Package 'lolog'

July 22, 2025

Maintainer Ian E. Fellows <ian@fellstat.com>

License MIT + file LICENCE

Title Latent Order Logistic Graph Models

LinkingTo Rcpp, BH

Type Package

LazyLoad yes

Description Estimation of Latent Order Logistic (LOLOG) Models for Networks.
LOLOGs are a flexible and fully general class of statistical graph models.
This package provides functions for performing MOM, GMM and variational
inference. Visual diagnostics and goodness of fit metrics are provided.
See Fellows (2018) <doi:10.48550/arXiv.1804.04583> for a detailed description of the methods.

Version 1.3.1

Depends R (>= 4.0.0), methods, Rcpp (>= 0.9.4)

Imports network, parallel, ggplot2, reshape2, intergraph, Matrix

Suggests testthat, inline, knitr, rmarkdown, ergm, BH, igraph

URL https://github.com/statnet/lolog

RcppModules lolog

RoxygenNote 7.2.2

VignetteBuilder knitr

NeedsCompilation yes

Author Ian E. Fellows [aut, cre],
Mark S. Handcock [ctb]

Repository CRAN

Date/Publication 2023-12-07 12:40:02 UTC

Contents

`as.BinaryNet` *Convert to either an UndirectedNet or DirectedNet object*

Description

Convert to either an UndirectedNet or DirectedNet object

Usage

```
as.BinaryNet(x, ...)
```

Arguments

x	the object
...	unused

Details

Converts network objects to BinaryNets. This function also converts other graph formats, such as igraph and tidygraph, utilizing intergraph::asNetwork.

Value

either an Rcpp_UndirectedNet or Rcpp_DirectedNet object

Examples

```
data(ukFaculty)
net <- as.BinaryNet(ukFaculty)
net
```

---

as.BinaryNet.default *Convert to either an UndirectedNet or DirectedNet object*

Description

Convert to either an UndirectedNet or DirectedNet object

Usage

```
## Default S3 method:
as.BinaryNet(x, ...)
```

Arguments

x	the object
...	unused

Details

Converts network objects to BinaryNets. This function also converts other graph formats, such as igraph and tidygraph, utilizing intergraph::asNetwork.

Value

either an Rcpp_UndirectedNet or Rcpp_DirectedNet object

Examples

```
data(ukFaculty)
net <- as.BinaryNet(ukFaculty)
net
```

`as.network` *Network conversion*

Description

Network conversion

Arguments

`x`	The object
`...`	Additional parameters

`as.network.Rcpp_DirectedNet`
Convert a DirectedNet to a network object

Description

Convert a DirectedNet to a network object

Usage

```
## S3 method for class 'Rcpp_DirectedNet'
as.network(x, ...)
```

Arguments

`x`	the object
`...`	unused

Value

A network object

See Also

`DirectedNet`

Examples

```
el <- matrix(c(1,2),ncol=2)

#make an UndirectedNet with one edge and 5 nodes
net <- new(UndirectedNet, el, 5L)

nw <- as.network(net)
nw
```

`as.network.Rcpp_UndirectedNet`
Convert a UndirectedNet to a network object

Description

Convert a UndirectedNet to a network object

Usage

```
## S3 method for class 'Rcpp_UndirectedNet'
as.network(x, ...)
```

Arguments

`x`	the object
`...`	unused

Value

A network object

See Also

`UndirectedNet`

Examples

```
el <- matrix(c(1,2),ncol=2)

#make an UndirectedNet with one edge and 5 nodes
net <- new(UndirectedNet, el, 5L)
net[1:5,1:5]

nw <- as.network(net)
nw
```

`BinaryNet` *BinaryNet*

Description

BinaryNet

Details

Rcpp_DirectedNet and Rcpp_UndirectedNet are the native network classes for the lolog package. They are designed for algorithmic performance, and are thin wrappers for an underlying C++ object. These network objects can be passed back and forth between R and C++ with little overhead. Because they are pointers to C++ objects, serialization via 'save' or 'dput' are not supported

`calculateStatistics` *Calculate network statistics from a formula*

Description

Calculate network statistics from a formula

Usage

```
calculateStatistics(formula)
```

Arguments

`formula`	A lolog formula (See `lolog`).

Examples

```
data(ukFaculty)
calculateStatistics(ukFaculty ~ edges + mutual + triangles)
```

`call-symbols` *Internal Symbols*

Description

Internal symbols used to access compiles code.

coef.lolog *Extracts estimated model coefficients.*

Description

Extracts estimated model coefficients.

Usage

```
## S3 method for class 'lolog'
coef(object, ...)
```

Arguments

object	A ‘lolog‘ object.
...	unused

Examples

```
# Extract parameter estimates as a numeric vector:
data(ukFaculty)
fit <- lolog(ukFaculty ~ edges)
coef(fit)
```

createCppModel *Creates a model*

Description

Creates a model

Usage

```
createCppModel(formula, cloneNet = TRUE, theta = NULL)
```

Arguments

formula	the model formula
cloneNet	create a deep copy of the network within the model object
theta	the model parameters.

Details

Creates a C++ Model object. In general this isn’t needed by most users of the package.

Examples

```
data(ukFaculty)
model <- createCppModel(ukFaculty ~ edges)
model$calculate()
model$statistics()
```

---

createLatentOrderLikelihood *Creates a probability model for a latent ordered network model*

Description

Creates a probability model for a latent ordered network model

Usage

```
createLatentOrderLikelihood(formula, theta = NULL)
```

Arguments

formula	A LOLOG formula. See link{lolog}
theta	Parameter values.

Value

An Rcpp object representing the likelihood model

Examples

```
# See the methods of the objects returned by this function
UndirectedLatentOrderLikelihood
DirectedLatentOrderLikelihood

# A Barabasi-Albert type graph model with 1000 vertices
el <- matrix(0, nrow=0, ncol=2)
net <- new(UndirectedNet, el, 1000L)
lolik <- createLatentOrderLikelihood(net ~ preferentialAttachment(), theta=1)
banet <- lolik$generateNetwork()$network # generate a random network from the model
degrees <- banet$degree(1:1000)
hist(degrees, breaks=100) # plot the degree distribution
order <- banet[["__order__"]] # The vertex inclusion order

# Earlier nodes have higher degrees
library(ggplot2)
qplot(order, degrees, alpha=I(.25)) + geom_smooth(method="loess")
```

gofit *Conduct goodness of fit diagnostics*

Description

Conduct goodness of fit diagnostics

Usage

```
gofit(object, ...)
```

Arguments

object	the object to evaluate
...	additional parameters

Details

see gofit.lolog

gofit.lolog *Goodness of Fit Diagnostics for a LOLOG fit*

Description

Goodness of Fit Diagnostics for a LOLOG fit

Usage

```
## S3 method for class 'lolog'
gofit(object, formula, nsim = 100, ...)
```

Arguments

object	the object to evaluate
formula	A formula specifying the statistics on which to evaluate the fit
nsim	The number of simulated statistics
...	additional parameters

Examples

```
library(network)
data(ukFaculty)

# Delete vertices missing group
delete.vertices(ukFaculty, which(is.na(ukFaculty %v% "Group")))

# A dyad independent model
fitind <- lolog(ukFaculty ~ edges() + nodeMatch("GroupC") + nodeCov("GroupC"))
summary(fitind)

# Check gof on degree distribution (bad!)
gind <- gofit(fitind, ukFaculty ~ degree(0:50))
gind
plot(gind)

#check gof on esp distribution (bad!)
gind <- gofit(fitind, ukFaculty ~ esp(0:25))
gind
plot(gind)

## Not run:

#include triangles and 2-stars (in and out)
fitdep <- lolog(ukFaculty ~ edges() + nodeMatch("GroupC") + nodeCov("GroupC") +
              triangles + star(2, direction="in") + star(2, direction="out"), nsamp=1500)
summary(fitdep)

# Check gof on (in + out) degree distribution (good!)
gdep <- gofit(fitdep, ukFaculty ~ degree(0:50))
gdep
plot(gdep)

#check gof on esp distribution (good!)
gdep <- gofit(fitdep, ukFaculty ~ esp(0:25))
gdep
plot(gdep)


## End(Not run)
```

`inlineLologPlugin` *An lolog plug-in for easy C++ prototyping and access*

Description

An lolog plug-in for easy C++ prototyping and access

The inline plug-in for lolog

Usage

```
inlineLologPlugin(...)

inlineLologPlugin
```

Arguments

...	plug-in arguments

Details

The lolog Rcpp plugin allows for the rapid prototyping of compiled code. new functions can be registered and exposed using cppFunction and new statistics can be compiled and registered using sourceCpp.

See Also

cppFunction, sourceCpp, cppFunction

Examples

```
## Not run:
# This creates a function in C++ to create an empty network of size n
# and expose it to R.
src <- "
lolog::BinaryNet<lolog::Directed> makeEmptyNetwork(const int n){
Rcpp::IntegerMatrix tmp(0,2);
lolog::BinaryNet<lolog::Directed> net(tmp, n);
return net;
}
"
Rcpp::registerPlugin("lolog",inlineLologPlugin)
emptyNetwork <- cppFunction(src,plugin="lolog")
net <- emptyNetwork(10L)
net[1:10,1:10]


## End(Not run)
```

LatentOrderLikelihood *LatentOrderLikelihood*

Description

LatentOrderLikelihood

lazega — *Collaboration Relationships Among Partners at a New England Law Firm*

Description

This data set comes from a network study of corporate law partnership that was carried out in a Northeastern US corporate law firm, referred to as SG&R, 1988-1991 in New England.

Usage

```
data(lazega)
```

Licenses and Citation

CC BY 4.0. When publishing results obtained using this data set, the original authors (Lazega, 2001) should be cited, along with this R package.

Copyright

Creative Commons Attribution-Share Alike 4.0 International License, see https://creativecommons.org/licenses/by/4.0/ for details.

Source

See http://elazega.fr/?page_id=609 and https://www.stats.ox.ac.uk/~snijders/siena/Lazega_lawyers_data.htm

References

Lazega, Emmanuel (2001), The Collegial Phenomenon: The Social Mechanisms of Cooperation among Peers in a Corporate Law Partnership, Oxford: Oxford University Press

lolog — *Fits a LOLOG model via Monte Carlo Generalized Method of Moments*

Description

lolog is used to fit Latent Order Logistic Graph (LOLOG) models. LOLOG models are motivated by the idea of network growth where the network begins empty, and edge variables are sequentially 'added' to the network with an either unobserved, or partially observed order s. Conditional upon the inclusion order, the probability of an edge has a logistic relationship with the change in network statistics.

Usage

```
lolog(
  formula,
  auxFormula = NULL,
  theta = NULL,
  nsamp = 1000,
  includeOrderIndependent = TRUE,
  targetStats = NULL,
  weights = "full",
  tol = 0.1,
  nHalfSteps = 10,
  maxIter = 100,
  minIter = 2,
  startingStepSize = 0.1,
  maxStepSize = 0.5,
  cluster = NULL,
  verbose = TRUE
)
```

Arguments

formula	A lolog formula for the sufficient statistics (see details).
auxFormula	A lolog formula of statistics to use for moment matching.
theta	Initial parameters values. Estimated via lologVariational if NULL.
nsamp	The number of sample networks to draw at each iteration.
includeOrderIndependent	If TRUE, all order independent terms in formula are used for moment matching.
targetStats	A vector of network statistics to use as the target for the moment equations. If NULL, the observed statistics for the network are used.
weights	The type of weights to use in the GMM objective. Either 'full' for the inverse of the full covariance matrix or 'diagonal' for the inverse of the diagonal of the covariance matrix.
tol	The Hotelling's T^2 p-value tolerance for convergence for the transformed moment conditions.
nHalfSteps	The maximum number of half steps to take when the objective is not improved in an iteration.
maxIter	The maximum number of iterations.
minIter	The minimum number of iterations.
startingStepSize	The starting dampening of the parameter update.
maxStepSize	The largest allowed value for dampening.
cluster	A parallel cluster to use for graph simulation.
verbose	Level of verbosity 0-3.

Details

LOLOG represents the probability of a tie, given the network grown up to a time point as

$$\text{logit}\big(p(y_{s_t} = 1|\eta, y^{t-1}, s_{\leq t})\big) = \theta \cdot c(y_{s_t} = 1|y^{t-1}, s_{\leq t})$$

where $s_{\leq t}$ is the growth order of the network up to time t, y^{t-1} is the state of the graph at time $t-1$. $c(y_{s_t}|y^{t-1}, s_{\leq t})$ is a vector representing the change in graph statistics from time $t-1$ to t if an edge is present, and θ is a vector of parameters.

The motivating growth order proceeds 'by vertex.' The network begins 'empty' and then vertices are 'added' to the network sequentially. The order of vertex inclusion may be random or fixed. When a vertex 'enters' the network, each of the edge variables connecting it and vertices already in the network are considered for edge creation in a completely random order.

LOLOG formulas contain a network, DirectedNet or UndirectedNet object on the left hand side. the right hand side contains the model terms used. for example,

`net ~ edges`

represents and Erdos-Renyi model and

`net ~ edges + preferentialAttachment()`

represents a Barabasi-Albert model. See `lolog-terms` for a list of allowed model statistics

Conditioning on (partial) vertex order can be done by placing an ordering variable on the right hand side of the '|' operator, as in

`net ~ edges + preferentialAttachment() | order`

'order' should be a numeric vector with as many elements as there are vertices in the network. Ties are allowed. Vertices with higher order values will always be included later. Those with the same values will be included in a random order in each simulated network.

offsets and constraints are specified by wrapping them with either `offset()` or `constraint()`, for example, the following specifies an Erdos-Renyi model with the constraint that degrees must be less that 10

`net ~ edges + constraint(boundedDegree(0L, 10L))`

If the model contains any order dependent statistics, additional moment constraints must be specified in `auxFormula`. Ideally these should be chosen to capture the features modeled by the order dependent statistics. For example, `preferentialAttachment` models the degree structure, so we might choose two-stars as a moment constraint.

`lolog(net ~ edges + preferentialAttachment(), net ~ star(2))`

will fit a Barabasi-Albert model with the number of edges and number of two-stars as moment constraints.

Value

An object of class 'lolog'. If the model is dyad independent, the returned object will also be of class "lologVariational" (see `lologVariational`, otherwise it will also be a "lologGmm" object.

lologGmm objects contain:

`method`	"Method of Moments" for order independent models, otherwise "Generalized Method of Moments"

formula	The model formula
auxFormula	The formula containing additional moment conditions
theta	The parameter estimates
stats	The statistics for each network in the last iteration
estats	The expected stats (G(y,s)) for each network in the last iteration
obsStats	The observed h(y) network statistics
targetStats	The target network statistics
obsModelStats	The observed g(y,s) network statistics
net	A network simulated from the fit model
grad	The gradient of the moment conditions (D)
vcov	The asymptotic covariance matrix of the parameter estimates
likelihoodModel	An object of class *LatentOrderLikelihood at the fit parameters

Examples

```
library(network)
set.seed(1)
data(flo)
flomarriage <- network(flo,directed=FALSE)
flomarriage %v% "wealth" <- c(10,36,27,146,55,44,20,8,42,103,48,49,10,48,32,3)

# A dyad independent model
fit <- lolog(flomarriage ~ edges + nodeCov("wealth"))
summary(fit)

# A dyad dependent model with 2-stars and triangles
fit2 <- lolog(flomarriage ~ edges + nodeCov("wealth") + star(2) + triangles, verbose=FALSE)
summary(fit2)

## Not run:

# An order dependent model
fit3 <- lolog(flomarriage ~ edges + nodeCov("wealth") + preferentialAttachment(),
              flomarriage ~ star(2:3), verbose=FALSE)
summary(fit3)

# Try something a bit more real
data(ukFaculty)

# Delete vertices missing group
delete.vertices(ukFaculty, which(is.na(ukFaculty %v% "Group")))

fituk <- lolog(ukFaculty ~ edges() + nodeMatch("GroupC") + nodeCov("GroupC") + triangles + star(2))
summary(fituk)
plot(fituk$net, vertex.col= ukFaculty %v% "Group" + 2)
```

```
## End(Not run)
```

`lolog-terms` *LOLOG Model Terms*

Description

LOLOG Model Terms

Statistic Descriptions

`edges` **(dyad-independent) (order-independent) (directed) (undirected)** *Edges:* This term adds one network statistic equal to the number of edges (i.e. nonzero values) in the network.

`star(k, direction="in")` **(order-independent) (directed) (undirected)** The `k` argument is a vector of distinct integers. This term adds one network statistic to the model for each element in `k`. The ith such statistic counts the number of distinct `k[i]`-stars in the network, where a k-star is defined to be a node N and a set of k different nodes $\{O_1, \ldots, O_k\}$ such that the ties $\{N, O_i\}$ exist for $i = 1, \ldots, k$. For directed networks, direction indicates whether the count is of in-stars (direction="in") or out-stars (direction="out")

`triangles()` **(order-independent) (directed) (undirected)** This term adds one statistic to the model equal to the number of triangles in the network. For an undirected network, a triangle is defined to be any set $\{(i, j), (j, k), (k, i)\}$ of three edges. For a directed network, a triangle is defined as any set of three edges $(i \rightarrow j)$ and $(j \rightarrow k)$ and either $(k \rightarrow i)$ or $(k \leftarrow i)$.

`clustering()` **(order-independent) (undirected)** The global clustering coefficient, defined as the number of triangles over the number of possible triangles https://en.wikipedia.org/wiki/Clustering_coefficient, or 3 * triangles / 2-stars.

`transitivity()` **(order-independent) (undirected)** The Soffer-Vazquez transitivity. This is clustering metric that adjusts for large degree differences and is described by C in Equation 6 of #' https://pubmed.ncbi.nlm.nih.gov/16089694/. Note The approximation of the number of possible shared neighbors between node i and j of min(d_i,d_j) - 1 in this implementation.

`mutual()` **(order-independent) (directed)** A count of the number of pairs of actors i and j for which $(i \rightarrow j)$ and $(j \rightarrow i)$ both exist.

`nodeMatch(name)` **(dyad-independent) (order-independent) (directed) (undirected)** For categorical network nodal variable 'name,' the number of edges between nodes with the same variable value.

`nodeMix(name)` **(dyad-independent) (order-independent) (directed) (undirected)** For categorical network nodal variable 'name,' adds one statistic for each combination of levels of the variable equal to the count of edges between those levels.

`degree(d, direction="undirected", lessThanOrEqual=FALSE)` **(order-independent) (directed) (undirected)** The `d` argument is a vector of distinct integers. This term adds one network statistic to the model for each element in `d`; the ith such statistic equals the number of nodes in the network of degree `d[i]`, i.e. with exactly `d[i]` edges. For directed networks if direction="undirected" degree is counted as the sum of the in and out degrees of a node. If direction="in" then in-degrees are used and direction="out" indicates out-degrees.

If lessThanOrEqual=TRUE, then the count is the number of nodes with degree less than or equal to d.

`twoPath` **(order-independent) (directed) (undirected)** This term adds one statistic to the model, equal to the number of 2-paths in the network. For a directed network this is defined as a pair of edges $(i \to j), (j \to k)$, where i and j must be distinct. That is, it is a directed path of length 2 from i to k via j. For directed networks a 2-path is also a mixed 2-star. For undirected networks a twopath is defined as a pair of edges $\{i, j\}, \{j, k\}$. That is, it is an undirected path of length 2 from i to k via j, also known as a 2-star.

`degreeCrossProd() (order-independent) (undirected)` This term adds one network statistic equal to the mean of the cross-products of the degrees of all pairs of nodes in the network which are tied.

`nodeCov(name)` **(dyad-independent) (order-independent) (directed) (undirected)** The `name` argument is a character string giving the name of a numeric attribute in the network's vertex attribute list. This term adds a single network statistic to the model equaling the sum of `name(i)` and `name(j)` for all edges (i, j) in the network. For categorical variables, levels are coded as 1,..,nlevels‘.

`edgeCov(x, name=NULL)` **(dyad-independent) (order-independent) (directed) (undirected)** The `x` argument is a square matrix of covariates, one for each possible edge in the network. This term adds one statistic to the model, equal to the sum of the covariate values for each edge appearing in the network. The `edgeCov` term applies to both directed and undirected networks. For undirected networks the covariates are also assumed to be undirected. If present, the `name` argument is a character string providing a name for the `edgeCov` term. The name will be "edgeCov.<name>". It is recommended that all `edgeCov` terms be given explicit names. In particular, if two unnamed `edgeCov` terms are supplied an error will occur (as they will have the same default name "edgeCov.".

`edgeCovSparse(x, name=NULL)` **(dyad-independent) (order-independent) (directed) (undirected)** Identical to edgeCov, except `x` should be a sparse matrix. This is especially useful for larger networks, where passing a dense matrix to `edgeCov` is too memory intensive.

`gwesp(alpha)` **(order-independent) (directed) (undirected)** This term is just like `gwdsp` except it adds a statistic equal to the geometrically weighted *edgewise* (not dyadwise) shared partner distribution with decay parameter `alpha` parameter, which should be non-negative.

`gwdegree(alpha, direction="undirected")` **(order-independent) (directed) (undirected)** This term adds one network statistic to the model equal to the weighted degree distribution with decay controlled by the `decay` parameter. The `alpha` parameter is the same as theta_s in equation (14) in Hunter (2007).

For directed networks if direction="undirected" degree is counted as the sum of the in and out degrees of a node. If direction="in" then in-degrees are used ans direction="out" indicates out-degrees.

`gwdsp(alpha)` **(order-independent) (directed) (undirected)** This term adds one network statistic to the model equal to the geometrically weighted dyadwise shared partner distribution with decay parameter `decay` parameter, which should be non-negative.

`esp(d, type=2)` **(order-independent) (directed) (undirected)** This term adds one network statistic to the model for each element in `d` where the ith such statistic equals the number of *edges* (rather than dyads) in the network with exactly `d[i]` shared partners. This term can be used with directed and undirected networks. For directed networks the count depends on type:

type = 1 : from -> to -> nbr -> from

type = 2 : from -> to <- nbr <- from (homogeneous)

type = 3 : either type 1 or 2

type = 4 : all combinations of from -> to <-> nbr <-> from

`geoDist(long, lat, distCuts=Inf)` **(dyad-independent) (order-independent) (undirected)** given nodal variables for longitude and latitude, calculates the sum of the great circle distance between connected nodes. distCuts splits this into separate statistics that count the sum of the minimum of the cut point and the distance.

`dist(names` **(dyad-independent) (order-independent) (undirected)** Calculates a statistic equal to the sum of the euclidean distances between connected nodes on the numeric nodal variables specified in names.

`preferentialAttachment(k=1, direction="in")` **(directed) (undirected)** An order dependent preferential attachment term. For each edge, adds

log((k+degree) / (n * (meanDegree + k)))

where degree is the current degree of the acting node, n is the network size, and meanDegree is the mean degree of the network. This depends upon the order in which edges are added. For directed networks, if direction="in" the in-degrees are used. If it is "out" the out degrees are used, otherwise "undirected" means that the sum of the in and out degrees are used.

`sharedNbrs(k=1)` **(undirected)** for each edge adds

log(k + shared / minDeg)

where shared is the current number of shared neighbors between the two nodes, and minDeg is the minimum of the current degrees of the two nodes (i.e. the number of possible shared neighbors).

`nodeLogMaxCov(name)` **(order-independent) (undirected)** For each edge (i,j) and nodal variable variable, add to the statistic

log(max(variable[i],variable[j]))

If the variable is a (partial) rank order of nodal inclusion into the network, this statistic can be useful in modeling the mean degree over the course of the growth process.

`nodeFactor(name, direction="undirected")` **(order-independent) (undirected) (directed)** The `name` argument is a character vector giving one or more names of categorical attributes in the network's vertex attribute list. This term adds multiple network statistics to the model, one for each of (a subset of) the unique values of the `attrname` attribute (or each combination of the attributes given). Each of these statistics gives the number of times a node with that attribute or those attributes appears in an edge in the network. In particular, for edges whose endpoints both have the same attribute values, this value is counted twice. For directed networks, if direction="in" then in-edges are used and direction="out" indicates out-edges.

`absDiff(name, power=1)` **(order-independent) (undirected) (directed)** The `name` argument is a character string giving the name of one or mode quantitative attribute in the network's vertex attribute list. This term adds one network statistic to the model equaling the sum of `sum(abs(name[i]-name[j])^pow)` for all edges (i,j) in the network.

Constraint Descriptions

`boundedDegree(lower,upper)` **(order-independent) (undirected)** Adds a constraint that the degrees for the network must be between lower and upper.

LologModels *Models*

Description

Models

logPackageSkeleton *Create a skeleton for a package extending lolog*

Description

Create a skeleton for a package extending lolog

Usage

```
logPackageSkeleton(path = ".")
```

Arguments

path	where to create the package

Details

lolog is a modular package, and can be extended at both the R and C++ level. This function will build a package skeleton that can be used as a starting point for development. To create the package in the current directory run:

```
logPackageSkeleton()
```

Build and install the package from the command line with

```
R CMD build LologExtension
```

```
R CMD INSTALL LologExtension_1.0.tar.gz
```

See Also

inlineLologPlugin

Examples

```
## Not run:

#install package
logPackageSkeleton()
system("R CMD build LogExtension")
system("R CMD INSTALL LogExtension_1.0.tar.gz")

library(LogExtension) #Load package

# Run model with new minDegree statistic
library(network)
m <- matrix(0,20,20)
for(i in 1:19) for(j in (i+1):20) m[i,j] <- m[j,i] <- rbinom(1,1,.1)
g <- network(m, directed=FALSE)
fit <- logVariational(g ~ edges() + minDegree(1L))
summary(fit)


## End(Not run)
```

`logVariational` *Fits a latent ordered network model using Monte Carlo variational inference*

Description

Fits a latent ordered network model using Monte Carlo variational inference

Usage

```
logVariational(
  formula,
  nReplicates = 5L,
  dyadInclusionRate = NULL,
  edgeInclusionRate = NULL,
  targetFrameSize = 5e+05
)
```

Arguments

`formula`	A lolog formula. See `link{lolog}`
`nReplicates`	An integer controlling how many dyad ordering to perform.
`dyadInclusionRate`	Controls what proportion of non-edges in each ordering should be dropped.
`edgeInclusionRate`	Controls what proportion of edges in each ordering should be dropped.

`targetFrameSize`
: Sets dyadInclusionRate so that the model frame for the logistic regression will have on average this amount of observations.

Details

This function approximates the maximum likelihood solution via a variational inference on the graph (y) over the latent edge variable inclusion order (s). Specifically, it replaces the conditional probability p(s | y) by p(s). If the LOLOG model contains only dyad independent terms, then these two probabilities are identical, and thus variational inference is exactly maximum likelihood inference. The objective function is

$$E_{p(s)}\left(\log p(y|S,\theta)\right)$$

This can be approximated by drawing samples from p(s) to approximate the expectation. The number of samples is controlled by the nReplicates parameter. The memory required is on the order of nReplicates * (# of dyads). For large networks this can be impractical, so adjusting dyadInclusionRate and edgeInclusionRate allows one to down sample the # of dyads in each replicate. By default these are set attempting to achieve as equal a number of edges and non-edges as possible while targeting a model frame with targetFrameSize number of rows.

If the model is dyad independent, replicates are redundant, and so nReplicates is set to 1 with a note.

The functional form of the objective function is equivalent to logistic regression, and so the `glm` function is used to maximize it. The asymptotic covariance of the parameter estimates is calculated using the methods of Westling (2015).

Value

An object of class c('lologVariationalFit','lolog','list') consisting of the following items:

`formula` The model formula

`method` "variational"

`theta` The fit parameter values

`vcov` The asymptotic covariance matrix for the parameter values.

`nReplicates` The number of replicates

`dyadInclusionRate`
: The rate at which non-edges are included

`edgeInclusionRate`
: The rate at which edges are included

`allDyadIndependent`
: Logical indicating model dyad independence

`likelihoodModel`
: An object of class *LatentOrderLikelihood at the fit parameters

`outcome` The outcome vector for the logistic regression

`predictors` The change statistic predictor matrix for the logistic regression

References

Westling, T., & McCormick, T. H. (2015). Beyond prediction: A framework for inference with variational approximations in mixture models. arXiv preprint arXiv:1510.08151.

Examples

```
library(network)
data(ukFaculty)

# Delete vertices missing group
delete.vertices(ukFaculty, which(is.na(ukFaculty %v% "Group")))

fit <- lologVariational(ukFaculty ~ edges() + nodeMatch("GroupC"),
                        nReplicates=1L, dyadInclusionRate=1)
summary(fit)
```

---

plot.gofit *Plots a gofit object*

Description

Plots a gofit object

Usage

```
## S3 method for class 'gofit'
plot(
  x,
  y,
  type = c("line", "box"),
  scaling = c("none", "std", "sqrt"),
  lineAlpha = 0.06,
  lineSize = 1,
  ...
)
```

Arguments

x	the gofit object
y	unused
type	type of plot, boxplot or lineplot
scaling	type of scaling of the network statistics. If "std", network statistics are scaling by subtracting off the observed statistics and scaling by the standard deviation. If "sqrt", network statistics are plotted on the square root scale (The square root is the variance stabilizing transformation for a Poisson random variable). The default is "none", where by the network statistics are not scaled.

lineAlpha	The transparency of the simulated statistics lines
lineSize	The width of the lines
...	passed to either boxplot or geom_line

Examples

```
library(network)
data(ukFaculty)

# Delete vertices missing group
delete.vertices(ukFaculty, which(is.na(ukFaculty %v% "Group")))

# A dyad independent model
fitind <- lolog(ukFaculty ~ edges() + nodeMatch("GroupC") + nodeCov("GroupC"))
summary(fitind)

# Check gof on degree distribution (bad!)
gind <- gofit(fitind, ukFaculty ~ degree(0:50))
plot(gind)
plot(gind, type="box")
```

---

plot.lologGmm *Conduct Monte Carlo diagnostics on a lolog model fit*

Description

This function creates simple diagnostic plots for MC sampled statistics produced from a lolog fit.

Usage

```
## S3 method for class 'lologGmm'
plot(x, type = c("histograms", "target", "model"), ...)
```

Arguments

x	A model fit object to be diagnosed.
type	The type of diagnostic plot. "histograms", the default, produces histograms of the sampled output statistic values with the observed statistics represented by vertical lines. "target" produces a pairs plot of the target output statistic values with the pairs of observed target statistics represented by red squares. output statistic values with the observed statistics represented by vertical lines. "model" produces a pairs plot of the sampled output statistic values with the pairs of observed statistics represented by red squares.
...	Additional parameters. Passed to geom_histogram if type="histogram" and pairs otherwise.

Details

Plots are produced that represent the distributions of the output sampled statistic values or the target statistics values. The values of the observed target statistics for the networks are also represented for comparison with the sampled statistics.

Examples

```
library(network)
set.seed(1)
data(flo)
flomarriage <- network(flo,directed=FALSE)
flomarriage %v% "wealth" <- c(10,36,27,146,55,44,20,8,42,103,48,49,10,48,32,3)


# An order dependent model
fit3 <- lolog(flomarriage ~ edges + nodeCov("wealth") + preferentialAttachment(),
              flomarriage ~ star(2:3), verbose=FALSE)
plot(fit3)
plot(fit3, "target")
plot(fit3, "model")
```

---

plot.Rcpp_DirectedNet *plot an DirectedNet object*

Description

plot an DirectedNet object

Usage

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

Arguments

x	the Rcpp_DirectedNet object
...	additional parameters for plot.network

Details

This is a thin wrapper around plot.network.

Examples

```
data(ukFaculty)
net <- as.BinaryNet(ukFaculty)
plot(net, vertex.col=net[["Group"]]+1)
```

plot.Rcpp_UndirectedNet *Plot an UndirectedNet object*

Description

Plot an UndirectedNet object

Usage

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

Arguments

x	the object
...	additional parameters for plot.network

Details

This is a thin wrapper around plot.network.

Examples

```
el <- matrix(c(1,2),ncol=2)
net <- new(UndirectedNet, el, 5L)
net[1,5] <- 1
net[2,5] <- 1
plot(net)
```

print.gofit *prints a gofit object*

Description

prints a gofit object

Usage

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

Arguments

x	The object
...	passed to print.data.frame

`print.lolog` *Print a 'lolog' object*

Description

Print a 'lolog' object

Usage

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

Arguments

`x`	the object
`...`	additional parameters (unused)

`print.logVariationalFit` *Print of a logVariationalFit object*

Description

Print of a logVariationalFit object

Usage

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

Arguments

`x`	the object
`...`	additional parameters (unused)

`registerDirectedStatistic` *Register Statistics*

Description

Register Statistics

Usage

```
registerDirectedStatistic
```

simulate.lolog *Generates BinaryNetworks from a fit lolog object*

Description

Generates BinaryNetworks from a fit lolog object

Usage

```
## S3 method for class 'lolog'
simulate(object, nsim = 1, seed = NULL, convert = FALSE, ...)
```

Arguments

object	A ‘lolog‘ object.
nsim	The number of simulated networks
seed	Either NULL or an integer that will be used in a call to set.seed before simulating
convert	convert to a network object#’
...	unused

Value

A list of BinaryNet (or network if convert=TRUE) objects. Networks contain an additional vertex covariate "__order__" that indicates the sequence order in which the vertex was ’added’ into the network.

Examples

```
library(network)
data(flo)
flomarriage <- network(flo,directed=FALSE)
flomarriage %v% "wealth" <- c(10,36,27,146,55,44,20,8,42,103,48,49,10,48,32,3)
fit <- lolog(flomarriage ~ edges + nodeCov("wealth"))
net <- simulate(fit)[[1]]
plot(net)
```

summary.lolog *Summary of a 'lolog' object*

Description

Summary of a 'lolog' object

Usage

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

Arguments

object	the object
...	additional parameters (unused)

Examples

```
data(lazega)
fit <- lologVariational(lazega ~ edges() + nodeMatch("office") + triangles,
                        nReplicates=50L, dyadInclusionRate=1)
summary(fit)
```

ukFaculty *Friendship network of a UK university faculty*

Description

The personal friendship network of a faculty of a UK university, consisting of 81 vertices (individuals) and 817 directed and weighted connections. The school affiliation of each individual is stored as a vertex attribute. The survey contained missing data for the school of two individuals.

Usage

```
data(ukFaculty)
```

Licenses and Citation

When publishing results obtained using this data set, the original authors (Nepusz T., Petroczi A., Negyessy L., Bazso F. 2008) should be cited, along with this R package.

Copyright

Source

The data set was originally reported by Nepusz et. al. (2008) and was subsequently processed and included by the igraphdata package. We have simply converted their network from an igraph to a network object.

References

Nepusz T., Petroczi A., Negyessy L., Bazso F.: Fuzzy communities and the concept of bridgeness in complex networks. Physical Review E 77:016107, 2008.

[*indexing*

Description

indexing

indexing

indexing

indexing

Usage

```
## S4 method for signature 'Rcpp_DirectedNet,ANY,ANY,ANY'
x[i, j, ..., maskMissing = TRUE, drop = TRUE]

## S4 method for signature 'Rcpp_UndirectedNet,ANY,ANY,ANY'
x[i, j, ..., maskMissing = TRUE, drop = TRUE]

## S4 replacement method for signature 'Rcpp_DirectedNet,ANY,ANY,ANY'
x[i, j, ...] <- value

## S4 replacement method for signature 'Rcpp_UndirectedNet,ANY,ANY,ANY'
x[i, j, ...] <- value
```

Arguments

x	object
i	indices
j	indices
...	unused
maskMissing	should missing values be masked by NA
drop	unused
value	values to assign

Examples

```
data(ukFaculty)
net <- as.BinaryNet(ukFaculty)


#dyad Extraction
net[1:2,1:5]
net$outNeighbors(c(1,2,3))

#dyad assignment
net[1,1:5] <- rep(NA,5)
net[1:2,1:5]
net[1:2,1:5,maskMissing=FALSE] #remove the mask over missing values and see
#nothing was really changed

#node variables
net$variableNames()
net[["Group"]]
net[["rnorm"]] <- rnorm(net$size())
net[["rnorm"]]
```

Index