Package: dnr (via r-universe)

August 21, 2024

Title Simulate Dynamic Networks using Exponential Random Graph Models (ERGM) Family

Version 0.3.5

Description Functions are provided to fit temporal lag models to dynamic networks. The models are build on top of exponential random graph models (ERGM) framework. There are functions for simulating or forecasting networks for future time points. Abhirup Mallik & Zack W. Almquist (2019) Stable Multiple Time Step Simulation/Prediction From Lagged Dynamic Network Regression Models, Journal of Computational and Graphical Statistics, 28:4, 967-979, <DOI:10.1080/10618600.2019.1594834>.

Depends R (>= 3.2.0), network, ergm

License GPL-3

LazyData yes

Imports sna, igraph, arm, glmnet

Suggests testthat, knitr

VignetteBuilder knitr

RoxygenNote 7.1.1

NeedsCompilation no

Author Abhirup Mallik [aut, cre], Zack Almquist [aut]

Maintainer Abhirup Mallik <abhirupkgp@gmail.com>

Date/Publication 2020-11-30 17:10:03 UTC

Repository https://abhirupkgp.r-universe.dev

RemoteUrl https://github.com/cran/dnr

RemoteRef HEAD

RemoteSha 91fcd30f8cd510118bfb6372f2553f9f84502edc

Contents

`beach` *Dynamically changing network of inter personal communication among the visitors of a beach in southern California.*

Description

A data set containing the dynamic network of inter personal interactions among the visitors of a beach in southern California.

Usage

```
beach
```

Format

A list with 31 elements, each element represent one observation in time. Each element is a network of varying size.

Source

Almquist, Z. W. and C. T. Butts (2014b). Logistic network regression for scalable analysis of networks with joint edge/ vertex dynamics. Sociological Methodology 44 (1), 1-33.

`binaryPlot` *binaryPlot*

Description

Plot for binary matrices, especially adjacency matrices.

Usage

```
binaryPlot(x, axlabs = TRUE, ...)
```

Arguments

`x`	matrix
`axlabs`	Binary, should the axis labels be shown.
`...`	title, xlabs, ylabs.

Details

binaryPlot

`clustCoef` *clustCoef*

Description

Calculates the cluster coefficient from a network

Usage

```
clustCoef(x)
```

Arguments

`x`	adjacency matrix

Details

Given a network in the form of adjacency matrix, this calculates the cluster coefficient. For a definition of cluster coefficient, please refer to the igraph documentation.

Value

scaler

Author(s)

Abhirup

Examples

```
clustCoef(beach[[1]][, ])
```

`engineEdge` *Implementation of simulation engine for dynamic networks using smoothing estimates of change statistics.*

Description

Implementation of simulation engine for dynamic networks using smoothing estimates of change statistics.

Usage

```
engineEdge(
  start_network,
  inputcoeff,
  ns,
  model.terms,
  model.formula,
  graph_mode,
  group,
  intercept,
  exvar,
  maxlag,
  lagmat,
  ylag,
  lambda = NA,
  method = "bayesglm",
  alpha.glmnet,
  paramout = TRUE
)
```

Arguments

`start_network`	Initial list of networks
`inputcoeff`	coefficient vector
`ns`	number of time points for simulation
`model.terms`	model terms in formula
`model.formula`	model formula (ergm)
`graph_mode`	'digraph' by default

group	group terms
intercept	intercept terms
exvar	extraneous covariates
maxlag	maximum lag
lagmat	lag matrix
ylag	lag vector for network lag terms
lambda	NA
method	'bayesglm' by default
alpha.glmnet	NA
paramout	T/F parameter estimation is returned.

Value

list: out_network: list of predicted networks coefmat: if paramout is TRUE, matrix of coefficients at all time.

Author(s)

Abhirup

Examples

```
## Not run:
input_network=rdNets[1:6];
model.terms=c("triadcensus.003", "triadcensus.012", "triadcensus.102", "triadcensus.021D", "gwesp");
model.formula = net~triadcensus(0:3)+gwesp(decay = 0, fixed=FALSE, cutoff=30)-1;
graph_mode='digraph';
group='dnc';
alpha.glmnet=1
directed=TRUE;
method <- 'bayesglm'
maxlag <- 3
lambda=NA
intercept = c("edges")
cdim <- length(model.terms)
lagmat <- matrix(sample(c(0,1),(maxlag+1)*cdim,replace = TRUE),ncol = cdim)
ylag <- rep(1,maxlag)
lagmat[1,] <- rep(0,ncol(lagmat))
out <- paramEdge(input_network,model.terms, model.formula,
                graph_mode="digraph",group,intercept = c("edges"),exvar=NA,
                maxlag = 3,
                lagmat = lagmat,
                ylag = rep(1,maxlag),
                lambda = NA, method='bayesglm',
                alpha.glmnet=1)
#
start_network <- input_network
inputcoeff <- out$coef$coef
nvertex <- 47
```

```
ns <- 10
exvar <- NA
tmp <- suppressWarnings(engineEdge(start_network=start_network,inputcoeff=inputcoeff,ns=ns,
                    model.terms=model.terms, model.formula=model.formula,
                    graph_mode=graph_mode,group=group,intercept=intercept,
                    exvar=exvar,
                    maxlag=maxlag,
                    lagmat=lagmat,
                    ylag=ylag,
                    lambda = NA, method='bayesglm',
                    alpha.glmnet=alpha.glmnet))
## End(Not run)
```

`engineEdgeBayes` *Implementation of simulation engine for dynamic networks using smoothing estimates of change statistics.*

Description

Implementation of simulation engine for dynamic networks using smoothing estimates of change statistics.

Usage

```
engineEdgeBayes(
  start_network,
  inputcoeff,
  ns,
  model.terms,
  model.formula,
  graph_mode,
  group,
  intercept,
  exvar,
  maxlag,
  lagmat,
  ylag,
  lambda = NA,
  method = "bayesglm",
  alpha.glmnet,
  paramout = TRUE,
  Theta = NA
)
```

Arguments

`start_network`	Initial list of networks

inputcoeff	coefficient vector
ns	number of time points for simulation
model.terms	model terms in formula
model.formula	model formula (ergm)
graph_mode	'digraph' by default
group	group terms
intercept	intercept terms
exvar	extraneous covariates
maxlag	maximum lag
lagmat	lag matrix
ylag	lag vector for network lag terms
lambda	NA
method	'bayesglm' by default
alpha.glmnet	NA
paramout	T/F parameter estimation is returned.
Theta	= prior probability matrix.

Examples

```
## Not run:
startNet <- rdNets[1:50]
model.terms=c("triadcensus.003", "triadcensus.012", "triadcensus.102", "triadcensus.021D", "gwesp")
model.formula = net~triadcensus(0:3)+gwesp(alpha=0, fixed=FALSE, cutoff=30)-1
graph_mode <- 'digraph'
group <- 'dnc'
alpha.glmnet <- 1
method <- 'bayesglm'
maxlag <- 3
lambda <- NA
intercept <- "edges"
cdim <- length(model.terms)
lagmat <- matrix(sample(c(0,1),(maxlag+1)*cdim,replace = TRUE),ncol = cdim)
ylag <- rep(1,maxlag)
lagmat[1,] <- rep(0,ncol(lagmat))

out.coef <- paramEdge(input_network = startNet,
                model.terms = model.terms,
                model.formula = model.formula,
                graph_mode='digraph',
                group=group,intercept = intercept,
                exvar=NA,
                maxlag = maxlag,
                lagmat = lagmat,
                ylag = ylag,
                lambda = NA, method='bayesglm',
                alpha.glmnet=1)
```

```
inputcoeff <- out.coef$coef$coef.edge
nvertex <- 47 ##find vertex here
ns <- 1
exvar <- NA
for(i in seq_along(startNet)) Theta <- Theta + startNet[[i]][,]
Theta <- Theta/length(startNet)
Theta <- thresh(Theta)
out.bayes <- engineEdgeBayes(start_network=startNet,
inputcoeff=inputcoeff,
ns=ns,
model.terms=model.terms,
model.formula=model.formula,
graph_mode=graph_mode,
group=group,intercept=intercept,
exvar=exvar,
maxlag=maxlag,
lagmat=lagmat,
ylag=ylag,
lambda = NA, method='bayesglm',
alpha.glmnet=alpha.glmnet,
Theta = Theta)

## End(Not run)
```

`engineEdgeNS` *Implementation of simulation engine for dynamic networks without using smoothing estimates of change statistics.*

Description

Implementation of simulation engine for dynamic networks without using smoothing estimates of change statistics.

Usage

```
engineEdgeNS(
  start_network,
  inputcoeff,
  ns,
  model.terms,
  model.formula,
  graph_mode,
  group,
  intercept,
  exvar,
  maxlag,
  lagmat,
```

```
  ylag,
  lambda = NA,
  method = "bayesglm",
  alpha.glmnet,
  paramout = TRUE
)
```

Arguments

start_network	Initial list of networks
inputcoeff	coefficient vector
ns	number of time points for simulation
model.terms	model terms in formula
model.formula	model formula (ergm)
graph_mode	'digraph' by default
group	group terms
intercept	intercept terms
exvar	extraneous covariates
maxlag	maximum lag
lagmat	lag matrix
ylag	lag vector for network lag terms
lambda	NA
method	'bayesglm' by default
alpha.glmnet	NA
paramout	T/F parameter estimation is returned.

Value

list: out_network: list of predicted networks coefmat: if paramout is TRUE, matrix of coefficients at all time.

Author(s)

Abhirup

Examples

```
## Not run:
input_network=rdNets[1:6];
model.terms=c("triadcensus.003", "triadcensus.012", "triadcensus.102", "triadcensus.021D", "gwesp");
model.formula = net~triadcensus(0:3)+gwesp(decay=0, fixed=FALSE, cutoff=30)-1;
graph_mode='digraph';
group='dnc';
alpha.glmnet=1
directed=TRUE;
```

```
method <- 'bayesglm'
maxlag <- 3
lambda=NA
intercept = c("edges")
cdim <- length(model.terms)
lagmat <- matrix(sample(c(0,1),(maxlag+1)*cdim,replace = TRUE),ncol = cdim)
ylag <- rep(1,maxlag)
lagmat[1,] <- rep(0,ncol(lagmat))
out <- paramEdge(input_network,model.terms, model.formula,
                graph_mode="digraph",group,intercept = c("edges"),exvar=NA,
                maxlag = 3,
                lagmat = lagmat,
                ylag = rep(1,maxlag),
                lambda = NA, method='bayesglm',
                alpha.glmnet=1)
#
start_network <- input_network
inputcoeff <- out$coef$coef
nvertex <- 47
ns <- 10
exvar <- NA
tmp <- suppressWarnings(engineEdgeNS(start_network=start_network,
                     inputcoeff=inputcoeff,ns=ns,
                     model.terms=model.terms, model.formula=model.formula,
                     graph_mode=graph_mode,group=group,intercept=intercept,
                     exvar=exvar,
                     maxlag=maxlag,
                     lagmat=lagmat,
                     ylag=ylag,
                     lambda = NA, method='bayesglm',
                     alpha.glmnet=alpha.glmnet))
## End(Not run)
```

`engineVertex` *Simulation Engine for dynamic Vertex case.*

Description

Simulation engine for dynamic networks with variable number of vertices. Implements exponential family based hierarchical model for vertices and the edges.

Usage

```
engineVertex(
  InputNetwork,
  numSim,
  maxLag,
  VertexStatsvec = rep(1, nvertexstats),
  VertexLag = rep(1, maxLag),
```

```
  VertexLagMatrix = matrix(1, maxLag, length(VertexStatsvec)),
  VertexModelGroup = NA,
  VertexAttLag = rep(1, maxLag),
  dayClassObserved = NA,
  dayClassFuture = NA,
  EdgeModelTerms,
  EdgeModelFormula,
  EdgeGroup = NA,
  EdgeIntercept = c("edges"),
  EdgeNetparam = NA,
  EdgeExvar = NA,
  EdgeLag = rep(1, maxLag),
  EdgeLagMatrix = matrix(1, maxLag, length(EdgeModelTerms)),
  regMethod = "bayesglm",
  paramout = TRUE
)
```

Arguments

InputNetwork	List of input networks
numSim	number of time points to simulate
maxLag	maximum Lag
VertexStatsvec	Binary vector for vertex model.
VertexLag	vector of lag for vertex
VertexLagMatrix	matrix of lags for vertex stats.
VertexModelGroup	Group term for vertex model.
VertexAttLag	Lag vector for group term for vertex.
dayClassObserved	Observed day class.
dayClassFuture	Dayclass vector for future, must be of size numsim.
EdgeModelTerms	Edge Model terms
EdgeModelFormula	Edge model formula
EdgeGroup	edge group term
EdgeIntercept	edge intercept
EdgeNetparam	edge network parameter name
EdgeExvar	edge extraneous variable
EdgeLag	edge Lag vector
EdgeLagMatrix	edge lag matrix
regMethod	regression method. "bayesglm" by default
paramout	T/F on if regression needs to run.

Value

List with following elements: SimNetwork: Output Networks EdgeParameterMat: Matrix of edge parameter VertexParameterMat: Matrix of Vertex parameters.

Author(s)

Abhirup

Examples

```
## Not run:
nvertexstats <- 9
maxLag = 3
VertexLag = rep(1, maxLag)
VertexLagMatrix <- matrix(0, maxLag, nvertexstats)
VertexLagMatrix[, c(4, 7)] <- 1
VertexLagMatrix[c(2,3),7] <- 0

getWeekend <- function(z){
    weekends <- c("Saturday", "Sunday")
    if(!network::is.network(z)){
        if(is.na(z)) return(NA)
    } else {
         zDay <- get.network.attribute(z, attrname = "day")
         out <- ifelse(zDay %in% weekends, 1, 0)
         return(out)
    }
}

dayClass <- numeric(length(beach))
for(i in seq_along(dayClass)) {
    dayClass[i] <- getWeekend(beach[[i]])
}
dayClass <- na.omit(dayClass)
simResult <- suppressWarnings(engineVertex(InputNetwork = beach,
                          numSim = 5,
                          maxLag = 3,
                          VertexStatsvec = rep(1, nvertexstats),
                          VertexModelGroup = "regular",
                          VertexAttLag = rep(1, maxLag),
                          VertexLag = rep(1, maxLag),
                          VertexLagMatrix = VertexLagMatrix,
                          dayClassObserved = dayClass,
                          dayClassFuture = c(1, 0, 0, 0, 0),
                          EdgeModelTerms = NA,
                          EdgeModelFormula = NA,
                          EdgeGroup = NA,
                          EdgeIntercept = c("edges"),
                          EdgeNetparam = c("logSize"),
                          EdgeExvar = NA,
                          EdgeLag = c(0, 1, 0),
                          paramout = TRUE
```

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

`engineVertexNS` *Simulation Engine for dynamic Vertex case without smoothing of estimated predictor matrices.*

Description

Simulation engine for dynamic networks with variable number of vertices. Implements exponential family based hierarchical model for vertices and the edges. This does not implement smoothing for estimated predictor matrices.

Usage

```
engineVertexNS(
  InputNetwork,
  numSim,
  maxLag,
  VertexStatsvec = rep(1, nvertexstats),
  VertexLag = rep(1, maxLag),
  VertexLagMatrix = matrix(1, maxLag, length(VertexStatsvec)),
  VertexModelGroup = NA,
  VertexAttLag = rep(1, maxLag),
  dayClassObserved = NA,
  dayClassFuture = NA,
  EdgeModelTerms,
  EdgeModelFormula,
  EdgeGroup = NA,
  EdgeIntercept = c("edges"),
  EdgeNetparam = NA,
  EdgeExvar = NA,
  EdgeLag = rep(1, maxLag),
  EdgeLagMatrix = matrix(1, maxLag, length(EdgeModelTerms)),
  regMethod = "bayesglm",
  paramout = TRUE
)
```

Arguments

`InputNetwork`	List of input networks
`numSim`	number of time points to simulate
`maxLag`	maximum Lag
`VertexStatsvec`	Binary vector for vertex model.
`VertexLag`	vector of lag for vertex

`VertexLagMatrix`
: matrix of lags for vertex stats.

`VertexModelGroup`
: Group term for vertex model.

`VertexAttLag`
: Lag vector for group term for vertex.

`dayClassObserved`
: Observed day class.

`dayClassFuture`
: Dayclass vector for future, must be of size numsim.

`EdgeModelTerms`
: Edge Model terms

`EdgeModelFormula`
: Edge model formula

`EdgeGroup`
: edge group term

`EdgeIntercept`
: edge intercept

`EdgeNetparam`
: edge network parameter name

`EdgeExvar`
: edge extraneous variable

`EdgeLag`
: edge Lag vector

`EdgeLagMatrix`
: edge lag matrix

`regMethod`
: regression method. "bayesglm" by default

`paramout`
: T/F on if regression needs to run.

Value

List with following elements: SimNetwork: Output Networks
EdgeParameterMat: Matrix of edge parameter
VertexParameterMat: Matrix of Vertex parameters.

Examples

```
## Not run:
nvertexstats <- 9
maxLag <- 3
VertexLag <- rep(1, maxLag)
VertexLagMatrix <- matrix(0, maxLag, nvertexstats)
VertexLagMatrix[, c(4, 7)] <- 1
VertexLagMatrix[c(2, 3), ] <- 1
simResult <- suppressWarnings(engineVertexNS(InputNetwork = beach,
                          numSim = 5,
                          maxLag = 3,
                          VertexStatsvec = rep(1, nvertexstats),
                          VertexModelGroup = "regular",
                          VertexAttLag = rep(1, maxLag),
                          VertexLag = rep(1, maxLag),
                          VertexLagMatrix = VertexLagMatrix,
                          EdgeModelTerms = NA,
                          EdgeModelFormula = NA,
                          EdgeGroup = NA,
```

```
                          EdgeIntercept = c("edges")
                          ))
## End(Not run)
```

---

expdeg *expdeg*

Description

Calculate the expectation of degree distribution of network

Usage

```
expdeg(x)
```

Arguments

x	adjacency matrix

Details

Given a network in adjacency matrix form, this calculates the expected degree statistic using igraph degree distribution function.

Value

scaler

Author(s)

Abhirup

Examples

```
expdeg(beach[[1]][, ])
```

ntriangles *ntriangles*

Description

Calculate number of triangles of a network

Usage

```
ntriangles(x)
```

Arguments

x	square matrix (adjacency matrix)

Details

This function calculates the number of triangles in a network given an adjacency matrix. We use igraph for this.

Value

scaler, number of triangles

Author(s)

Abhirup

Examples

```
ntriangles(beach[[1]][, ])
```

paramEdge *Parameter estimation for static vertex case.*

Description

Parameter estimation for the static vertex case.

Usage

```
paramEdge(
  input_network,
  model.terms,
  model.formula,
  graph_mode = "digraph",
  group,
  intercept = c("edges"),
  exvar = NA,
  maxlag = 3,
 lagmat = matrix(sample(c(0, 1), (maxlag + 1) * length(model.terms), replace = T),
    ncol = length(model.terms)),
  ylag = rep(1, maxlag),
  lambda = NA,
  method = "glmnet",
  alpha.glmnet = 1,
  paramout = TRUE
)
```

Arguments

input_network	Input network.
model.terms	model terms, must be ERGM terms expanded.
model.formula	ERGM formula for each time point.
graph_mode	'digraph' by default for bidirectional.
group	grouping covariates for vertices.
intercept	intercept terms.
exvar	Extraneous variables
maxlag	maximum lag.
lagmat	Matrix of dimension (maxlag+1)x(length(model.terms))
ylag	lag vectors of length=maxlag.
lambda	NA
method	Regression method, default is 'bayesglm'
alpha.glmnet	if regularization is used. not needed for bayesglm.
paramout	TRUE by default. if parameters are needed.

Value

list with elements: coef: coefficients mplematfull: full matrix of change statistics mplemat: subset of matrix of change statistics

Author(s)

Abhirup

Examples

```
## Not run:
input_network=rdNets[1:6]
model.terms=c("triadcensus.003", "triadcensus.012", "triadcensus.102", "triadcensus.021D", "gwesp");
model.formula = net~triadcensus(0:3)+gwesp(decay=0, fixed=FALSE, cutoff=30)-1;
graph_mode='digraph';
group='dnc';
alpha.glmnet=1
directed=TRUE;
method <- 'bayesglm'
maxlag <- 3
lambda=NA
intercept = c("edges")
cdim <- length(model.terms)
lagmat <- matrix(sample(c(0,1),(maxlag+1)*cdim,replace = TRUE),ncol = cdim)
ylag <- rep(1,maxlag)
exvar <- NA
out <- paramEdge(input_network,model.terms, model.formula,
                graph_mode='digraph',group,intercept = c("edges"),exvar=NA,
                maxlag = 3,
                lagmat = matrix(sample(c(0,1),(maxlag+1)*cdim,
                                       replace = TRUE),ncol = cdim),
                ylag = rep(1,maxlag),
                lambda = NA, method='bayesglm',
                alpha.glmnet=1)
## End(Not run)
```

---

paramVertex *Parameter estimation for Vertex dynamics*

Description

Parameter estimation fro dynamic vertex case. The interface remaining almost identical to the static vertex one.

Usage

```
paramVertex(
  InputNetwork,
  VertexStatsvec = rep(1, nvertexstats),
  maxLag,
  VertexLag = rep(1, maxLag),
  VertexLagMatrix = matrix(1, maxLag, length(VertexStatsvec)),
  VertexModelGroup = NA,
  VertexAttLag = rep(1, maxLag),
  dayClass = NA,
  EdgeModelTerms,
  EdgeModelFormula,
```

```
  EdgeGroup,
  EdgeIntercept = c("edges"),
  EdgeNetparam = NA,
  EdgeExvar = NA,
  EdgeLag = rep(1, maxLag),
  EdgeLagMatrix = matrix(1, maxLag, length(EdgeModelTerms)),
  regMethod = "bayesglm",
  paramout = FALSE
)
```

Arguments

InputNetwork	list of networks.
VertexStatsvec	binary vector of size 8.
maxLag	maximum lag, numeric.
VertexLag	binary vector of length maxLag.
VertexLagMatrix	binary matrix of size maxLag x 8.
VertexModelGroup	Grouping term for vertex model. Must be from vertex attribute list.
VertexAttLag	Lag vector for vertex group terms. Of length maxLag.
dayClass	Any network level present time attribute vector. Here used to indicate week/weekend as 0/1.
EdgeModelTerms	Model terms in edge model.
EdgeModelFormula	Model formula in edge model.
EdgeGroup	Group terms in edge model.
EdgeIntercept	Intercept for edge model.
EdgeNetparam	Network level parameter for edge model (currently only supported parameter is current network size).
EdgeExvar	Extraneous variable for edge model.
EdgeLag	binary vector of length maxLag.
EdgeLagMatrix	binary matrix of dim maxLag x length(EdgeModelTerms)
regMethod	Regression method. default: "bayesglm"
paramout	T/F Should the parameter estimates be returned?

Details

The Vertex model parameter list is as follows (Freeman degree, In degree, Out degree, Eigen Centrality, Between centrality, Info centrality, Closeness centrality, log k cycles, log size). For more details about the definitions of the terms, please refer to the vertexstats.R file, which implements all of these. The definitions are in sna or igraph.

Value

list with following elements:
EdgeCoef: edge coefficients.
Edgemplematfull: MPLE matrix from edges.
Edgemplemat: Subsetted MPLE matrix.
VertexCoef: Coefficients from vertex.
Vstats: Vertex statistics matrix.
EdgePredictor0: Edge predictors with imputations with 0.
EdgePredictor1: Edge predictors with imputations with 1.
EdgePredictorNA: Edge predictors with imputations with NA.
EdgeFit: Edge model.
VertexStatsFull: Vertex statistics matrix, full.
VertexFit: Vertex model.

Author(s)

Abhirup

Examples

```
nvertexstats <- 9
maxLag = 3
VertexLag = rep(1, maxLag)
VertexLagMatrix <- matrix(0, maxLag, nvertexstats)
VertexLagMatrix[, c(4, 7)] <- 1
VertexLagMatrix[c(2,3),7] <- 0

getWeekend <- function(z){
    weekends <- c("Saturday", "Sunday")
    if(!network::is.network(z)){
        if(is.na(z)) return(NA)
    } else {
         zDay <- get.network.attribute(z, attrname = "day")
         out <- ifelse(zDay %in% weekends, 1, 0)
         return(out)
    }
}

dayClass <- numeric(length(beach))
for(i in seq_along(dayClass)) {
    dayClass[i] <- getWeekend(beach[[i]])
}
dayClass <- na.omit(dayClass)


out <- paramVertex(InputNetwork = beach,
                   maxLag = 3,
                   VertexStatsvec = rep(1, nvertexstats),
                   VertexModelGroup = "regular",
                   VertexLag = rep(1, maxLag),
```

```
                    VertexLagMatrix = VertexLagMatrix,
                    dayClass = dayClass,
                    EdgeModelTerms = NA,
                    EdgeModelFormula = NA,
                    EdgeGroup = NA,
                    EdgeIntercept = c("edges"),
                    EdgeNetparam = c("logSize"),
                    EdgeExvar = NA,
                    EdgeLag = c(1, 1, 0),
                    paramout = TRUE)
```

`paramVertexOnly` *Parameter estimation for Vertex model only for a list of dynamic networks.*

Description

Parameter estimation for Vertex model only for a list of dynamic networks.

Usage

```
paramVertexOnly(
  InputNetwork,
  VertexStatsvec = rep(1, nvertexstats),
  maxLag,
  VertexLag = rep(1, maxLag),
  VertexLagMatrix = matrix(1, maxLag, length(VertexStatsvec)),
  dayClass = NA,
  regMethod = "bayesglm"
)
```

Arguments

`InputNetwork`	Input network list.
`VertexStatsvec`	Binary vector of size 9, indicating vertex model.
`maxLag`	maximum lag.
`VertexLag`	Binary vector of size maxLag, indicating Lag terms in the model.
`VertexLagMatrix`	Binary matrix indicating lagged vertex statistics in the model.
`dayClass`	Any network level present time attribute vector. Here used to indicate week/weekend as 0/1.
`regMethod`	one of "glm", "glmnet", "bayesglm"

Value

List of 3 elements:
VertexFit: Output from regEngine.
VertexStats: Subsetted vertex stats matrix.
VertexStatsFull: Full matrix of vertex stats.

Author(s)

Abhirup

Examples

```
nvertexstats <- 9
maxLag = 3
VertexLag = rep(1, maxLag)
VertexLagMatrix <- matrix(0, maxLag, nvertexstats)
VertexLagMatrix[, c(4, 7)] <- 1
VertexLagMatrix[c(2,3),7] <- 0
getWeekend <- function(z){
    weekends <- c("Saturday", "Sunday")
    if(!network::is.network(z)){
        if(is.na(z)) return(NA)
    } else {
         zDay <- get.network.attribute(z, attrname = "day")
         out <- ifelse(zDay %in% weekends, 1, 0)
         return(out)
    }
}

## for(i in 1:31) print(getWeekend(beach[[i]]))
## generate a vector of network level exogenous variable
dayClass <- numeric(length(beach))
for(i in seq_along(dayClass)) {
    dayClass[i] <- getWeekend(beach[[i]])
}
out <- paramVertexOnly(InputNetwork = beach,
                       maxLag = 3,
                       VertexStatsvec = rep(1, nvertexstats),
                       VertexLag = rep(1, maxLag),
                       VertexLagMatrix = VertexLagMatrix,
                       dayClass = dayClass)
```

---

paramVertexOnlyGroup *Parameter estimation for Vertex model only for a list of dynamic networks.*

Description

Parameter estimation for Vertex model only for a list of dynamic networks.

Usage

```
paramVertexOnlyGroup(
  InputNetwork,
  VertexStatsvec = rep(1, nvertexstats),
  maxLag,
```

```
  VertexModelGroup = NA,
  VertexLag = rep(1, maxLag),
  VertexAttLag = rep(1, maxLag),
  VertexLagMatrix = matrix(1, maxLag, length(VertexStatsvec)),
  regMethod = "bayesglm"
)
```

Arguments

InputNetwork	Input network list.
VertexStatsvec	Binary vector of size 9, indicating vertex model.
maxLag	maximum lag.
VertexModelGroup	Group term for vertex model.
VertexLag	Binary vector of size maxLag, indicating Lag terms in the model.
VertexAttLag	Vertex group term lag vector.
VertexLagMatrix	Binary matrix indicating lagged vertex statistics in the model.
regMethod	one of "glm", "glmnet", "bayesglm"

Value

List of 3 elements:
VertexFit: Output from regEngine.
VertexStats: Subsetted vertex stats matrix.
VertexStatsFull: Full matrix of vertex stats.

Author(s)

Abhirup

Examples

```
nvertexstats <- 9
InputNetwork <- beach
maxLag <- 3
VertexStatsvec <- rep(1, nvertexstats)
VertexLag <- rep(1, maxLag)
regMethod <- "bayesglm"
VertexModelGroup <- "regular"
VertexLagMatrix <- matrix(0, maxLag, nvertexstats)
VertexLagMatrix[, c(4, 7)] <- 1
VertexLagMatrix[c(2,3),7] <- 0
Vout1 <- paramVertexOnlyGroup(InputNetwork = beach,
                          maxLag = maxLag,
                          VertexStatsvec = VertexStatsvec,
                          VertexModelGroup = VertexModelGroup,
                          VertexLag = VertexLag,
                          VertexLagMatrix = VertexLagMatrix)
summary(Vout1$VertexFit$fit)
```

rdNets *Blog citation network*

Description

A data set of temporal inter and intra group blog citation network, with fixed number of vertex.

Usage

```
rdNets
```

Format

A list with 484 elements. Each element is a network of size 47 number of vertices.

Source

Butts, C. T. and B. R. Cross (2009). Change and external events in computer-mediated citation networks, English language weblogs and the 2004 u.s. electoral cycle. The Journal of Social Structure 10 (3), 1-29.

regEngine *General purpose regression engine for the methods bayesglm, glm and glmnet*

Description

General purpose regression engine for the methods bayesglm, glm and glmnet

Usage

```
regEngine(
  XYdata,
  method = "bayesglm",
  regIntercept = FALSE,
  lambda = NA,
  alpha = 1
)
```

Arguments

XYdata	matrix with X and Y columns. First column is named as y, other columns are X.
method	string among ("glm", "glmnet", "bayesglm").
regIntercept	Logical. Should intercept be included in the model?
lambda	for method "glmnet".
alpha	for "glmnet"

Value

list with elements: coef, se, lambda, fit (Coefficients, SE, lambda, if used, fit object.)

Author(s)

Abhirup

`vdegree` *vdegree*

Description

Calculates the degree of each vertices.

Usage

```
vdegree(x)
```

Arguments

`x`	Adjacency matrix.

Details

Given a network as adjacency matrix, calculate degree stats for each vertex.

Value

vector of length number of vertices.

Author(s)

Abhirup

Examples

```
vdegree(beach[[1]][, ])
```

Index