Package ‘kmBlock’

July 24, 2025

Type Package

Title k-Means Like Blockmodeling of One-Mode and Linked Networks

Version 0.1.4

Date 2025-07-23

Maintainer Aleš Žiberna <ales.ziberna@fdv.uni-lj.si>

Description Implements k-means like blockmodeling of one-mode and linked networks as presented in Žiberna (2020) <doi:10.1016/j.socnet.2019.10.006>. The development of this package is financially supported by the Slovenian Research Agency (<https://www.arrs.si/>) within the research programs P5-0168 and the research projects J7-8279 (Blockmodeling multilevel and temporal networks) and J5-2557 (Comparison and evaluation of different approaches to blockmodeling dynamic networks by simulations with application to Slovenian co-authorship networks).

License GPL (>= 2)

Encoding UTF-8

Imports blockmodeling, doParallel, doRNG, foreach, Rcpp (>= 1.0.0)

LinkingTo Rcpp, RcppArmadillo

Author Aleš Žiberna [aut, cre]

NeedsCompilation yes

RoxygenNote 7.2.3

Repository CRAN

Date/Publication 2025-07-24 14:40:10 UTC

Contents

`critFunKmeans`	*Function that computes criterion function used in k-means like one-mode blockmodeling. If `clu` is a list, the method for linked/multilevel networks is applied*

Description

Function that computes criterion function used in k-means like one-mode blockmodeling. If `clu` is a list, the method for linked/multilevel networks is applied

Usage

```
critFunKmeans(
  M,
  clu,
  weights = NULL,
  diagonal = c("ignore", "seperate", "same"),
  limitType = c("none", "inside", "outside"),
  limits = NULL
)
```

Arguments

`M`	A matrix representing the (usually valued) network. For multi-relational networks, this should be an array with the third dimension representing the relation.
`clu`	A partition. Each unique value represents one cluster. If the network is one-mode, than this should be a vector, else a list of vectors, one for each mode. Similarly, if units are comprised of several sets, clu should be the list containing one vector for each set.
`weights`	The weights for each cell in the matrix/array. A matrix or an array with the same dimensions as `M`.
`diagonal`	How should the diagonal values be treated. Possible values are: • ignore - diagonal values are ignored • seperate - diagonal values are treated separately • same - diagonal values are treated the same as all other values
`limitType`	What do the limits represent, on which "side" of this limits should the values lie. Possible values: "none","inside","outside"
`limits`	If `diagonal` is `"ignore"` or `"same"`, an array with dimensions equal to: • number of clusters (of all types) • number of clusters (of all types) • number of relations • 2 - the first is lower limit and the second is upper limit If `diagonal` is `"seperate"`, a list of two array. The first should be as described above, representing limits for off diagonal values. The second should be similar with only 3 dimensions, as one of the first two must be omitted.

Value

A list similar to optParC in package `blockmodeling`.

`kmBlockC`	*Function that performs k-means like one-mode blockmodeling. If* `clu` *is a list, the method for linked/multilevel networks is applied*

Description

Function that performs k-means like one-mode blockmodeling. If `clu` is a list, the method for linked/multilevel networks is applied

Usage

```
kmBlockC(
  M,
  clu,
  weights = NULL,
  diagonal = c("ignore", "seperate", "same"),
  limitType = c("none", "inside", "outside"),
  limits = NULL
)
```

Arguments

`M` A matrix representing the (usually valued) network. For multi-relational networks, this should be an array with the third dimension representing the relation.

`clu` A partition. Each unique value represents one cluster. If the network is one-mode, than this should be a vector, else a list of vectors, one for each mode. Similarly, if units are comprised of several sets, clu should be the list containing one vector for each set.

`weights` The weights for each cell in the matrix/array. A matrix or an array with the same dimensions as `M`.

`diagonal` How should the diagonal values be treated. Possible values are:

- ignore - diagonal values are ignored
- seperate - diagonal values are treated separately
- same - diagonal values are treated the same as all other values

`limitType` What do the limits represent, on which "side" of this limits should the values lie. Possible values: "none","inside","outside"

`limits` If `diagonal` is `"ignore"` or `"same"`, an array with dimensions equal to:

- number of clusters (of all types)
- number of clusters (of all types)
- number of relations
- 2 - the first is lower limit and the second is upper limit

If `diagonal` is `"seperate"`, a list of two array. The first should be as described above, representing limits for off diagonal values. The second should be similar with only 3 dimensions, as one of the first two must be omitted.

Value

A list similar to optParC in package `blockmodeling`.

See Also

`kmBlockORPC`

`kmBlockORPC` *A function for optimizing multiple random partitions using k-means one-mode and linked blockmodeling. Calls* `kmBlockC` *for optimizing individual random partitions.*

Description

A function for optimizing multiple random partitions using k-means one-mode and linked blockmodeling. Calls `kmBlockC` for optimizing individual random partitions.

Usage

```
kmBlockORPC(
  M,
  k,
  rep,
  save.initial.param = TRUE,
  deleteMs = TRUE,
  max.iden = 10,
  return.all = FALSE,
  return.err = TRUE,
  seed = NULL,
  parGenFun = blockmodeling::genRandomPar,
  mingr = NULL,
  maxgr = NULL,
  addParam = list(genPajekPar = TRUE, probGenMech = NULL),
  maxTriesToFindNewPar = rep * 10,
  skip.par = NULL,
  printRep = ifelse(rep <= 10, 1, round(rep/10)),
  n = NULL,
  nCores = 1,
  useParLapply = TRUE,
  cl = NULL,
  stopcl = is.null(cl),
  ...
)
```

Arguments

`M`	A square matrix giving the adjaciency relationg between the network's nodes (aka vertexes)
`k`	The number of clusters used in the generation of partitions.
`rep`	The number of repetitions/different starting partitions to check.
`save.initial.param`	Should the inital parameters(`approaches`, ...) of using `kmBlockC` be saved. The default value is `TRUE`.
`deleteMs`	Delete networks/matrices from the results of to save space. Defaults to `TRUE`.
`max.iden`	Maximum number of results that should be saved (in case there are more than `max.iden` results with minimal error, only the first `max.iden` will be saved).
`return.all`	If `FALSE`, solution for only the best (one or more) partition/s is/are returned.
`return.err`	Should the error for each optimized partition be returned. Defaults to `TRUE`.
`seed`	Optional. The seed for random generation of partitions.
`parGenFun`	The function (object) that will generate random partitions. The default function is `genRandomPar`. The function has to accept the following parameters: `k` (number o of partitions by modes, `n` (number of units by modes), `seed` (seed value for random generation of partition), `addParam` (a list of additional parameters).
`mingr`	Minimal allowed group size.
`maxgr`	Maximal allowed group size.
`addParam`	A list of additional parameters for function specified above. In the usage section they are specified for the default function `genRandomPar`.
`maxTriesToFindNewPar`	The maximum number of partition try when trying to find a new partition to optimize that was not yet checked before - the default value is `rep * 1000`.
`skip.par`	The partitions that are not allowed or were already checked and should therefore be skipped.
`printRep`	Should some information about each optimization be printed.
`n`	The number of units by "modes". It is used only for generating random partitions. It has to be set only if there are more than two modes or if there are two modes, but the matrix representing the network is one mode (both modes are in rows and columns).
`nCores`	Number of cores to be used. Value `0` means all available cores. It can also be a cluster object.
`useParLapply`	Should `parLapplyLB` be used (otherwise `foreach` is used). Defaults to true as it needs less dependencies. It might be removed in future releases and only allow the use of parLapplyLB.
`cl`	The cluster to use (if formed beforehand). Defaults to `NULL`.
`stopcl`	Should the cluster be stopped after the function finishes. Defaults to `is.null(cl)`.
`...`	Arguments passed to other functions, see `kmBlockC`.

Value

A list of class "opt.more.par" containing:

`M`	The one- or multi-mode matrix of the network analyzed
`res`	If `return.all` = `TRUE` - A list of results the same as `best` - one `best` for each partition optimized.
`best`	A list of results from `kmBlockC`, only without `M`.
`err`	If `return.err` = `TRUE` - The vector of errors or inconsistencies = -log-likelihoods.
`ICL`	Integrated classification likelihood for the best partition.
`checked.par`	If selected - A list of checked partitions. If `merge.save.skip.par` is `TRUE`, this list also includes the partitions in `skip.par`.
`call`	The call to this function.
`initial.param`	If selected - The initial parameters are used.
`Random.seed`	.Random.seed at the end of the function.
`cl`	Cluster used for parallel computations if supplied as an input parameter.

Warning

It should be noted that the time needed to optimize the partition depends on the number of units (aka nodes) in the networks as well as the number of clusters due to the underlying algorithm. Hence, partitioning networks with several hundred units and large number of blocks (e.g., >5) can take a long time (from 20 minutes to a few hours or even days).

Author(s)

Aleš, Žiberna

References

Žiberna, Aleš (2020). k-means-based algorithm for blockmodeling linked networks. Social Networks 32(1), 105-126, doi:10.1016/j.socnet.2019.10.006.

See Also

`kmBlockC`

Examples

```
# Simple one-mode network
library(blockmodeling)
k<-2
blockSizes<-rep(20,k)
IM<-matrix(c(0.8,.4,0.2,0.8), nrow=2)
if(any(dim(IM)!=c(k,k))) stop("invalid dimensions")

set.seed(2021)
clu<-rep(1:k, times=blockSizes)
```

```
n<-length(clu)
M<-matrix(rbinom(n*n,1,IM[clu,clu]),ncol=n, nrow=n)
diag(M)<-0
plotMat(M)

resORP<-kmBlockORPC(M,k=2, rep=10, return.all = TRUE)
plot(resORP)
clu(resORP)


# Linked network
library(blockmodeling)
set.seed(2021)
IM<-matrix(c(0.8,.4,0.2,0.8), nrow=2)
clu<-rep(1:2, each=20)
n<-length(clu)
nClu<-length(unique(clu))
M1<-matrix(rbinom(n^2,1,IM[clu,clu]),ncol=n, nrow=n)
M2<-matrix(rbinom(n^2,1,IM[clu,clu]),ncol=n, nrow=n)
M12<-diag(n)
nn<-c(n,n)
k<-c(2,2)
Ml<-matrix(0, nrow=sum(nn),ncol=sum(nn))
Ml[1:n,1:n]<-M1
Ml[n+1:n,n+1:n]<-M2
Ml[n+1:n, 1:n]<-M12
plotMat(Ml)

resMl<-kmBlockORPC(M=Ml, k=k, n=nn, rep=10)
plot(resMl)
clu(resMl)
```

Index