Package ‘lolog’

October 13, 2022

Maintainer  Ian E. Fellows <ian@fellstat.com>
License      MIT + file LICENCE
Title        Latent Order Logistic Graph Models
LinkingTo    Rcpp, BH
Type         Package
LazyLoad     yes

Version      1.3
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  7.1.1
VignetteBuilder  knitr
NeedsCompilation yes
Author       Ian E. Fellows [aut, cre],
Mark S. Handcock [ctb]
Repository   CRAN
Date/Publication 2021-07-01 07:50:06 UTC

**R topics documented:**

<table>
<thead>
<tr>
<th>as.BinaryNet</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>as.BinaryNet.default</td>
<td>3</td>
</tr>
</tbody>
</table>
## as.BinaryNet

Convert to either an UndirectedNet or DirectedNet object

### Description

Convert to either an UndirectedNet or DirectedNet object

### Usage

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

Arguments

x the object

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

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

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

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

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

Arguments

- `x` the object
- `...` unused

Value

A network object

See Also

`DirectedNet`
Examples

e1 <- matrix(c(1,2),ncol=2)

# make an UndirectedNet with one edge and 5 nodes
net <- new(UndirectedNet, e1, 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
...

Value

A network object

See Also

UndirectedNet

Examples

e1 <- matrix(c(1,2),ncol=2)

# make an UndirectedNet with one edge and 5 nodes
net <- new(UndirectedNet, e1, 5L)
net[1:5,1:5]

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

```r
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.
createLatentOrderLikelihood

*Create a probability model for a latent ordered network model*

**Description**

Creates a probability model for a latent ordered network model

**Usage**

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

**Arguments**

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

**Value**

An Rcpp object representing the likelihood model

**Examples**

```r
# 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

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

```r
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)
```

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
```r
# 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]
```

```
lazega  
\textit{Collaboration Relationships Among Partners at a New England Law Firm}

\textbf{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.

\textbf{Usage}

data(lazega)

\textbf{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.

\textbf{Copyright}

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

\textbf{Source}

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

\textbf{References}


lolog  
\textit{Fits a LOLOG model via Monte Carlo Generalized Method of Moments}

\textbf{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,  # A lolog formula for the sufficient statistics (see details).
    auxFormula = NULL,  # A lolog formula of statistics to use for moment matching.
    theta = NULL,  # Initial parameters values. Estimated via lologVariational if NULL.
    nsamp = 1000,  # The number of sample networks to draw at each iteration.
    includeOrderIndependent = TRUE,  # If TRUE, all order independent terms in formula are used for moment matching.
    targetStats = NULL,  # 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 = "full",  # 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 = 0.1,  # The Hotelling's T^2 p-value tolerance for convergence for the transformed moment conditions.
    nHalfSteps = 10,  # The maximum number of half steps to take when the objective is not improved in an iteration.
    maxIter = 100,  # The maximum number of iterations.
    minIter = 2,  # The minimum number of iterations.
    startingStepSize = 0.1,  # The starting dampening of the parameter update.
    maxStepSize = 0.5,  # The largest allowed value for dampening.
    cluster = NULL,  # A parallel cluster to use for graph simulation.
    verbose = TRUE  # Level of verbosity 0-3.
)

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

$$\logit(p(y_{st} = 1 | \eta, y^{t-1}, s \leq t)) = \theta \cdot c(y_{st} = 1 | y^{t-1}, s \leq t)$$

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 $t - 1$. $c(y_{st} | y^{t-1}, s \leq t)$ is a vector representing the change in graph statistics from time $t - 1$ to $t$ if an edge is present, and $\theta$ 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 than 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"
lolog

formula
auxFormula
theta
stats
estats
obsStats
targetStats
obsModelStats
net
grad
cov
likelihoodModel

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)
## Description

LOLOG Model Terms

### NA

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

### Constraint Descriptions

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

## Description

Models

## Description

Create a skeleton for a package extending lolog

### Usage

```r
lologPackageSkeleton(path = ".")
```

### Arguments

- **path** where to create the package
lologVariational

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:

lologPackageSkeleton()

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
lologPackageSkeleton()
system("R CMD build LologExtension")
system("R CMD INSTALL LologExtension_1.0.tar.gz")

library(LologExtension) #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 <- lologVariational(g ~ edges() + minDegree(1L))
summary(fit)

## End(Not run)

lologVariational  Fits a latent ordered network model using Monte Carlo variational inference

Description

Fits a latent ordered network model using Monte Carlo variational inference
Usage

lologVariational(
  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


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

```r
## 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

```r
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")
```
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

```r
## 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. "model" produces a pairs plot of the sampled output statistic values with the pairs of 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

```r
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")
```
\section*{plot.Rcpp_DirectedNet}

\textit{plot an DirectedNet object}

\section*{Description}

plot an DirectedNet object

\section*{Usage}

\begin{verbatim}
## S3 method for class 'Rcpp_DirectedNet'
plot(x, ...)
\end{verbatim}

\section*{Arguments}

\begin{itemize}
  \item \textbf{x} \hspace{1cm} the Rcpp_DirectedNet object
  \item \textbf{...} \hspace{1cm} additional parameters for plot.network
\end{itemize}

\section*{Details}

This is a thin wrapper around \texttt{plot.network}.

\section*{Examples}

\begin{verbatim}
data(ukFaculty)
net <- as.BinaryNet(ukFaculty)
plot(net, vertex.col=net["Group"]+1)
\end{verbatim}

\section*{plot.Rcpp_UndirectedNet}

\textit{Plot an UndirectedNet object}

\section*{Description}

Plot an UndirectedNet object

\section*{Usage}

\begin{verbatim}
## S3 method for class 'Rcpp_UndirectedNet'
plot(x, ...)
\end{verbatim}

\section*{Arguments}

\begin{itemize}
  \item \textbf{x} \hspace{1cm} the object
  \item \textbf{...} \hspace{1cm} additional parameters for plot.network
\end{itemize}
Details

This is a thin wrapper around `plot.network`.

Examples

```r
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

```r
## 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

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

Arguments

- `x`: the object
- `...`: additional parameters (unused)
print.lologVariationalFit

Print of a lologVariationalFit object

Description

Print of a lologVariationalFit object

Usage

## S3 method for class 'lologVariationalFit'
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

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>object</code></td>
<td>A 'lolog' object.</td>
</tr>
<tr>
<td><code>nsim</code></td>
<td>The number of simulated networks</td>
</tr>
<tr>
<td><code>seed</code></td>
<td>Either NULL or an integer that will be used in a call to set.seed before simulating</td>
</tr>
<tr>
<td><code>convert</code></td>
<td>convert to a network object#</td>
</tr>
<tr>
<td>...</td>
<td>unused</td>
</tr>
</tbody>
</table>

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

```r
library(network)
data(flo)
flomarriage <- network(flo,directed=FALSE)
flomarriage %v% "wealth" <- c(10,36,27,146,56,44,20,8,42,103,48,49,10,48,32,3)
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

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>object</code></td>
<td>the object</td>
</tr>
<tr>
<td>...</td>
<td>additional parameters (unused)</td>
</tr>
</tbody>
</table>

Examples

```r
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**

Creative Commons Attribution-Share Alike 2.0 UK: England & Wales License, see [http://creativecommons.org/licenses/by-sa/2.0/uk/](http://creativecommons.org/licenses/by-sa/2.0/uk/) for details.

**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**

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

* datasets
  lazega, 12
  ukFaculty, 26
  [ . , Rcpp_DirectedNet, ANY, ANY, ANY-method ( ), 26
  [ . , Rcpp_DirectedNet-method ( ), 26
  [ . , Rcpp_UndirectedNet, ANY, ANY, ANY-method ( ), 26
  [ . , Rcpp_UndirectedNet-method ( ), 26
  [ <- ( ), 26
  [ <- , Rcpp_DirectedNet, ANY, ANY, ANY-method ( ), 26
  [ <- , Rcpp_DirectedNet-method ( ), 26
  [ <- , Rcpp_UndirectedNet, ANY, ANY, ANY-method ( ), 26
  [ <- , Rcpp_UndirectedNet-method ( ), 26
  _lolog_initStats (call-symbols), 6
  _rcpp_module_boot_lolog (call-symbols), 6
  as.BinaryNet, 2
  as.BinaryNet.default, 3
  as.network, 4
  as.network.Rcpp_DirectedNet, 4
  as.network.Rcpp_UndirectedNet, 5
  BinaryNet, 6
  calculateStatistics, 6
  call-symbols, 6
  coef.lolog, 7
  cppFunction, 11
  createCppModel, 7
  createLatentOrderLikelihood, 8
  DirectedLatentOrderLikelihood (LatentOrderLikelihood), 11
  DirectedModel (LologModels), 16
  DirectedNet, 6
  DirectedNet (BinaryNet), 6
  geom_histogram, 21
  glm, 18
  gofit, 9
  gofit.lolog, 9, 9
  initLologStatistics (call-symbols), 6
  inlineLologPlugin, 10, 17
  LatentOrderLikelihood, 11
  lazega, 12
  lolog, 6, 12
  lolog-terms, 16
  LologModels, 16
  lologPackageSkeleton, 16
  lologVariational, 13, 14, 17
  pairs, 21
  plot.gofit, 19
  plot.lologGmm, 21
  plot.network, 22, 23
  plot.Rcpp_DirectedNet, 22
  plot.Rcpp_UndirectedNet, 22
  print.gofit, 23
  print.lolog, 23
  print.lologVariationalFit, 24
  Rcpp_DirectedLatentOrderLikelihood-class (LatentOrderLikelihood), 11
  Rcpp_DirectedModel-class (LologModels), 16
  Rcpp_DirectedNet-class (BinaryNet), 6
  Rcpp_UndirectedLatentOrderLikelihood-class (LatentOrderLikelihood), 11
  Rcpp_UndirectedModel-class (LologModels), 16
  Rcpp_UndirectedNet-class (BinaryNet), 6
  registerDirectedOffset (registerDirectedStatistic), 24
  registerDirectedStatistic, 24
registerUndirectedOffset
  (registerDirectedStatistic), 24
registerUndirectedStatistic
  (registerDirectedStatistic), 24
runLologCppTests (call-symbols), 6

simulate.lolog, 24
sourceCpp, //
summary.lolog, 25

ukFaculty, 26
UndirectedLatentOrderLikelihood
  (LatentOrderLikelihood), 11
UndirectedModel (LologModels), 16
UndirectedNet, 5
UndirectedNet (BinaryNet), 6