Package 'missSBM'

September 16, 2019

Type Package

Title Handling Missing Data in Stochastic Block Models

Version 0.2.1

Maintainer Julien Chiquet <julien.chiquet@inra.fr>

Description
When a network is partially observed (here, NAs in the adjacency matrix rather than 1 or 0 due to missing information between node pairs), it is possible to account for the underlying process that generates those NAs. 'missSBM' adjusts the popular stochastic block model from network data sampled under various missing data conditions, as described in Tabouy, Barbillon and Chiquet (2019) <doi:10.1080/01621459.2018.1562934>.

URL https://jchiquet.github.io/missSBM

BugReports https://github.com/jchiquet/missSBM/issues

License GPL-3

Encoding UTF-8

LazyData true

RoxygenNote 6.1.1

Depends R (>= 3.4.0)

Imports Rcpp, methods, ape, igraph, nloptr, ggplot2, corrplot, R6, magrittr

LinkingTo Rcpp, RcppArmadillo

Collate 'RcppExports.R' 'SBM-Class.R' 'SBM_fit-Class.R' 'SBM_fit_covariates-Class.R' 'SBM_fit_nocovariate-Class.R' 'SBM_sampler-Class.R' 'er_network.R' 'estimate.R' 'frenchblog2007.R' 'missSBM-package.R' 'utils_missSBM.R' 'networkSampling-Class.R' 'networkSampling_fit-Class.R' 'missSBM_fit-Class.R' 'missSBM_collection-Class.R' 'networkSampler-Class.R' 'prepare_data.R' 'sample.R' 'sampledNetwork-Class.R' 'simulate.R' 'utils-pipe.R' 'utils_initialization.R' 'war.R'

Suggests aricode, blockmodels, testthat, covr, knitr, rmarkdown,

VignetteBuilder knitr

NeedsCompilation yes

Author Julien Chiquet [aut, cre] (<https://orcid.org/0000-0002-3629-3429>),
Pierre Barbillon [aut] (<https://orcid.org/0000-0002-7766-7693>),
Timothée Tabouy [aut]

Repository CRAN

Date/Publication 2019-09-16 10:10:02 UTC

R topics documented:

`dyadSampler` *Virtual class for all dyad-centered samplers*

Description

Virtual class for all dyad-centered samplers

Usage

```
dyadSampler
```

Format

An object of class `R6ClassGenerator` of length 24.

er_network *ER ego centred network*

Description

A dataset containing the weighted PPI network centred around the ESR1 (ER) protein

Usage

```
er_network
```

Format

A sparse symmetric matrix with 741 rows and 741 columns ESR1

Source

https://string-db.org/

Examples

```
data("er_network")
class(er_network)
```

estimate *Estimation of SBMs with missing data*

Description

Variational inference from sampled network data on a collection of Stochastic Block Models indexed by block number.

Usage

```
estimate(sampledNet, vBlocks, sampling, clusterInit = "hierarchical",
  useCovariates = TRUE, control = list())
```

Arguments

sampledNet	An object with class sampledNetwork, typically obtained with the function prepare_data (real-word data) or sample (simulation).
vBlocks	The vector of number of blocks considered in the collection
sampling	The sampling design for the modelling of missing data: MAR designs ("dyad", "node") and NMAR designs ("double-standard", "block-dyad", "block-node" ,"degree")

`clusterInit`	Initial method for clustering: either a character in "hierarchical", "spectral" or "kmeans", or a list with `length(vBlocks)` vectors, each with size `ncol(adjacencyMatrix)`, providing a user-defined clustering. Default is "hierarchical".
`useCovariates`	logicial. If covariates are present in sampledNet, should they be used for the inference or of the network sampling design, or just for the SBM inference? default is TRUE.
`control`	a list of parameters controlling the variational EM algorithm. See details.

Details

The list of parameters `control` essentially tunes the optimization process and the variational EM algorithm, with the following parameters

- "threshold"stop when an optimization step changes the objective function by less than threshold. Default is 1e-4.
- "maxIter"V-EM algorithm stops when the number of iteration exceeds maxIter. Default is 200
- "fixPointIter"number of fix-point iterations in the Variational E step. Default is 5.
- "cores"integer for number of cores used. Default is 1.
- "trace"integer for verbosity (0, 1, 2). Default is 1. Useless when `cores` > 1

The different sampling designs are split into two families in which we find dyad-centered and node-centered samplings. See <doi:10.1080/01621459.2018.1562934> for complete description.

- Missing at Random (MAR)
  - "dyad": parameter = p and
  $$p = P(Dyad(i,j)issampled)$$
  - "node": parameter = p and
  $$p = P(Nodeiissampled)$$
  - "covar-dyad": parameter = beta in R^M and
  $$P(Dyad(i,j)issampled) = logistic(parameter'covarArray(i,j,))$$
  - "covar-node": parameter = nu in R^M and
  $$P(Nodeiissampled) = logistic(parameter'covarMatrix(i,)$$
- Not Missing At Random (NMAR)
  - "double-standard": parameter = (p0,p1) and
  $$p0 = P(Dyad(i,j)issampled|thedyadisequalto0) =$$
  , p1 = P(Dyad (i,j) is sampled | the dyad is equal to 1)
  - "block-node": parameter = c(p(1),...,p(Q)) and
  $$p(q) = P(Nodeiissampled|nodeiisinclusterq)$$
  - "block-dyad": parameter = c(p(1,1),...,p(Q,Q)) and
  $$p(q,l) = P(Edge(i,j)issampled|nodeiisinclusterqandnodejisinclusterl)$$
  - "degree": parameter = c(a,b) and
  $$logit(a + b * Degree(i)) = P(Nodeiissampled|Degree(i))$$

Value

Returns an R6 object with class missSBM_collection.

See Also

sample, simulate, missSBM_collection and missSBM_fit.

Examples

```
## SBM parameters
directed <- FALSE
N <- 300 # number of nodes
Q <- 3   # number of clusters
alpha <- rep(1,Q)/Q     # mixture parameter
pi <- diag(.45,Q) + .05 # connectivity matrix

## simulate a SBM without covariates
sbm <- missSBM::simulate(N, alpha, pi, directed)

## Sample network data
samplingParameters <- .5 # the sampling rate
sampling <- "dyad"       # the sampling design
sampledNet <- missSBM::sample(sbm$adjacencyMatrix, sampling, samplingParameters)

## Inference :
vBlocks <- 1:5 # number of classes
collection <- missSBM::estimate(sampledNet, vBlocks, sampling)
collection$ICL
coef(collection$bestModel$fittedSBM, "connectivity")

myModel <- collection$bestModel
plot(myModel, "monitoring")
coef(myModel, "sampling")
coef(myModel, "connectivity")
head(predict(myModel))
head(fitted(myModel))
```

---

frenchblog2007 *Political Blogosphere network prior to 2007 French presidential election*

Description

French Political Blogosphere network dataset consists of a single day snapshot of over 200 political blogs automatically extracted the 14 October 2006 and manually classified by the "Observatoire Presidentielle" project. Originally part of the 'mixer' package

Usage

```
frenchblog2007
```

Format

An igraph object with 196 nodes. The vertex attribute "party" provides a possible clustering of the nodes.

Source

https://www.linkfluence.com/

Examples

```
data(frenchblog2007)
igraph::V(frenchblog2007)$party
igraph::plot.igraph(frenchblog2007,
  vertex.color = factor(igraph::V(frenchblog2007)$party),
  vertex.label = NA
 )
```

`missSBM` *Adjusting Stochastic Block Models under various missing data conditions*

Description

The missSBM package provides five functions:

- `simulate`a function to define and draw network data according to a stochastic block model
- `sample`a function to sample an existing network according to a variety of sampling designs
- `estimate`a function to perform inference of SBM from network data with missing entries under various sampling designs.
- `prepare_data`a function to format real-world network data (adjacency matrix and covariates) to perform the estimation under missing data condition
- `smooth`a function to smooth an existing collection of SBM estimation, to avoid being trapped in local maxima.

Details

These function leads to the manipulation of a variety of R object, with their respective fields and methods. They are all automatically generated by the top-level functions itemized above, so that the user should generally to use their constructor or internal methods directly. The user should only have a basic understanding of the fields of each object to manipulate the output in R. The main objects are the following:

- `sampledNetwork`an object to store sampled network data (i.e. with missing dyads)
- `SBM_sampler`an object to define a SBM to sample from
- `SBM_fit`an object to define and store an SBM fit
- `networkSampler`an object to define a network sampling to sample from

- networkSamplingan object to define and store a network sampling fit
- missSBM_fitan object that put together an SBM fit and and network sampling fit - the main point of the missSBM package !
- missSBM_collectionan object to store a collection of missSBM_fit, ordered by number of block

Author(s)

Timothée Tabouy, Pierre Barbillon, Julien Chiquet

References

Timothée Tabouy, Pierre Barbillon & Julien Chiquet (2019) "Variational Inference for Stochastic Block Models from Sampled Data", Journal of the American Statistical Association, <doi:10.1080/01621459.2018.1562934>

---

missSBM_collection *An object to represent a collection of missSBM_fit*

Description

This R6 class stores a collection of missSBM_fit. Comes with basic printing methods an field access.

Usage

```
missSBM_collection
```

Format

An object of class `R6ClassGenerator` of length 24.

Fields

`models` a list of models

`ICL` the vector of Integrated Classification Criterion (ICL) associated to the models in the collection (the smaller, the better)

`bestModel` the best model according to the ICL

`optimizationStatus` a data.frame summarizing the optimization process for all models

See Also

The function `estimate`, which produces an instance of this class. The function `smooth` can be used to smooth the ICL on a collection of model, as post-treatment.

missSBM_fit *R6 Class definition of an SBM-fit*

Description

This class is designed to adjust a Stochastic Block Model on a network with missing entries.

Usage

```
missSBM_fit
```

Format

An object of class `R6ClassGenerator` of length 24.

networkSampler *Definition of R6 Class 'networkSampling_sampler'*

Description

This class is use to define a sampling model for a network. Inherits from 'networkSampling'. Owns a rSampling method which takes an adjacency matrix as an input and send back an object with class sampledNetwork.

Usage

```
networkSampler
```

Format

An object of class `R6ClassGenerator` of length 24.

See Also

`sampledNetwork`

networkSampling *Definition of R6 Class 'networkSampling'*

Description

this virtual class is the mother of all subtypes of networkSampling (either sampler or fit) It is used to define a sampling model for a network. It has a rSampling method which takes an adjacency matrix as an input and send back an object with class sampledNetwork.

Usage

```
networkSampling
```

Format

An object of class `R6ClassGenerator` of length 24.

networkSamplingDyads_fit *Virtual class used to define a family of networkSamplingDyads_fit*

Description

Virtual class used to define a family of networkSamplingDyads_fit

Usage

```
networkSamplingDyads_fit
```

Format

An object of class `R6ClassGenerator` of length 24.

networkSamplingNodes_fit *Virtual class used to define a family of networkSamplingNodes_fit*

Description

Virtual class used to define a family of networkSamplingNodes_fit

Usage

```
networkSamplingNodes_fit
```

Format

An object of class `R6ClassGenerator` of length 24.

prepare_data *Prepare network data for estimation with missing data*

Description

This function puts together the adjacency matrix of a network and an optional list of covariates into a single `sampledNetwork` object, ready to use for inference with the `estimate` function of the missSBM package.

Usage

```
prepare_data(adjacencyMatrix, covariates = NULL,
  similarity = missSBM:::l1_similarity)
```

Arguments

`adjacencyMatrix`	The adjacency matrix of the network (NAs allowed)
`covariates`	An optional list with M entries (the M covariates). If the covariates are node-centred, each entry of `covariates` must be a size-N vector; if the covariates are dyad-centred, each entry of `covariates` must be N x N matrix.
`similarity`	An optional R x R -> R function to compute similarities between node covariates. Default is `l1_similarity`, that is, -abs(x-y). Only relevant when the `covariates` is a list of size-N vectors.

Value

Returns an R6 object with class `sampledNetwork`.

See Also

estimate and sampledNetwork.

Examples

```
data(war)
adj_beligerent <- war$beligerent %>% igraph::as_adj(sparse = FALSE)
sampledNet_war_nocov <- prepare_data(adj_beligerent)
sampledNet_war_withcov <- prepare_data(adj_beligerent, list(military_power = war$beligerent$power))
```

---

sample *Sampling of network data*

Description

This function samples observations in an adjacency matrix according to a given sampling design. The final results is an adjacency matrix with the dimension as the input, yet with additional NAs.

Usage

```
sample(adjacencyMatrix, sampling, parameters, clusters = NULL,
  covariates = NULL, similarity = l1_similarity, intercept = 0)
```

Arguments

adjacencyMatrix	The N x N adjacency matrix of the network to sample. If adjacencyMatrix is symmetric, we assume an undirected network with no loop; otherwise the network is assumed directed.
sampling	The sampling design used to sample the adjacency matrix, see details
parameters	The sampling parameters adapted to each sampling
clusters	An optional clustering membership vector of the nodes, only necessary for block samplings
covariates	A list with M entries (the M covariates). If the covariates are node-centred, each entry of covariates must be a size-N vector; if the covariates are dyad-centred, each entry of covariates must be N x N matrix.
similarity	An optional function to compute similarities between node covariates. Default is l1_similarity, that is, -abs(x-y). Only relevant when the covariates are node-centered (i.e. covariates is a list of size-N vectors).
intercept	An optional intercept term to be added in case of the presence of covariates. Default is 0.

Details

The different sampling designs are split into two families in which we find dyad-centered and node-centered samplings. See <doi:10.1080/01621459.2018.1562934> for complete description.

- Missing at Random (MAR)
  - "dyad": parameter = p
  $$p = P(Dyad(i,j) is sampled)$$
  - "node": parameter = p and
  $$p = P(Node i is sampled)$$
  - "covar-dyad": parameter = beta in R^M and
  $$P(Dyad(i,j) is sampled) = logistic(parameter' covarArray(i,j,))$$
  - "covar-node": parameter = nu in R^M and
  $$P(Node i is sampled) = logistic(parameter' covarMatrix(i,)$$
- Not Missing At Random (NMAR)
  - "double-standard": parameter = (p0,p1) and
  $$p0 = P(Dyad(i,j) is sampled | the dyad is equal to 0) =$$
  , p1 = P(Dyad (i,j) is sampled | the dyad is equal to 1)
  - "block-node": parameter = c(p(1),...,p(Q)) and
  $$p(q) = P(Node i is sampled | node i is in cluster q)$$
  - "block-dyad": parameter = c(p(1,1),...,p(Q,Q)) and
  $$p(q,l) = P(Edge(i,j) is sampled | node i is in cluster q and node j is in cluster l)$$
  - "degree": parameter = c(a,b) and
  $$logit(a + b * Degree(i)) = P(Node i is sampled | Degree(i))$$

Value

an object with class `sampledNetwork` containing all the useful information about the sampling. Can then feed the `estimate` function.

See Also

The class `sampledNetwork`

Examples

```
## SBM parameters
directed <- FALSE
N <- 300 # number of nodes
Q <- 3   # number of clusters
alpha <- rep(1,Q)/Q     # mixture parameter
pi <- diag(.45,Q) + .05 # connectivity matrix

## simulate a SBM without covariates
sbm <- missSBM::simulate(N, alpha, pi, directed)

## Sample network data

# some sampling design and their associated parameters
sampling_parameters <- list(
   "dyad" = .3,
   "node" = .3,
   "double-standard" = c(0.4, 0.8),
   "block-node" = c(.3, .8, .5),
   "block-dyad" = pi,
   "degree" = c(.01, .01)
 )

sampled_networks <- list()

for (sampling in names(sampling_parameters)) {
  sampled_networks[[sampling]] <-
     missSBM::sample(
       adjacencyMatrix = sbm$adjacencyMatrix,
       sampling        = sampling,
       parameters      = sampling_parameters[[sampling]],
       cluster         = sbm$memberships
     )
}

## SSOOOO long, but fancy
old_par <- par(mfrow = c(2,3))
for (sampling in names(sampling_parameters)) {
  plot(sampled_networks[[sampling]],
    clustering = sbm$memberships, main = paste(sampling, "sampling"))
}
par(old_par)
```

`sampledNetwork` *An R6 Class to represent sampled network data*

Description

The function `sample` and `prepare_data` produces an instance of an object with class `sampledNetwork`.

Usage

```
sampledNetwork
```

Format

An object of class `R6ClassGenerator` of length 24.

Details

All fields of this class are only accessible for reading. This class comes with a basic plot, summary and print methods

Fields

`samplingRate` percentage of observed dyads

`nNodes` number of nodes

`nDyads` number of dyads

`is_directed` direction

`adjacencyMatrix` adjacency matrix (with NA)

`covarMatrix` the matrix of covariates (if applicable)

`covarArray` the array of covariates (if applicable)

`dyads` list of potential dyads in the network

`missingDyads` array indices of missing dyads

`observedDyads` array indices of observed dyads

`samplingMatrix` matrix of observed and non-observed edges

`observedNodes` vector of observed and non-observed nodes

`NAs` boolean for NA entries in the adjacencyMatrix

Examples

```
## SBM parameters
directed <- FALSE
N <- 300 # number of nodes
Q <- 3   # number of clusters
alpha <- rep(1,Q)/Q     # mixture parameter
pi <- diag(.45,Q) + .05 # connectivity matrix

## simulate a SBM without covariates
sbm <- missSBM::simulate(N, alpha, pi, directed)

## Sample network data
sampled_network <-
     missSBM::sample(
       adjacencyMatrix = sbm$adjacencyMatrix,
       sampling        = "double-standard",
       parameters      = c(0.4, 0.8)
```

```
    )

summary(sampled_network)
print(sampled_network)
plot(sampled_network, clustering = sbm$memberships)
```

`SBM_fit` *R6 Class definition of an SBM-fit*

Description

This class is designed to adjust a Stochastic Block Model on a fully observed network. The doVEM method performs inference via Variational EM.

Usage

```
SBM_fit
```

Format

An object of class `R6ClassGenerator` of length 24.

Details

This class is virtual: inference is effective only for instance of one of the two child classes 'SBM_fit_nocovariate' and 'SBM_fit_covariates'

`SBM_sampler` *An R6 Class to represent a sampler for a SBM*

Description

The function `simulate` produces an instance of an object with class `SBM_sampler`.

Usage

```
SBM_sampler
```

Format

An object of class `R6ClassGenerator` of length 24.

Details

All fields of this class are only accessible for reading. This class comes with a set of methods, some of them being useful for the user (see examples)

- R6 methods:$rBlocks(), $rAdjancencyMatrix()
- S3 methodsprint(), plot()

Fields

`nNodes` The number of nodes

`nBlocks` The number of blocks

`nCovariates` The number of covariates

`nDyads` The number of possible dyad in the network (depends on the direction)

`direction` A character indicating if the network is directed or undirected

`hasCovariates` a boolean indicating if the model has covariates

`mixtureParam` the vector of mixture parameters

`connectParam` the matrix of connectivity: inter/intra probabilities of connection when the network does not have covariates, or a logit scaled version of it.

`covarParam` the vector of parameters associated with the covariates

`covarArray` the array of covariates

See Also

The function `simulate`.

Examples

```
## SBM parameters
directed <- FALSE
N <- 300 # number of nodes
Q <- 3   # number of clusters
alpha <- rep(1,Q)/Q     # mixture parameter
pi <- diag(.45,Q) + .05 # connectivity matrix
gamma <- log(pi/(1-pi)) # logit transform fo the model with covariates
M <- 2 # two Gaussian covariates
covarMatrix <- matrix(rnorm(N*M,mean = 0, sd = 1), N, M)
covarParam  <- rnorm(M, -1, 1)

## draw a SBM without covariates through simulateSBM
sbm <- missSBM::simulate(N, alpha, pi, directed)

## equivalent construction from the SBM_sampler class itslef
sbm_s <- SBM_sampler$new(directed, N, alpha, pi)
sbm_s$rBlocks() # draw some blocks
sbm_s$rAdjMatrix() # draw some edges

coef(sbm_s, "mixture")
coef(sbm_s, "connectivity")
summary(sbm_s)
```

simulate *Simulation of an SBM*

Description

Generates a realization (blocks and adjacency matrix) of a Stochastic Block model

Usage

```
simulate(nNodes, mixtureParam, connectParam, directed = FALSE,
  covariates = NULL, covarParam = NULL)
```

Arguments

nNodes	The number of nodes
mixtureParam	The mixture parameters
connectParam	The connectivity matrix (inter/intra clusters probabilities. provided on a logit scale for a model with covariates)
directed	Boolean variable to indicate whether the network is directed or not. Default to FALSE.
covariates	A list with M entries (the M covariates). Each entry of the list must be an N x N matrix.
covarParam	An optional vector of parameters associated with the covariates, with size M

Value

an object with class SBM_sampler

See Also

The class SBM_sampler

Examples

```
## SBM parameters
directed <- FALSE
N <- 300 # number of nodes
Q <- 3   # number of clusters
M <- 2 # two Gaussian covariates
alpha <- rep(1, Q)/Q     # mixture parameters
pi <- diag(.45, Q) + .05 # connectivity matrix
eta <- rnorm(M, -1, 1)  # covariate parametes
gamma <- log(pi/(1-pi)) # logit transform of pi for the model with covariates
X <- replicate(M, matrix(rnorm(N * N ,mean = 0, sd = 1), N, N), simplify = FALSE)

## draw a SBM without covariates
sbm <- missSBM::simulate(N, alpha, pi, directed)
```

```
coef(sbm, "connectivity")

## draw a SBM model with node-centred covariates
sbm_cov <- missSBM::simulate(N, alpha, gamma, directed, X, eta)
coef(sbm_cov, "covariates")

old_param <- par(mfrow = c(1,2))
plot(sbm)
plot(sbm_cov)
par(old_param)
```

`smooth` *Smooth the path ICL in a collection of missSBM_fit models*

Description

Apply a split and/or merge strategy of the clustering in a path of models in a collection of SBM ordered by number of block. The goal is to find better initialization. This results in a "smoothing" of the ICL, that should be close to concave.

Usage

```
smooth(Robject, type = c("forward", "backward", "both"),
  control = list())
```

Arguments

`Robject`	an object with class missSBM_collection, i.e. an output from `estimate`
`type`	character indicating what kind of ICL smoothing should be use among "forward", "backward" or "both". Default is "forward".
`control`	a list controlling the variational EM algorithm. See details.

Details

The list of parameters `control` controls the optimization process and the variational EM algorithm, with the following entries

- "iterates"integer for the number of iteration of smoothing. Default is 1.
- "threshold"stop when an optimization step changes the objective function by less than threshold. Default is 1e-4.
- "maxIter"V-EM algorithm stops when the number of iteration exceeds maxIter. Default is 200
- "fixPointIter"number of fix-point iteration for the Variational E step. Default is 5.
- "cores"integer for number of cores used. Default is 1.
- "trace"integer for verbosity. Useless when `cores` > 1

Value

an invisible missSBM_collection, in which the ICL has been smoothed

`war` *War data set*

Description

This dataset contains two networks where the nodes are countries and an edge in network "beligerent" means that the two countries have been at least once at war between years 1816 to 2007 while an edge in network "alliance" means that the two countries have had a formal alliance between years 1816 to 2012. The network 'beligerent' have less nodes since countries which have not been at war are not considered.

Usage

```
war
```

Format

A list with 2 two igraph objects, `alliance` and `beligerent`. Each graph have three attributes: 'name' (the country name), 'power' (a score related to military power: the higher, the better) and 'trade' (a score related to the trade effort between pairs of countries).

Source

networks were extracted from <http://www.correlatesofwar.org/>

References

Sarkees, Meredith Reid and Frank Wayman (2010). Resort to War: 1816 - 2007. Washington DC: CQ Press.

Gibler, Douglas M. 2009. International military alliances, 1648-2008. CQ Press

Examples

```
data(war)
class(war$beligerent)
igraph::gorder(war$alliance)
igraph::gorder(war$beligerent)
igraph::edges(war$alliance)
igraph::get.graph.attribute(war$alliance)
```

Index