Package 'egoTERGM'

February 5, 2019

Date 2019-2-5

Type Package

Title Estimation of Ego-Temporal Exponential Random Graph Models via Expectation Maximization (EM)

Version 2.1.0

Author Benjamin W. Campbell [aut, cre],
Michael Salter-Townshend [ctb],
Thomas Brendan Murphy [ctb]

Maintainer Benjamin W. Campbell <campbell.1721@osu.edu>

URL http://github.com/benjamin-w-campbell/egoTERGM

Description Estimation of ego-temporal exponential random graph models with two-stage estimation including initialization through k-means clustering on temporal exponential random graph model parameters and EM as per Campbell (2018) <doi:10.7910/DVN/TWHEZ9>.

License GPL-3

Encoding UTF-8

LazyData true

RoxygenNote 6.1.1

Depends R (>= 3.4.0), parallel (>= 3.3.3), btergm (>= 1.9.0), xergm.common (>= 1.7.7)

Imports boot (>= 1.3-18), ergm (>= 3.7.1), network (>= 1.13.0), stats (>= 3.3.3), sna (>= 2.4), GGally (>= 1.3.2), Matrix (>= 1.2-8), speedglm (>= 0.3-2), methods (>= 3.3.3)

Suggests knitr, rmarkdown

VignetteBuilder knitr

NeedsCompilation no

Repository CRAN

Date/Publication 2019-02-05 21:55:57 UTC

R topics documented:

`ego_ergm` *Estimation of ego-Exponential Random Graph Model (ego-ERGM) using Expectation Maximization (EM) as per Salter-Townshend and Murphy (2015).*

Description

This function estimates an ego-ERGM. Code taken from Salter-Townshend and Murphy (2015)'s replication archive.

Usage

```
ego_ergm(net = NULL, form = NULL, core_size = 1, min_size = 5,
  roles = 3, directed = TRUE, edge_covariates = FALSE,
  seed = 12345, forking = FALSE, ncpus = 1, steps = 50,
  tol = 1e-06)
```

Arguments

`net`	The cross-sectional network that an ego-ERGM will be fit on. Must be presented as a network object. Any vertex attributes should be attached to networks. Currently the function does not support comparisons of whole networks.
`form`	The formula comprised of ERGM or TERGM terms used to distinguish between clusters assignments. Specified as a vector of comma separated terms. No default.
`core_size`	The order of alters to include. The default value of one implies only looking at an ego's alters and the connections among them.
`min_size`	The minimum number of nodes an ego-network must achieve to be included. Defaults to five.
`roles`	The number of roles that should be fit. Defaults to 3.
`directed`	Should the longitudinal network be treated as directed? If so, specify as the default TRUE.

edge_covariates
: Are edge covariates included in the form term? IF so, specify as TRUE. No default.

seed
: The seed set to replicate analysis for pseudorandom number generator.

forking
: If parallelization via forking should be used (TRUE) or if no parallel processing should be used (FALSE). Currently, sockets are not supported.

ncpus
: The number of CPUs that should should be used for estimation, defaults to 1.

steps
: The number of default EM steps that should be taken, defaults to 50.

tol
: The difference in parameter estimates between EM iterations to determine if the algorithm has converged. Defaults to 1e-6.

Value

A list of model results, including lambda (the probability of assignments), group.theta (the roles by terms cluster centroids), EE.BIC (the Salter-Townshend and Murphy BIC cross-sectional BIC), role_assignments (a data frame of the most likely assignments), and reduced_networks (network with excluded ego).

References

Box-Steffensmeier, Janet M., Benjamin W. Campbell, Dino P. Christenson, Zachary Navabi. (2018): Role analysis using the ego-ERGM: A Look at environmental interest group coalitions. *Social Networks* 52: 213-227. https://doi.org/10.1016/j.socnet.2017.08.004

Campbell, Benjamin W. (2018): Inferring Latent Roles in Longitudinal Networks. *Political Analysis* 26(3): 292-311. https://doi.org/10.1017/pan.2018.20

Salter-Townshend, Michael and Thomas Brendan Murphy. (2015): Role Analysis in Networks using Mixtures of Exponential Random Graph Models. *Journal of Computational and Graphical Statistics* 24(2): 520-538. https://doi.org/10.1080/10618600.2014.923777

Examples

```
# Code from xergm.common and their preparation of the Knecht network
library(xergm.common)
set.seed(1)

data("knecht")

for (i in 1:length(friendship)) {
 rownames(friendship[[i]]) <- paste("Student.", 1:nrow(friendship[[i]]), sep="")
 colnames(friendship[[i]]) <- paste("Student.", 1:nrow(friendship[[i]]), sep="")
}
rownames(primary) <- rownames(friendship[[1]])
colnames(primary) <- colnames(friendship[[1]])
sex <- demographics$sex
names(sex) <- rownames(friendship[[1]])
# step 2: imputation of NAs and removal of absent nodes:
friendship <- xergm.common::handleMissings(friendship, na = 10, method = "remove")
friendship <- xergm.common::handleMissings(friendship, na = NA, method = "fillmode")
```

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

# Reduce down to first time-step
ego_ergm_fit <- ego_ergm(net = net[[1]],
                         form = c("edges", "mutual", "triangle",
                                  "nodeicov('idegsqrt')", "nodeocov('odegsqrt')",
                                  "nodematch('sex')"),
                         core_size = 1,
                         min_size = 5,
                         roles = 3,
                         forking = FALSE,
                         ncpus = 1,
                         directed = TRUE,
                         edge_covariates = FALSE,
                         seed = 12345,
                         steps = 50,
                         tol = 1e-06)
```

`ego_tergm` *Estimation of ego-Temporal Exponential Random Graph Model (ego-TERGM) using Expectation Maximization (EM).*

Description

This function estimates an ego-TERGM on a longitudinally observed network. Currently the function does not support comparisons of whole networks.

Usage

```
ego_tergm(net = NULL, form = NULL, core_size = 1, min_size = 5,
  roles = 3, add_drop = TRUE, directed = TRUE,
  edge_covariates = FALSE, seed = 12345, R = 10, forking = FALSE,
  ncpus = 1, steps = 50, tol = 1e-06)
```

Arguments

net	The longitudinally observed network that an ego-TERGM will be fit on. Must be presented as a list of networks. Any vertex attributes should be attached to networks. Currently the function does not support comparisons of whole networks.
form	The formula comprised of ERGM or TERGM terms used to distinguish between clusters assignments. Specified as a vector of comma separated terms. No default.
core_size	The order of alters to include. The default value of one implies only looking at an ego's alters and the connections among them.
min_size	The minimum number of nodes an ego-network must achieve to be included. Defaults to five.
roles	The number of roles that should be fit. Defaults to 3.
add_drop	Do nodes drop out of the network or enter it? If so, specify as the default TRUE.
directed	Should the longitudinal network be treated as directed? If so, specify as the default TRUE.
edge_covariates	Are edge covariates included in the form term? IF so, specify as TRUE. No default. These should be stored as network attributes.
seed	The seed set to replicate analysis for pseudorandom number generator.
R	The number of bootstrap replications that should be used for the estimation of a bootstrapped MPLE estimated TERGM for model initialization. Defaults to 10.
forking	If parallelization via forking should be used (TRUE) or if no parallel processing should be used (FALSE). Currently, sockets are not supported.
ncpus	The number of CPUs that should should be used for estimation, defaults to 1.
steps	The number of default EM steps that should be taken, defaults to 50.
tol	The difference in parameter estimates between EM iterations to determine if the algorithm has converged. Defaults to 1e-6.

Value

A list of model results and input values, including net (original networks), lambda (the probability of assignments), group.theta (the roles by terms cluster centroids), EE.BIC (the Salter-Townshend and Murphy BIC cross-sectional BIC), TS.BIC (the Campbell BIC penalizing for time-steps), role_assignments (a data frame of the most likely assignments), reduced_networks (A list of the networks with excluded egos), ego_nets (a list of ego-networks), and ego_nets_used (N x T matrix of logicals here TRUE refers to ego-networks kept).

References

Campbell, Benjamin W. (2018): Inferring Latent Roles in Longitudinal Networks. *Political Analysis* 26(3): 292-311. https://doi.org/10.1017/pan.2018.20

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

Examples

```
# Code from xergm.common and their preparation of the Knecht network
library(xergm.common)
set.seed(1)

data("knecht")

for (i in 1:length(friendship)) {
 rownames(friendship[[i]]) <- paste("Student.", 1:nrow(friendship[[i]]), sep="")
 colnames(friendship[[i]]) <- paste("Student.", 1:nrow(friendship[[i]]), sep="")
}
rownames(primary) <- rownames(friendship[[1]])
colnames(primary) <- colnames(friendship[[1]])
sex <- demographics$sex
names(sex) <- rownames(friendship[[1]])
# step 2: imputation of NAs and removal of absent nodes:
friendship <- xergm.common::handleMissings(friendship, na = 10, method = "remove")
friendship <- xergm.common::handleMissings(friendship, na = NA, method = "fillmode")
# step 3: add nodal covariates to the networks
for (i in 1:length(friendship)) {
  s <- xergm.common::adjust(sex, friendship[[i]])
  friendship[[i]] <- network::network(friendship[[i]])
  friendship[[i]] <- network::set.vertex.attribute(friendship[[i]], "sex", s)
  idegsqrt <- sqrt(sna::degree(friendship[[i]], cmode = "indegree"))
  friendship[[i]] <- network::set.vertex.attribute(friendship[[i]],
                                                   "idegsqrt", idegsqrt)
  odegsqrt <- sqrt(sna::degree(friendship[[i]], cmode = "outdegree"))
  friendship[[i]] <- network::set.vertex.attribute(friendship[[i]],
                                                   "odegsqrt", odegsqrt)
}
sapply(friendship, network::network.size)
net <- friendship
rm(list=setdiff(ls(), "net"))

ego_tergm_fit <- ego_tergm(net = net,
                           form = c("edges", "mutual", "triangle",
                                    "nodeicov('idegsqrt')", "nodeocov('odegsqrt')",
                                    "nodematch('sex')"),
                           core_size = 1,
                           min_size = 5,
                           roles = 3,
                           add_drop = TRUE,
                           directed = TRUE,
                           edge_covariates = FALSE,
                           seed = 12345,
                           R = 10,
                           forking = FALSE,
                           ncpus = 1,
                           steps = 50,
                           tol = 1e-06)
```

ego_tergm_fit *Example Ego-TERGM Output*

Description

This is the output from a call to the ego-TERGM estimated according to the Knecht vignette.

Usage

```
data("ego_tergm_fit")
```

Format

Ego-TERGM example output, estimated from the Knecht vignette "network"

interpret_ego_tergm *Custom interpret function for ego-TERGM.*

Description

This function assists the user in interpreting output from the ego_tergm function.

Usage

```
interpret_ego_tergm(ego_tergm_fit = NULL, custom_var_names = NULL)
```

Arguments

ego_tergm_fit	The output from a fitted "ego_tergm".
custom_var_names	A vector of character terms in the same order as the form object fed to ego_tergm of clearer names for these variables.

References

Campbell, Benjamin W. (2018): Inferring Latent Roles in Longitudinal Networks. *Political Analysis* 26(3): 292-311. https://doi.org/10.1017/pan.2018.20

Examples

```
interpret_ego_tergm(ego_tergm_fit = ego_tergm_fit)
interpret_ego_tergm(ego_tergm_fit = ego_tergm_fit,
                    custom_var_names = c("Edges", "Mutual", "Triangle",
                                         "In-Degree", "Out-Degree", "Sex Homophily"))
```

`plot_ego_tergm` *Function to plot role assignments.*

Description

This function assists the user in interpreting output from the ego_tergm function.

Usage

```
plot_ego_tergm(ego_tergm_fit = NULL, plot_indices = NULL,
  node.size = 6, edge.size = 1, edge.color = "grey")
```

Arguments

`ego_tergm_fit`	The output from a fitted "ego_tergm".
`plot_indices`	A vector of indices reflecting the time steps that plots should be returned for. In networks observed over many time intervals it may be unrealistic to plot every network.
`node.size`	The size of the nodes fed to ggnet2.
`edge.size`	The size of the edges fed to ggnet2.
`edge.color`	The color of edges fed to ggnet2.

Value

A list of network plots with nodes colored by role assignment.

References

Campbell, Benjamin W. (2018): Inferring Latent Roles in Longitudinal Networks. *Political Analysis* 26(3): 292-311. https://doi.org/10.1017/pan.2018.20

Examples

```
plots <- plot_ego_tergm(ego_tergm_fit = ego_tergm_fit)
plots[[1]]
plots[[2]]
```

prepare_for_tergm *Prepares ego-TERGM output for xergm's btergm function.*

Description

This takes the output of an ego-TERGM call and prepares it for use by the xergm btergm function. Note: This routine assumes temporal independence within ego-networks and independence across ego-networks.

Usage

```
prepare_for_tergm(ego_tergm_fit = NULL)
```

Arguments

ego_tergm_fit	The output of an ego-TERGM call.

Value

A list of length G containing pooled cluster assignments. First-level elements of this list may be fed to a btergm call.

References

Campbell, Benjamin W. (2018): Inferring Latent Roles in Longitudinal Networks. *Political Analysis* 26(3): 292-311. https://doi.org/10.1017/pan.2018.20

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

Examples

```
# Code from xergm.common and their preparation of the Knecht network
library(xergm.common)
set.seed(1)

data("knecht")

for (i in 1:length(friendship)) {
 rownames(friendship[[i]]) <- paste("Student.", 1:nrow(friendship[[i]]), sep="")
 colnames(friendship[[i]]) <- paste("Student.", 1:nrow(friendship[[i]]), sep="")
}
rownames(primary) <- rownames(friendship[[1]])
colnames(primary) <- colnames(friendship[[1]])
sex <- demographics$sex
names(sex) <- rownames(friendship[[1]])
# step 2: imputation of NAs and removal of absent nodes:
friendship <- xergm.common::handleMissings(friendship, na = 10, method = "remove")
```

```
friendship <- xergm.common::handleMissings(friendship, na = NA, method = "fillmode")
# step 3: add nodal covariates to the networks
for (i in 1:length(friendship)) {
  s <- xergm.common::adjust(sex, friendship[[i]])
  friendship[[i]] <- network::network(friendship[[i]])
  friendship[[i]] <- network::set.vertex.attribute(friendship[[i]], "sex", s)
  idegsqrt <- sqrt(sna::degree(friendship[[i]], cmode = "indegree"))
  friendship[[i]] <- network::set.vertex.attribute(friendship[[i]],
                                                   "idegsqrt", idegsqrt)
  odegsqrt <- sqrt(sna::degree(friendship[[i]], cmode = "outdegree"))
  friendship[[i]] <- network::set.vertex.attribute(friendship[[i]],
                                                   "odegsqrt", odegsqrt)
}
sapply(friendship, network::network.size)
net <- friendship
rm(list=setdiff(ls(), "net"))

ego_tergm_fit <- ego_tergm(net = net,
                           form = c("edges", "mutual", "triangle",
                                    "nodeicov('idegsqrt')", "nodeocov('odegsqrt')",
                                    "nodematch('sex')"),
                           core_size = 1,
                           min_size = 5,
                           roles = 3,
                           add_drop = TRUE,
                           directed = TRUE,
                           edge_covariates = FALSE,
                           seed = 12345,
                           R = 10,
                           forking = FALSE,
                           ncpus = 1,
                           steps = 50,
                           tol = 1e-06)

net_list <- prepare_for_tergm(ego_tergm_fit)

role1_btergm <- btergm(net_list[[1]] ~ edges + mutual + triangle + nodeicov('idegsqrt') +
                                       nodeocov('odegsqrt') + nodematch('sex'),
                       R = 500)
```

`sim_egonets` *Simulation of longitudinal ego-networks according to a mixture of data generating processes and fitting of ego-TERGM to that network.*

Description

This function simulated longitudinal ego-networks and estimates an ego-TERGM. Useful for monte carlos and proofs of concept.

Usage

```
sim_egonets(form = NULL, params = NULL, roles = NULL,
  N_per_role = NULL, t_steps = NULL, egonet_size = NULL, R = 10,
  forking = FALSE, ncpus = 1, steps = 50, tol = 1e-06,
  seed = 12345)
```

Arguments

form	A vector of ERGM terms that should be used to generate the networks.
params	A "roles" by "form" matrix of ERGM simulation parameters with groups defined on the row and model terms defined on the column.
roles	An integer for the number of distinct mixture groups that should be simulated.
N_per_role	An integer for the number of different longitudinally observed ego-networks that should be simulated per role in roles.
t_steps	An integer for the number of time steps that each ego-network should be observed across.
egonet_size	An integer for the size of each ego-network simulated.
R	The number of bootstrap replications that should be used for the estimation of a bootstrapped MPLE estimated TERGM for model initialization. Defaults to 10.
forking	If parallelization via forking should be used (TRUE) or if no parallel processing should be used (FALSE). Currently, sockets are not supported.
ncpus	The number of CPUs that should should be used for estimation, defaults to 1.
steps	The number of default EM steps that should be taken, defaults to 50.
tol	The difference in parameter estimates between EM iterations to determine if the algorithm has converged. Defaults to 1e-6.
seed	The seed set to replicate analysis for pseudorandom number generator.

Value

A list of simulated ego-networks and the output of the ego_tergm function fit to this.

References

#' Campbell, Benjamin W. (2018): Inferring Latent Roles in Longitudinal Networks. *Political Analysis* 26(3): 292-311. https://doi.org/10.1017/pan.2018.20

Salter-Townshend, Michael and Thomas Brendan Murphy. (2015): Role Analysis in Networks using Mixtures of Exponential Random Graph Models. *Journal of Computational and Graphical Statistics* 24(2): 520-538. https://doi.org/10.1080/10618600.2014.923777

Examples

```
net <- sim_egonets(form = c("edges", "gwesp(0.8,fixed=TRUE)", "gwdegree(decay=0.8,fixed=TRUE)"),
                   params = rbind(c(-3,1,0), c(-1,-2,-1), c(-2,0,2)),
                   roles = 3,
```

```
N_per_role = 10,
t_steps = 3,
egonet_size = 20,
seed = 12345,
R = 30,
forking = FALSE,
ncpus = 1,
steps = 50,
tol = 1e-6)
```

`stability_validation` *Stability validation of ego-Temporal Exponential Random Graph Model (ego-TERGM) fit.*

Description

This function examines the stability of ego-TERGM results to the pooling window. It takes some proportion of the window's network and compares the result of a model fit on these time periods to the original fit.

Usage

```
stability_validation(ego_tergm_fit = NULL, splitting_probability = 0.5,
  seed = 12345, R = 10, forking = FALSE, ncpus = 1, steps = 50,
  tol = 1e-06)
```

Arguments

`ego_tergm_fit`	The output of a previously fit ego-TERGM fit using the ego_tergm function. This is the model that stability validation will be performed on.
`splitting_probability`	A value from 0 to 1 that determines the probability that any given network is assigned to the comparison group.
`seed`	The seed set to replicate analysis for pseudorandom number generator.
`R`	The number of bootstrap replications that should be used for the estimation of a bootstrapped MPLE estimated TERGM for model initialization. Defaults to 10.
`forking`	If parallelization via forking should be used (TRUE) or if no parallel processing should be used (FALSE). Currently, sockets are not supported.
`ncpus`	The number of CPUs that should should be used for estimation, defaults to 1.
`steps`	The number of default EM steps that should be taken, defaults to 50.
`tol`	The difference in parameter estimates between EM iterations to determine if the algorithm has converged. Defaults to 1e-6.

Value

Returns comparison_table (a matrix of cross-tabulation results to compare common cluster assignments or if incompatible a table of relative proportions sorted by value to allow for comparisons under set incompatibility and label switching), networks_sampled (which networks were included in the new validation sample), comparison_lambda (the matrix of role assignments for validation networks), comparison_group.theta (centroids for validation networks), comparison_EE.BIC (Salter-Townshend and Murphy BIC that doesn't penalize for longitudinal networks for validation networks), comparison_TS.BIC (BIC that penalizes for longitudinal networks for validation networks), comparison_role_assignments (role assignments for validation networks), and comparison_ego_nets (validation ego-networks). Note that labels may switch.

References

Campbell, Benjamin W. (2018): Inferring Latent Roles in Longitudinal Networks. *Political Analysis* 26(3): 292-311. https://doi.org/10.1017/pan.2018.20

Examples

```
# Code from xergm.common and their preparation of the Knecht network
library(xergm.common)
set.seed(1)

data("knecht")

for (i in 1:length(friendship)) {
 rownames(friendship[[i]]) <- paste("Student.", 1:nrow(friendship[[i]]), sep="")
 colnames(friendship[[i]]) <- paste("Student.", 1:nrow(friendship[[i]]), sep="")
}
rownames(primary) <- rownames(friendship[[1]])
colnames(primary) <- colnames(friendship[[1]])
sex <- demographics$sex
names(sex) <- rownames(friendship[[1]])
# step 2: imputation of NAs and removal of absent nodes:
friendship <- xergm.common::handleMissings(friendship, na = 10, method = "remove")
friendship <- xergm.common::handleMissings(friendship, na = NA, method = "fillmode")
# step 3: add nodal covariates to the networks
for (i in 1:length(friendship)) {
  s <- xergm.common::adjust(sex, friendship[[i]])
  friendship[[i]] <- network::network(friendship[[i]])
  friendship[[i]] <- network::set.vertex.attribute(friendship[[i]], "sex", s)
  idegsqrt <- sqrt(sna::degree(friendship[[i]], cmode = "indegree"))
  friendship[[i]] <- network::set.vertex.attribute(friendship[[i]],
                                                   "idegsqrt", idegsqrt)
  odegsqrt <- sqrt(sna::degree(friendship[[i]], cmode = "outdegree"))
  friendship[[i]] <- network::set.vertex.attribute(friendship[[i]],
                                                   "odegsqrt", odegsqrt)
}
sapply(friendship, network::network.size)
net <- friendship
rm(list=setdiff(ls(), "net"))
```

```
ego_tergm_fit <- ego_tergm(net = net,
                           form = c("edges", "mutual", "triangle",
                                    "nodeicov('idegsqrt')", "nodeocov('odegsqrt')",
                                    "nodematch('sex')"),
                           core_size = 1,
                           min_size = 5,
                           roles = 3,
                           add_drop = TRUE,
                           directed = TRUE,
                           edge_covariates = FALSE,
                           seed = 12345,
                           R = 10,
                           forking = FALSE,
                           ncpus = 1,
                           steps = 50,
                           tol = 1e-06)

stability_check <- stability_validation(ego_tergm_fit = ego_tergm_fit, seed = 614)
print(stability_check$comparison_table)
```

Index