4 method for signature 'ergm'
interpret(
  object,
  formula = getformula(object),
  coefficients = coef(object),
  target = NULL,
  type = "tie",
  i,
```

```
  j
)

## S4 method for signature 'btergm'
interpret(
  object,
  formula = getformula(object),
  coefficients = coef(object),
  target = NULL,
  type = "tie",
  i,
  j,
  t = 1:object@time.steps
)

## S4 method for signature 'mtergm'
interpret(
  object,
  formula = getformula(object),
  coefficients = coef(object),
  target = NULL,
  type = "tie",
  i,
  j,
  t = 1:object@time.steps
)
```

Arguments

object	An ergm, btergm, or mtergm object.
...	Further arguments to be passed on to subroutines.
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).
target	The response network on which probabilities are based. Depending on whether the function is applied to an ergm or btergm/mtergm object, this can be either a single network or a list of networks. By default, the (list of) network(s) provided as the left-hand side of the (T)ERGM formula is used.
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 j = 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 column 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 j = 7?

t — A vector of (numerical) time steps for which the probabilities should be computed. This only applies to btergm amd mtergm 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.

Details

The interpret function facilitates interpretation of ERGMs and TERGMs at the micro level, as described in Desmarais and Cranmer (2012). There are methods for ergm objects, btergm objects, and mtergm 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 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.

The typical procedure is to manually enumerate all dyads or sender-receiver-time combinations with certain properties and repeat the same thing with some alternative properties for contrasting the two groups. Then apply the `interpret` function to the two groups of dyads and compute a measure of central tendency (e.g., mean or median) and possibly some uncertainy measure (i.e., confidence intervals) from the distribution of dyadic probabilities in each group. For example, if there is a gender attribute, one can sample male-male or female-female dyads, compute the distributions of edge probabilities for the two sets of dyads, and create boxplots or barplots with confidence intervals for the two types of dyads in order to contrast edge probabilities for male versus female same-sex dyads.

See also the `edgeprob` function for automatic computation of all dyadic edge probabilities.

Methods (by class)

- `interpret(ergm)`: Interpret method for `ergm` objects
- `interpret(btergm)`: Interpret method for `btergm` objects
- `interpret(mtergm)`: Interpret method for `mtergm` objects

References

Desmarais, Bruce A. and Skyler J. Cranmer (2012): Micro-Level Interpretation of Exponential Random Graph Models with Application to Estuary Networks. *Policy Studies Journal* 40(3): 402–434. doi:10.1111/j.15410072.2012.00459.x.

Leifeld, Philip, Skyler J. Cranmer and Bruce A. Desmarais (2017): Temporal Exponential Random Graph Models with btergm: Estimation and Bootstrap Confidence Intervals. *Journal of Statistical Software* 83(6): 1–36. doi:10.18637/jss.v083.i06.

Czarna, Anna Z., Philip Leifeld, Magdalena Smieja, Michael Dufner and Peter Salovey (2016): Do Narcissism and Emotional Intelligence Win Us Friends? Modeling Dynamics of Peer Popularity Using Inferential Network Analysis. *Personality and Social Psychology Bulletin* 42(11): 1588–1599. doi:10.1177/0146167216666265.

See Also

Other interpretation: `edgeprob()`, `marginalplot()`

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)


##### The following examples illustrate the behavior of  #####
##### the interpret function with undirected and/or      #####
##### bipartite graphs with or without structural zeros. #####

library("statnet")
library("btergm")

# micro-level interpretation for undirected network with structural zeros
set.seed(12345)
mat <- matrix(rbinom(400, 1, 0.1), nrow = 20, ncol = 20)
mat[1, 5] <- 1
mat[10, 7] <- 1
mat[15, 3] <- 1
mat[18, 4] < 1
nw <- network(mat, directed = FALSE, bipartite = FALSE)
cv <- matrix(rnorm(400), nrow = 20, ncol = 20)
offsetmat <- matrix(rbinom(400, 1, 0.1), nrow = 20, ncol = 20)
offsetmat[1, 5] <- 1
offsetmat[10, 7] <- 1
offsetmat[15, 3] <- 1
offsetmat[18, 4] < 1
model <- ergm(nw ~ edges + kstar(2) + edgecov(cv) + offset(edgecov(offsetmat)),
              offset.coef = -Inf)
summary(model)

# tie-level interpretation (note that dyad interpretation would not make any
# sense in an undirected network):
interpret(model, type = "tie", i = 1, j = 2)  # 0.28 (= normal dyad)
interpret(model, type = "tie", i = 1, j = 5)  # 0.00 (= structural zero)

# node-level interpretation; note the many 0 probabilities due to the
# structural zeros; also note the warning message that the probabilities may
# be slightly imprecise because -Inf needs to be approximated by some large
# negative number (-9e8):
interpret(model, type = "node", i = 1, j = 3:5)
```

```
# repeat the same exercise for a directed network
nw <- network(mat, directed = TRUE, bipartite = FALSE)
model <- ergm(nw ~ edges + istar(2) + edgecov(cv) + offset(edgecov(offsetmat)),
              offset.coef = -Inf)
interpret(model, type = "tie", i = 1, j = 2)  # 0.13 (= normal dyad)
interpret(model, type = "tie", i = 1, j = 5)  # 0.00 (= structural zero)
interpret(model, type = "dyad", i = 1, j = 2)  # results for normal dyad
interpret(model, type = "dyad", i = 1, j = 5)  # results for i->j struct. zero
interpret(model, type = "node", i = 1, j = 3:5)

# micro-level interpretation for bipartite graph with structural zeros
set.seed(12345)
mat <- matrix(rbinom(200, 1, 0.1), nrow = 20, ncol = 10)
mat[1, 5] <- 1
mat[10, 7] <- 1
mat[15, 3] <- 1
mat[18, 4] < 1
nw <- network(mat, directed = FALSE, bipartite = TRUE)
cv <- matrix(rnorm(200), nrow = 20, ncol = 10)  # some covariate
offsetmat <- matrix(rbinom(200, 1, 0.1), nrow = 20, ncol = 10)
offsetmat[1, 5] <- 1
offsetmat[10, 7] <- 1
offsetmat[15, 3] <- 1
offsetmat[18, 4] < 1
model <- ergm(nw ~ edges + b1star(2) + edgecov(cv)
              + offset(edgecov(offsetmat)), offset.coef = -Inf)
summary(model)

# tie-level interpretation; note the index for the second mode starts with 21
interpret(model, type = "tie", i = 1, j = 21)

# dyad-level interpretation does not make sense because network is undirected;
# node-level interpretation prints warning due to structural zeros, but
# computes the correct probabilities (though slightly imprecise because -Inf
# is approximated by some small number:
interpret(model, type = "node", i = 1, j = 21:25)

# compute all dyadic probabilities
dyads <- edgeprob(model)
dyads

## End(Not run)
```

---

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

Description

Longitudinal classroom friendship network and behavior (Andrea Knecht).

Format

Note: the data have to be transformed before they can be used with **btergm** and related packages (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)

Details

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.

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: doi:10.17026/dansz9bh2bp
- 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 this 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. https://dspace.library.uu.nl/handle/1874/25950.

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. doi:10.1111/j.14679507.2009.00564.x.

Leifeld, Philip and Skyler J. Cranmer (2019): A Theoretical and Empirical Comparison of the Temporal Exponential Random Graph Model and the Stochastic Actor-Oriented Model. Network Science 7(1): 20–51. doi:10.1017/nws.2018.26.

Leifeld, Philip, Skyler J. Cranmer and Bruce A. Desmarais (2018): Temporal Exponential Random Graph Models with btergm: Estimation and Bootstrap Confidence Intervals. *Journal of Statistical Software* 83(6): 1–36. doi:10.18637/jss.v083.i06.

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. doi: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:
# ========================================================================
# The following example was taken from the JSS article about btergm
# that is referenced above (Leifeld, Cranmer and Desmarais 2018).
# ========================================================================

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

# 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]])
sex <- demographics$sex
names(sex) <- 1:length(sex)

# step 2: imputation of NAs and removal of absent nodes:
friendship <- handleMissings(friendship, na = 10, method = "remove")
friendship <- handleMissings(friendship, na = NA, method = "fillmode")

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

# step 4: plot the networks
pdf("knecht.pdf")
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 = TRUE, edge.col = "grey50")
}
dev.off()

# step 5: estimate TERGMS without and with temporal dependencies
model.2a <- btergm(friendship ~ edges + mutual + ttriple +
```

```
    transitiveties + ctriple + nodeicov("idegsqrt") +
    nodeicov("odegsqrt") + nodeocov("odegsqrt") +
    nodeofactor("sex") + nodeifactor("sex") + nodematch("sex") +
    edgecov(primary), R = 100)

model.2b <- btergm(friendship ~ edges + mutual + ttriple +
    transitiveties + ctriple + nodeicov("idegsqrt") +
    nodeicov("odegsqrt") + nodeocov("odegsqrt") +
    nodeofactor("sex") + nodeifactor("sex") + nodematch("sex") +
    edgecov(primary) + delrecip + memory(type = "stability"),
    R = 100)

# step 6: alternatively, estimate via MCMC-MLE:
model.2d <- mtergm(friendship ~ edges + mutual + ttriple +
    transitiveties + ctriple + nodeicov("idegsqrt") +
    nodeicov("odegsqrt") + nodeocov("odegsqrt") +
    nodeofactor("sex") + nodeifactor("sex") + nodematch("sex") +
    edgecov(primary) + delrecip + memory(type = "stability"),
    control = control.ergm(MCMC.samplesize = 5000, MCMC.interval = 2000))

# step 7: GOF assessment with out-of-sample prediction
# (note the commentaries and corrections at
#  https://doi.org/10.1017/nws.2022.7 and
#  https://doi.org/10.1017/nws.2022.6)
model.2e <- btergm(friendship[1:3] ~ edges + mutual + ttriple +
    transitiveties + ctriple + nodeicov("idegsqrt") +
    nodeicov("odegsqrt") + nodeocov("odegsqrt") +
    nodeofactor("sex") + nodeifactor("sex") + nodematch("sex") +
    edgecov(primary) + delrecip + memory(type = "stability"),
    R = 100)

gof.2e <- gof(model.2e, nsim = 100, target = friendship[[4]],
    formula = friendship[3:4] ~ edges + mutual + ttriple +
    transitiveties + ctriple + nodeicov("idegsqrt") +
    nodeicov("odegsqrt") + nodeocov("odegsqrt") +
    nodeofactor("sex") + nodeifactor("sex") + nodematch("sex") +
    edgecov(primary) + delrecip + memory(type = "stability"),
    coef = coef(model.2b), statistics = c(esp, dsp, geodesic,
    deg, triad.undirected, rocpr))
pdf("gof-2e.pdf", width = 8, height = 6)
plot(gof.2e)
dev.off()

## End(Not run)
```

`marginalplot` *Plot marginal effects for two-way interactions in (T)ERGMs*

Description

Plot marginal effects for two-way interactions in (T)ERGMs.

Usage

```
marginalplot(
  model,
  var1,
  var2,
  inter,
  ci = 0.95,
  rug = FALSE,
  point = FALSE,
  structzeromat = NULL,
  zeroline = TRUE,
  color = "black",
  xlab = NULL,
  ylab = NULL
)
```

Arguments

model	An ergm object as generated by the **ergm** package. Note that marginal effects plots cannot be created for btergm objects because the variance-covariance matrix is not valid. However, it should be possible to apply the marginalplot function to MCMC-MLE-estimated TERGMs because the ergm object is stored in the ergm slot of an mtergm object. To do this, supply the ergm object instead of the mtergm object (e.g., marginalplot(mtergmobject@ergm)).
var1	Name of the first main variable. This is the focal variable.
var2	Name of the second main variable. This is the conditioning variable.
inter	Name of the interaction effect.
ci	Significance level.
rug	Display the distribution of the conditioning variable at the bottom of the plot?
point	Display error bars for the levels of the conditioning variable (instead of a continuous curve)?
structzeromat	An optional matrix object which indicates dyads that should be deleted prior to the calculation of the confidence interval for the marginal effect curve. This is useful when such a matrix was used to indicate structural zeros during estimation. In this event, the dyads characterized by structural zeros are not allowed to be tied, therefore they should be removed from the set of dyads used for the calculation of marginal effects. The matrix should contain ones for structural zeros and zeros for entries that should be used.
zeroline	Draw a horizontal line to indicate zero for the first main variable?
color	Color of the curve, confidence interval, and distribution.
xlab	Axis label for the second (conditioning) variable.
ylab	Axis label for the first (focal) variable.

Details

The `marginalplot` function creates marginal effects plots for ERGMs with interaction effects. The user has to supply the `ergm` object and the coefficient names of the first main variable, the second main variable, and the interaction term as stored in the coefficients vector inside the `ergm` object. It is possible to draw continuous curves or discrete error bars depending on the nature of the data (using the `point` argument). The distribution of the second (conditioning) variable can be plotted at the bottom of the viewport using the `rug` argument.

The resulting marginal effects plot is a `ggplot2` plot. This means it can be extended by plotting additional elements and using themes.

See Also

Other interpretation: `edgeprob()`, `interpret()`

Examples

```
## Not run:
# data preparation
data("florentine")
n <- network.size(flobusiness)
wealth <- get.vertex.attribute(flobusiness, "wealth")
priorates <- get.vertex.attribute(flobusiness, "priorates")
wealth.icov <- matrix(rep(wealth, n), ncol = n, byrow = TRUE)
priorates.icov <- matrix(rep(priorates, n), ncol = n, byrow = TRUE)
interac <- wealth.icov * priorates.icov

# estimate model with interaction effect
model <- ergm(flobusiness ~ edges + esp(1) + edgecov(wealth.icov)
               + edgecov(priorates.icov) + edgecov(interac))

# plot the interaction (note the additional optional ggplot2 elements)
marginalplot(model, var1 = "edgecov.wealth.icov",
             var2 = "edgecov.priorates.icov", inter = "edgecov.interac",
             color = "darkred", rug = TRUE, point = FALSE,
             xlab = "Priorates", ylab = "Wealth") +
  ggplot2::theme_bw() +
  ggplot2::ggtitle("Interaction effect")

## End(Not run)
```

`mtergm` *Estimate a TERGM by MCMC-MLE*

Description

Estimate a TERGM by Markov Chain Monte Carlo Maximum Likelihood Estimation

Usage

```
mtergm(formula, constraints = ~., returndata = FALSE, verbose = TRUE, ...)
```

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).
constraints	Constraints of the ERGM. See ergm for details.
returndata	Return the processed input data instead of estimating and returning the model? In the btergm case, this will return a data frame with the dyads of the dependent variable/network and the change statistics for all covariates. In the mtergm case, this will return a list object with the blockdiagonal network object for the dependent variable and blockdiagonal matrices for all dyadic covariates and the offset matrix for the structural zeros.
verbose	Print details about data preprocessing and estimation settings.
...	Further arguments to be handed over to the ergm function.

Details

The mtergm function computes TERGMs by MCMC MLE (or MPLE with uncorrected standard errors) via blockdiagonal matrices and structural zeros. It acts as a wrapper for the **ergm** package. The btergm function is faster than the mtergm function but is only asymptotically unbiased the longer the time series. The mtergm function yields unbiased estimates and standard errors but may suffer from degeneracy if the model is not specified in good keeping with the true data-generating process.

Author(s)

Philip Leifeld, Skyler J. Cranmer, Bruce A. Desmarais

References

Leifeld, Philip and Skyler J. Cranmer (2019): A Theoretical and Empirical Comparison of the Temporal Exponential Random Graph Model and the Stochastic Actor-Oriented Model. *Network Science* 7(1): 20-51. doi:10.1017/nws.2018.26.

Leifeld, Philip, Skyler J. Cranmer and Bruce A. Desmarais (2017): Temporal Exponential Random Graph Models with btergm: Estimation and Bootstrap Confidence Intervals. *Journal of Statistical Software* 83(6): 1-36. doi:10.18637/jss.v083.i06.

See Also

btergm tbergm

Examples

```
library("network")
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::network(mat)  # create network object
  networks[[i]] <- nw          # add network to the list
}

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

## Not run:
fit2 <- mtergm(networks ~ edges + istar(2) + edgecov(covariates))
summary(fit2)

## End(Not run)

# For examples with real data, see help("knecht") or help("alliances").
```

`mtergm-class` *An S4 Class to represent a fitted TERGM by MCMC-MLE*

Description

An S4 class to represent a fitted TERGM by MCMC-MLE.

Usage

```
## S4 method for signature 'mtergm'
show(object)

## S4 method for signature 'mtergm'
coef(object, invlogit = FALSE, ...)

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

timesteps.mtergm(object)

## S4 method for signature 'mtergm'
summary(object, ...)
```

Arguments

`object`	An `mtergm` object.
`invlogit`	Apply inverse logit transformation to the estimates and/or confidence intervals? That is, $\frac{1}{1+\exp(-x)}$, where x is the respective value.
`...`	Currently not in use.

Details

`mtergm` objects result from MCMC-MLE-based estimation of a TERGM via the `mtergm` function. They contain the coefficients, standard errors, and p-values, among other details.

Functions

- `show(mtergm)`: Show the coefficients of an `mtergm` object.
- `coef(mtergm)`: Return the coefficients of an `mtergm` object.
- `nobs(mtergm)`: Return the coefficients of an `mtergm` object.
- `timesteps.mtergm()`: Return the number of time steps saved in an `mtergm` object.
- `summary(mtergm)`: Return the coefficients of an `mtergm` object.

Slots

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

`se` Object of class `"numeric"`. The standard errors.

`pval` Object of class `"numeric"`. The p-values.

`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).

`formula2` The revised formula with the object references after applying the `tergmprepare` function.

`auto.adjust` Object of class `"logical"`. Indicates whether automatic adjustment of dimensions was done before estimation.

`offset` Object of class `"logical"`. Indicates whether an offset matrix with structural zeros was used.

`directed` Object of class `"logical"`. Are the dependent networks directed?

`bipartite` Object of class `"logical"`. Are the dependent networks bipartite?

`estimate` Estimate: either MLE or MPLE.

`loglik` Log likelihood of the MLE.

`aic` Akaike's Information Criterion.

`bic` Bayesian Information Criterion.

`ergm` The original `ergm` object as estimated by the `ergm` function in the **ergm** package.

`nvertices` Number of vertices.

`data` The data after processing by the `tergmprepare` function.

Author(s)

Philip Leifeld

See Also

Other tergm-classes: btergm-class, createBtergm(), createMtergm(), createTbergm(), tbergm-class

---

simulate.btergm *Simulate Networks from a* btergm *Object*

Description

Simulate networks from a btergm object using MCMC sampler.

Simulate networks from an mtergm object using MCMC sampler.

Usage

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

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

Arguments

object	A btergm or mtergm object, resulting from a call of the btergm or mtergm 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.
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 last (= most recent) network is used.
formula	A model formula from which the new network(s) should be simulated. By default, the formula is taken from the btergm object.
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?
...	Further arguments are handed over to the simulate_formula function in the **ergm** package.

Details

The simulate.btergm function is a wrapper for the simulate_formula function in the **ergm** package (see help("simulate.ergm")). 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. doi:10.1016/j.physa.2011.10.018.

Leifeld, Philip, Skyler J. Cranmer and Bruce A. Desmarais (2018): Temporal Exponential Random Graph Models with btergm: Estimation and Bootstrap Confidence Intervals. *Journal of Statistical Software* 83(6): 1–36. doi:10.18637/jss.v083.i06.

Examples

```
## Not run:
# fit a TERGM to some toy data
library("network")
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 covariate
  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)

# simulate 12 new networks from the last (= 10th) time step
sim1 <- simulate(fit, nsim = 12)

# simulate 1 network from the first time step
sim2 <- simulate(fit, index = 1)

# simulate network from t = 5 with larger covariate coefficient
coefs <- coef(fit)
coefs["edgecov.covariates[[i]]"] <- 0.5
sim3 <- simulate(fit, index = 5, coef = coefs)

## End(Not run)
```

---

tbergm *Estimate a TERGM using Bayesian estimation*

Description

Estimate a TERGM using Bayesian estimation.

Usage

```
tbergm(formula, returndata = FALSE, verbose = TRUE, ...)
```

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).
returndata	Return the processed input data instead of estimating and returning the model? In the btergm case, this will return a data frame with the dyads of the dependent variable/network and the change statistics for all covariates. In the mtergm

	case, this will return a list object with the blockdiagonal network object for the dependent variable and blockdiagonal matrices for all dyadic covariates and the offset matrix for the structural zeros.
verbose	Print details about data preprocessing and estimation settings.
...	Further arguments to be handed over to the bergm function in the **Bergm** package.

Details

The tbergm function computes TERGMs by Bayesian estimation via blockdiagonal matrices and structural zeros. It acts as a wrapper for the bergm function in the **Bergm** package.

Author(s)

Philip Leifeld

References

Caimo, Alberto and Nial Friel (2012): Bergm: Bayesian Exponential Random Graphs in R. *Journal of Statistical Software* 61(2): 1-25. doi:10.18637/jss.v061.i02.

See Also

btergm mtergm

---

tbergm-class *An S4 class to represent a fitted TERGM using Bayesian estimation*

Description

An S4 class to represent a fitted TERGM using Bayesian estimation.

Usage

```
## S4 method for signature 'tbergm'
show(object)

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

timesteps.tbergm(object)

## S4 method for signature 'tbergm'
summary(object, ...)
```

Arguments

object	A tbergm object.
...	Further arguments for the summary function in the **Bergm** package.

Details

tbergm objects result from Bayesian estimation of a TERGM using the tbergm function. They contain the original bergm object and some additional information.

Functions

- show(tbergm): Show the coefficients of a tbergm object.
- nobs(tbergm): Return the number of observations saved in a tbergm object.
- timesteps.tbergm(): Return the number of time steps saved in a tbergm object.
- summary(tbergm): Summary of a fitted tbergm object.

Slots

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).

formula2 The revised formula with the object references after applying the tergmprepare function.

auto.adjust Object of class "logical". Indicates whether automatic adjustment of dimensions was done before estimation.

offset Object of class "logical". Indicates whether an offset matrix with structural zeros was used.

directed Object of class "logical". Are the dependent networks directed?

bipartite Object of class "logical". Are the dependent networks bipartite?

estimate Estimate: "bergm" for Bayesian estimation.

bergm The original bergm object as estimated by the bergm function in the **Bergm** package.

nvertices Number of vertices.

data The data after processing by the tergmprepare function.

Author(s)

Philip Leifeld

See Also

Other tergm-classes: btergm-class, createBtergm(), createMtergm(), createTbergm(), mtergm-class

tergm-terms *Temporal dependencies for TERGMs*

Description

Network statistics that span multiple time points.

Transform a covariate using a function of time.

Usage

```
timecov(
  covariate,
  minimum = 1,
  maximum = length(covariate),
  transform = function(t) 1 + (0 * t) + (0 * t^2),
  onlytime = FALSE
)
```

Arguments

covariate	The list of networks or matrices for which to create a time covariate. This can be the list of networks on the left-hand side of the formula, in which case a time trend is created as a covariate list of matrices, or it can be a list of networks or matrices that is used as a dyadic covariate on the right-hand side of the formula, in which case an interaction effect between the time trend and the covariate is created. If used as a model term inside a formula, `covariate = NULL` is permitted, in which case the networks on the left-hand side will be used to form a time trend.
minimum, maximum	For time steps below the `minimum` value and above the `maximum` value, the time covariate is set to 0. These arguments can be used to create step-wise, discrete effects, for example to use a value of 0 up to an external event and 1 from that event onwards in order to control for influences of external events.
transform	In the default case, edges are modeled as being linearly increasingly important over time (i.e., a linear time trend). By tweaking the `transform` function, arbitrary functional forms of time can be tested. For example, `transform = sqrt` (for a geometrically decreasing time effect), `transform = function(x) x^2` (for a geometrically increasing time effect), `transform = function(t) t` (for a linear time trend) or polynomial functional forms (e.g., `transform = function(t) 0 + (1 * t) + (1 * t^2)`) can be used.
onlytime	If `TRUE`, return a time trend only. If `FALSE`, return an interaction between the time trend and the covariate. Note that the model term may need to be called twice or more inside a formula: one time to create the time trend main effect and one time for each interaction term; you also need to include the main effects for the covariates separately using `edgecov` or similar terms.

Details

In addition to the ERGM user terms that can be estimated within a single network (see ergm-terms), the **btergm** package provides additional model terms that can be used within a formula. These additional statistics span multiple time periods and are therefore called "temporal dependencies." Examples include memory terms (i.e., positive autoregression, dyadic stability, edge innovation, or edge loss), delayed reciprocity or mutuality, and time covariates (i.e., functions of time or interactions with time):

`delrecip(mutuality = FALSE, lag = 1)` The `delrecip` term checks for delayed reciprocity. For example, if node `j` is tied to node `i` at `t = 1`, does this lead to a reciprocation of that tie back from `i` to `j` at `t = 2`? If `mutuality = TRUE` is set, this extends not only to ties, but also non-ties. That is, if `i` is not tied to `j` at `t = 1`, will this lead to `j` not being tied to `i` at `t = 2`, in addition to positively reciprocal patterns over time? The `lag` argument controls the size of the temporal lag: with `lag = 1`, reciprocity over one consecutive time period is checked. Note that as `lag` increases, the number of time steps on the dependent variable decreases.

`memory(type = "stability", lag = 1)` Memory terms control for the impact of a previous network on the current network. Four different types of memory terms are available: positive autoregression (`type = "autoregression"`) checks whether previous ties are carried over to the current network; dyadic stability (`type = "stability"`) checks whether both edges and non-edges are stable between the previous and the current network; edge loss (`type = "loss"`) checks whether ties in the previous network have been dissolved and no longer exist in the current network; and edge innovation (`type = "innovation"`) checks whether previously unconnected nodes have the tendency to become tied in the current network. The `lag` argument accepts integer values and controls whether the comparison is made with the previous network (`lag = 1`), the pre-previous network (`lag = 2`) etc. Note that as `lag` increases, the number of time steps on the dependent variable decreases.

`timecov(x = NULL, minimum = 1, maximum = NULL, transform = function(t) t)` The `timecov` model term checks for linear or non-linear time trends with regard to edge formation. Optionally, this can be combined with a covariate to create an interaction effect between a dyadic covariate and time in order to test whether the importance of a covariate increases or decreases over time. In the default case, edges modeled as being linearly increasingly important over time. By tweaking the `transform` function, arbitrary functional forms of time can be tested. For example, `transform = sqrt` (for a geometrically decreasing time effect), `transform = function(x) x^2` (for a geometrically increasing time effect), `transform = function(t) t` (for a linear time trend) or polynomial functional forms (e.g., `0 + (1 * t) + (1 * t^2)`) can be used. For time steps below the `minimum` value and above the `maximum` value, the time covariate is set to 0. These arguments can be used to create step-wise, discrete effects, for example to use a value of 0 up to an external event and 1 from that event onwards in order to control for influences of external events.

The `timecov` model term checks for linear or non-linear time trends with regard to edge formation. Optionally, this can be combined with a covariate to create an interaction effect between a dyadic covariate and time in order to test whether the importance of a covariate increases or decreases over time. The function can either be used in a formula with `btergm`, `mtergm`, or `tbergm`, or it can be executed directly for manual inclusion of the results as a covariate.

Functions

- `timecov()`: Time trends and temporal covariate interactions

References

Leifeld, Philip, Skyler J. Cranmer and Bruce A. Desmarais (2017): Temporal Exponential Random Graph Models with btergm: Estimation and Bootstrap Confidence Intervals. *Journal of Statistical Software* 83(6): 1-36. doi:10.18637/jss.v083.i06.

`tergmprepare` *Prepare data structure for TERGM estimation, including composition change*

Description

Prepare data structure for TERGM estimation, including composition change.

Usage

```
tergmprepare(formula, offset = TRUE, blockdiag = FALSE, verbose = TRUE)
```

Arguments

`formula`	The original formula provided by the user, given the data structures in the workspace.
`offset`	Indicates whether absent nodes should be added where they are missing (`offset = TRUE`) or removed where they are not missing (`offset = FALSE`).
`blockdiag`	Should the time steps be arranged in a blockdiagonal matrix for use with MCMC-MLE or Bayesian estimation (`blockdiag = TRUE`), or should they be kept as items in a list for use with `btergm` (`blockdiag = FALSE`)?
`verbose`	Report details about dimension adjustment?

Details

This is a helper function that adjusts the dimensions of networks or covariates within a given time step to each other by removing nodes that are not present across all objects within a time step or by adding nodes where they are missing (and simultaneously adding entries to a list of structural zero matrices to indicate their absence). It is not necessary to have identical (numbers of) nodes across time steps as long as the dimensions of the matrices, networks, and vectors match cross-sectionally within time steps, given that temporal dependency terms like memory are interpreted as dyadic covariates in a given time step. This helper function also creates these dyadic covariate data structures for some of the custom temporal model terms, such as `memory` and `delrecip`. Leifeld, Cranmer and Desmarais (2018) contain additional details on composition change, dimension adjustment of matrices, and temporal dependencies. Note that this function should not normally be used by the end user. It is automatically called internally by the estimation functions to make the dimensions of all objects conformable to each other for estimation. Use this function only for diagnostic purposes!

Value

A list with the following slots:

lhs.original A character object containing the original name of the object on the left-hand side of the formula provided by the user. This is saved here because the formula is manipulated such that the left-hand side of the formula contains a new item `networks[[i]]`.

networks The list of networks on the left-hand side of the formula after dimension adjustment, or a blockdiagonal network representing the left-hand side of the formula after dimension adjustment if argument `blockdiag = TRUE` was used.

num.vertices The maximum number of nodes of any time point after adjustment of dimensions.

directed Are the networks directed?

bipartite Are the networks bipartite?

form The formula after manipulation and adjustment of the data, including `networks[[i]]` on the left-hand side and an added offset covariate on the right-hand side of the formula, in addition to added indices for the covariate terms.

time.steps The number of time steps of the dataset.

rhs.terms The right-hand side of the formula after adjustment, as a vector of `character` objects representing the terms.

covnames A `character` vector containing the names of the objects in which the networks and covariates are stored, according to the manipulated formula. This includes `"networks"` (for the left-hand side of the formula) and all objects containing exogenous covariates on the right-hand side of the formula after manipulation.

... Each of the covariates mentioned in the slot `covnames` is stored as an element of the list, either as a list of matrices or networks (if `blockdiag = FALSE`) or as a matrix or network object (if `blockdiag = TRUE`).

auto.adjust Did the function have to adjust the dimensions of the networks or covariates at all?

nvertices A matrix containing the number of nodes in the rows and columns of each object at each time step, after adjustment.

offsmat A list of offset covariate matrices or a large blockdiagonal offset covariate matrix containing structural zeros. If `offset = FALSE`, this matrix or list of matrices will contain only zeros. If `offset = TRUE`, they will contain ones where nodes were absent in the original data.

Author(s)

Philip Leifeld

Index