

Package: INCVCommunityDetection (via r-universe)

May 12, 2026

Type Package

Title Inductive Node-Splitting Cross-Validation for Community Detection

Version 0.1.0

Description Implements Inductive Node-Splitting Cross-Validation (INCV) for selecting the number of communities in stochastic block models. Provides f-fold and random-split node-level cross-validation, along with competing methods including CROISSANT, Edge Cross-Validation (ECV), and Node Cross-Validation (NCV). Supports both SBM and Degree-Corrected Block Models (DCBM), with multiple loss functions (L2, binomial deviance, AUC). Includes network simulation utilities for SBM, RDPG, and latent space models.

URL <https://github.com/ivylinzhang97/incv-community-detection>

BugReports <https://github.com/ivylinzhang97/incv-community-detection/issues>

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.3

Depends R (>= 3.5.0)

Imports methods, Matrix, RSpectra, ClusterR, irlba, parallel, cluster, Rfast, data.table, IMIFA, stats

Suggests latentnet, network, rdist, igraph, testthat (>= 3.0.0), knitr, rmarkdown

VignetteBuilder knitr

Config/testthat/edition 3

Config/pak/sysreqs libgmp3-dev make

Repository <https://ivylinzhang97.r-universe.dev>

Date/Publication 2026-03-08 04:37:45 UTC

RemoteUrl <https://github.com/ivylinzhang97/incv-community-detection>

RemoteRef HEAD

RemoteSha 829a0cf1002e944c7a68326c42c884c064c1c95a

Contents

AUC	2
best.perm.label.match	3
bin.dev	3
blockmodel.gen.fast	4
community.sim	5
community.sim.sbm	5
croissant.blockmodel	6
croissant.latent	7
croissant.rdpd	8
croissant.tune.regsp	9
ECV.for.blockmodel	10
ECV.undirected.Rank	11
edge.index.map	12
eigen.DCBM.est	12
fast.DCBM.est	13
fast.SBM.est	13
l2	14
latent.gen	14
matched.lab	15
NCV.for.blockmodel	16
neglog	16
nscv.f.fold	17
nscv.random.split	18
SBM.prob	19
SBM.spectral.clustering	20
sparse.RDPG.gen	20
Index	22

AUC

Negative AUC for matrix predictions

Description

Negative AUC for matrix predictions

Usage

AUC(A, P)

Arguments

- A Observed binary adjacency (matrix or vector).
- P Predicted probability (matrix or vector).

Value

Negative AUC value (for minimisation-based model selection).

`best.perm.label.match` *Find the best permutation label matching*

Description

Matches community labels in `lab` to reference labels in `fixed` via a greedy maximum-overlap permutation.

Usage

```
best.perm.label.match(lab, fixed, n = length(lab), K = max(lab, fixed))
```

Arguments

- `lab` Integer vector of labels to permute.
- `fixed` Integer vector of reference labels.
- `n` Number of nodes (default `length(lab)`).
- `K` Number of communities (default `max(lab, fixed)`).

Value

A $K \times K$ permutation matrix.

`bin.dev` *Binomial deviance loss*

Description

Binomial deviance loss

Usage

```
bin.dev(x, y)
```

Arguments

x	Observed binary matrix/vector.
y	Predicted probability matrix/vector.

Value

Total binomial deviance (non-finite terms set to 0).

`blockmodel.gen.fast` *Generate a fast SBM or DCBM network (sparse)*

Description

Uses parallel edge sampling for speed; returns a sparse adjacency matrix.

Usage

```
blockmodel.gen.fast(n, K, B, psi = rep(1, n), PI = rep(1/K, K), ncore = 1)
```

Arguments

n	Number of nodes.
K	Number of communities.
B	K x K block probability matrix.
psi	Degree heterogeneity vector (length n); set to all 1s for SBM.
PI	Community prior probabilities (length K).
ncore	Number of cores for parallel computation.

Value

A list with:

A	Sparse symmetric adjacency matrix.
member	Community membership vector.
psi	Scaled degree heterogeneity vector.

community.sim	<i>Simulate a Stochastic Block Model network</i>
---------------	--

Description

Generates an adjacency matrix from a planted-partition SBM with k communities. The first community has size n_1 ; the remaining communities share the leftover nodes roughly equally.

Usage

```
community.sim(k = 2, n = 1000, n1 = 100, p = 0.3, q = 0.1)
```

Arguments

k	Number of communities (default 2).
n	Total number of nodes (default 1000).
n_1	Size of the smallest community (default 100).
p	Within-community connection probability (default 0.3).
q	Between-community connection probability (default 0.1).

Value

A list with:

membership	Integer vector of community labels (length n).
adjacency	$n \times n$ binary symmetric adjacency matrix.

Examples

```
net <- community.sim(k = 3, n = 120, n1 = 30, p = 0.5, q = 0.1)
table(net$membership)
```

community.sim.sbm	<i>Simulate an SBM with distance-decaying block probabilities</i>
-------------------	---

Description

Generates an SBM where block probabilities decay exponentially with the distance between community indices: $B[k_1, k_2] = \rho * \eta^{\min(|k_1 - k_2|, 3)}$.

Usage

```
community.sim.sbm(n, n1, eta = 0.3, rho = 0.1, K = 3)
```

Arguments

n	Total number of nodes.
n1	Size of the first (smallest) community.
eta	Decay parameter for inter-community probabilities (default 0.3).
rho	Scaling factor for the block probability matrix (default 0.1).
K	Number of communities (default 3).

Value

A list with:

adjacency	n x n binary symmetric adjacency matrix.
membership	Integer vector of community labels.
conn	K x K block probability matrix.

croissant.blockmodel *CROISSANT for blockmodel selection*

Description

Cross-validated, Overlapping, In-Sample Selection of the Number of communities and model Type. Jointly selects between SBM and DCBM and the number of communities K using overlapping node subsamples.

Usage

```
croissant.blockmodel(
  A,
  K.CAND,
  s,
  o,
  R,
  tau = 0,
  laplace = FALSE,
  dc.est = 2,
  loss = c("l2", "bin.dev", "AUC"),
  ncore = 1
)
```

Arguments

A	Adjacency matrix (n x n, can be sparse).
K.CAND	Candidate community numbers (integer vector, or a single integer interpreted as 1:K.CAND).
s	Number of non-overlapping subsamples.

o	Size of the overlapping set.
R	Number of independent repetitions.
tau	Regularisation parameter for the adjacency (default 0).
laplace	Logical; use graph Laplacian normalisation (default FALSE).
dc.est	DCBM estimation method: 1 = eigenvector, 2 = fast (default 2).
loss	Character vector of loss functions to evaluate. Supported: "l2", "bin.dev", "AUC".
ncore	Number of cores for parallel computation.

Value

A list containing:

loss	data.table of loss values by candidate model and K.
Candidate Models	"SBM and DCBM".
Candidate Values	The candidate K vector.
*.model	Selected model string (e.g. "SBM-3") for each loss.

croissant.latent	<i>CROISSANT for latent space model dimension selection</i>
------------------	---

Description

Selects the latent dimension for a latent space network model using the CROISSANT framework with MLE fitting.

Usage

```
croissant.latent(
  A,
  d.cand,
  s,
  o,
  R,
  loss = c("l2", "bin.dev", "AUC"),
  ncore = 1
)
```

Arguments

A	Adjacency matrix.
d.cand	Candidate embedding ranks.
s	Number of non-overlapping subsamples.
o	Overlap size.
R	Number of repetitions.
loss	Loss functions to evaluate.
ncore	Number of cores.

Value

A list with loss table and selected dimension per loss.

croissant.rdp	<i>CROISSANT for RDPG rank selection</i>
---------------	--

Description

Selects the embedding rank for a Random Dot Product Graph model using the CROISSANT overlapping node-subsample framework.

Usage

```
croissant.rdp(
  A,
  d.cand,
  s,
  o,
  R,
  laplace = FALSE,
  loss = c("l2", "bin.dev", "AUC"),
  ncore = 1
)
```

Arguments

A	Adjacency matrix.
d.cand	Candidate embedding ranks.
s	Number of non-overlapping subsamples.
o	Overlap size.
R	Number of repetitions.
laplace	Use Laplacian normalisation (default FALSE).
loss	Loss functions to evaluate.
ncore	Number of cores.

Value

A list with loss table and selected rank per loss.

croissant.tune.regsp *CROISSANT for regularisation parameter tuning in spectral methods*

Description

Selects the regularisation parameter tau for spectral clustering using the CROISSANT framework.

Usage

```
croissant.tune.regsp(
  A,
  K,
  tau.cand,
  DCBM = FALSE,
  s,
  o,
  R,
  laplace = FALSE,
  dc.est = 2,
  loss = c("l2", "bin.dev", "AUC"),
  ncore = 1
)
```

Arguments

A	Adjacency matrix.
K	Fixed number of communities.
tau.cand	Candidate regularisation parameter values.
DCBM	Logical; if TRUE, use row-normalised eigenvectors (DCBM).
s, o, R	CROISSANT design parameters.
laplace	Use Laplacian normalisation.
dc.est	DCBM estimation type (1 or 2).
loss	Loss functions to evaluate.
ncore	Number of cores.

Value

A list with loss table and selected tau per loss.

ECV.for.blockmodel *Edge Cross-Validation for blockmodel selection*

Description

Selects both the model type (SBM vs DCBM) and number of communities by holding out random edges and evaluating predictive performance.

Usage

```
ECV.for.blockmodel(
  A,
  max.K,
  cv = NULL,
  B = 3,
  holdout.p = 0.1,
  tau = 0,
  dc.est = 2,
  kappa = NULL
)
```

Arguments

A	Adjacency matrix (n x n).
max.K	Maximum number of communities to consider.
cv	Number of cross-validation folds (NULL for holdout; default NULL).
B	Number of holdout repetitions (default 3).
holdout.p	Fraction of edges held out (default 0.1).
tau	Regularisation parameter (default 0).
dc.est	DCBM estimation type (1 or 2; default 2).
kappa	Truncation parameter (default NULL).

Details

The algorithm is based on foundation code from Lin Zhang's PhD dissertation in 2019.

Value

A list with loss vectors and selected model strings:

l2	SBM L2 losses by K.
dev	SBM deviance losses by K.
auc	SBM AUC losses by K.
dc.l2, dc.dev, dc.auc	DCBM losses.
l2.model, dev.model, auc.model	Selected model strings.

ECV.undirected.Rank *Edge Cross-Validation for RDPG rank selection*

Description

Selects the embedding rank for an RDPG by holding out random edges.

Usage

```
ECV.undirected.Rank(  
  A,  
  max.K,  
  B = 3,  
  holdout.p = 0.1,  
  soft = FALSE,  
  fast = FALSE  
)
```

Arguments

A	Adjacency matrix.
max.K	Maximum rank to consider.
B	Number of holdout repetitions (default 3).
holdout.p	Fraction of edges held out (default 0.1).
soft	Logical (default FALSE).
fast	Logical (default FALSE).

Details

The algorithm is based on foundation code from Lin Zhang's PhD dissertation.

Value

A list with:

rank.sse, rank.dev, rank.auc	Selected ranks by each loss.
sse, dev, auc	Average loss vectors.

<code>edge.index.map</code>	<i>Map a linear edge index to row-column indices in the upper triangle</i>
-----------------------------	--

Description

Given a linear index u over the upper triangle of a symmetric matrix (column-major order), returns the corresponding row (x) and column (y) indices. The mapping is independent of matrix size.

Usage

```
edge.index.map(u)
```

Arguments

<code>u</code>	Integer vector of linear edge indices (1-based).
----------------	--

Value

A list with components x (row indices) and y (column indices).

Examples

```
edge.index.map(1:6)
```

<code>eigen.DCBM.est</code>	<i>Eigenvector-based DCBM estimation</i>
-----------------------------	--

Description

Estimates DCBM parameters using the row-norms of the leading eigenvectors for degree heterogeneity.

Usage

```
eigen.DCBM.est(A, g, n = nrow(A), K = max(g), psi.omit = 0, p.sample = 1)
```

Arguments

<code>A</code>	Adjacency matrix.
<code>g</code>	Community label vector.
<code>n</code>	Number of nodes.
<code>K</code>	Number of communities.
<code>psi.omit</code>	Number of leading nodes to exclude from psi estimation (used in CROISSANT overlapping designs).
<code>p.sample</code>	Sampling proportion for correction (default 1).

Value

A list with Bsum and psi.

fast.DCBM.est	<i>Fast DCBM parameter estimation</i>
---------------	---------------------------------------

Description

Estimates the block sum matrix and degree heterogeneity parameters for a Degree-Corrected Block Model.

Usage

```
fast.DCBM.est(A, g, n = nrow(A), K = max(g), psi.omit = 0, p.sample = 1)
```

Arguments

A	Adjacency matrix.
g	Community label vector.
n	Number of nodes.
K	Number of communities.
psi.omit	Number of leading nodes to exclude from psi estimation (used in CROISSANT overlapping designs).
p.sample	Sampling proportion for correction (default 1).

Value

A list with:

Bsum	K x K block sum matrix (divided by p.sample).
psi	Degree heterogeneity vector.

fast.SBM.est	<i>Fast SBM block probability estimation</i>
--------------	--

Description

Estimates the K x K block probability matrix from an adjacency matrix and community labels.

Usage

```
fast.SBM.est(A, g, n = nrow(A), K = max(g))
```

Arguments

A	Adjacency matrix (n x n).
g	Integer community label vector (length n).
n	Number of nodes.
K	Number of communities.

Value

K x K estimated block probability matrix.

l2	<i>L2 loss between two matrices</i>
----	-------------------------------------

Description

L2 loss between two matrices

Usage

`l2(x, y)`

Arguments

x, y	Numeric matrices of the same dimension.
------	---

Value

The Frobenius norm $\sqrt{\text{sum}((x - y)^2)}$.

latent.gen	<i>Generate a latent space network</i>
------------	--

Description

Generate a latent space network

Usage

`latent.gen(n, d, alpha = 1, sparsity = 1, ncore = 1)`

Arguments

n	Number of nodes.
d	Latent dimension.
alpha	Intercept parameter controlling overall density.
sparsity	Sparsity multiplier.
ncore	Number of cores for parallel computation.

Value

A list with:

A	Sparse symmetric adjacency matrix.
Z	n x d latent position matrix.

<code>matched.lab</code>	<i>Apply label permutation to match reference</i>
--------------------------	---

Description

Apply label permutation to match reference

Usage

```
matched.lab(lab, fixed, n = length(lab), K = max(lab, fixed))
```

Arguments

lab	Integer vector of labels to relabel.
fixed	Integer vector of reference labels.
n	Number of nodes.
K	Number of communities.

Value

Relabelled integer vector aligned to `fixed`.

NCV.for.blockmodel *Node Cross-Validation for blockmodel selection*

Description

Selects both the model type (SBM vs DCBM) and number of communities by holding out random nodes and evaluating predictive performance on the held-out subgraph.

Usage

```
NCV.for.blockmodel(A, max.K, cv = 3, dc.est = 1)
```

Arguments

A	Adjacency matrix (n x n).
max.K	Maximum number of communities to consider.
cv	Number of node folds (default 3).
dc.est	DCBM estimation type (1 or 2; default 1).

Value

A list with:

dev, l2, auc	SBM loss vectors by K.
dc.dev, dc.l2, dc.auc	DCBM loss vectors by K.
l2.model, dev.model, auc.model	Selected model strings.

neglog *Safe negative log-likelihood term*

Description

Computes $-n * \log(p)$, returning 0 when $p \leq 0$.

Usage

```
neglog(n = 1, p = 0.5)
```

Arguments

n	Numeric count.
p	Probability (between 0 and 1).

Value

Numeric value of $-n * \log(p)$ or 0 if $p \leq 0$.

nscv.f.fold

Inductive Node-Splitting Cross-Validation with f-fold splitting

Description

Performs f-fold node-split cross-validation to select the number of communities k in a Stochastic Block Model. Nodes are randomly partitioned into f folds; for each fold the held-out nodes are assigned to communities via the training-set spectral clustering, and the held-out negative log-likelihood and MSE are computed.

Usage

```
nscv.f.fold(
  A,
  k.vec = 2:6,
  restricted = TRUE,
  f = 10,
  method = "affinity",
  p.est.type = 3
)
```

Arguments

A	Symmetric adjacency matrix (n x n, binary).
k.vec	Integer vector of candidate community numbers (default 2:6).
restricted	Logical. If TRUE, use a restricted SBM (one within- and one between-community probability). Default TRUE.
f	Number of folds (default 10).
method	Inference method for assigning held-out nodes: "affinity" (default) assigns by maximum average connection, or "loss" assigns by minimum negative log-likelihood.
p.est.type	How to re-estimate the probability matrix for evaluation: 1 = training only, 2 = training + testing, 3 = testing only (default 3).

Value

A list with:

k.loss	Selected k that minimises CV negative log-likelihood.
k.mse	Selected k that minimises CV MSE.
cv.loss	Average CV negative log-likelihood for each k.
cv.mse	Average CV MSE for each k.

Examples

```
set.seed(42)
net <- community.sim(k = 3, n = 150, n1 = 50, p = 0.5, q = 0.1)
result <- nscv.f.fold(net$adjacency, k.vec = 2:5, f = 5)
result$k.loss
```

nscv.random.split *Inductive Node-Splitting Cross-Validation with random node splits*

Description

Performs repeated random-split node-level cross-validation. At each iteration a random fraction split of nodes is used for training.

Usage

```
nscv.random.split(
  A,
  k.vec = 2:6,
  restricted = TRUE,
  split = 0.66,
  ite = 100,
  method = "affinity",
  p.est.type = 3
)
```

Arguments

A	Symmetric adjacency matrix (n x n, binary).
k.vec	Integer vector of candidate community numbers (default 2:6).
restricted	Logical. If TRUE, use a restricted SBM (one within- and one between-community probability). Default TRUE.
split	Fraction of nodes used for training (default 0.66).
ite	Number of random split iterations (default 100).
method	Inference method for assigning held-out nodes: "affinity" (default) assigns by maximum average connection, or "loss" assigns by minimum negative log-likelihood.
p.est.type	How to re-estimate the probability matrix for evaluation: 1 = training only, 2 = training + testing, 3 = testing only (default 3).

Value

A list with:

<code>k.loss</code>	Selected k that minimises CV negative log-likelihood.
<code>k.mse</code>	Selected k that minimises CV MSE.
<code>cv.loss</code>	Average CV negative log-likelihood for each k.
<code>cv.mse</code>	Average CV MSE for each k.

SBM.prob	<i>Estimate SBM connection probabilities and negative log-likelihood</i>
----------	--

Description

Given a clustering and adjacency matrix, estimates the block probability matrix and computes the negative log-likelihood.

Usage

```
SBM.prob(cluster, k, A, restricted = TRUE)
```

Arguments

<code>cluster</code>	Integer vector of cluster labels.
<code>k</code>	Number of clusters.
<code>A</code>	Adjacency matrix corresponding to the nodes in cluster.
<code>restricted</code>	Logical. If TRUE, uses a restricted SBM with a single within-community probability p and a single between-community probability q . If FALSE, estimates a separate probability for each pair of communities.

Value

A list with:

<code>p.matrix</code>	$k \times k$ estimated block probability matrix.
<code>negloglike</code>	Negative log-likelihood of the model.

 SBM.spectral.clustering

Spectral clustering for a Stochastic Block Model

Description

Performs spectral clustering on an adjacency matrix by computing the top k singular vectors and applying k -means++ via `ClusterR::KMeans_rcpp`.

Usage

```
SBM.spectral.clustering(A, k = 2)
```

Arguments

`A` Symmetric adjacency matrix ($n \times n$, binary or weighted).
`k` Number of clusters (default 2).

Value

A list with component `cluster`, an integer vector of length n .

Examples

```
net <- community.sim(k = 3, n = 90, n1 = 30, p = 0.5, q = 0.1)
cl <- SBM.spectral.clustering(net$adjacency, k = 3)
table(cl$cluster, net$membership)
```

 sparse.RDPG.gen

Generate a sparse RDPG network

Description

Generate a sparse RDPG network

Usage

```
sparse.RDPG.gen(n, d, sparsity.multiplier = 1, ncore = 1)
```

Arguments

`n` Number of nodes.
`d` Latent dimension.
`sparsity.multiplier` Multiplier to control network density.
`ncore` Number of cores for parallel computation.

Value

A list with:

- A Sparse symmetric adjacency matrix.
- P n x n probability matrix.

Index

AUC, [2](#)

best.perm.label.match, [3](#)

bin.dev, [3](#)

blockmodel.gen.fast, [4](#)

community.sim, [5](#)

community.sim.sbm, [5](#)

croissant.blockmodel, [6](#)

croissant.latent, [7](#)

croissant.rdp, [8](#)

croissant.tune.regsp, [9](#)

ECV.for.blockmodel, [10](#)

ECV.undirected.Rank, [11](#)

edge.index.map, [12](#)

eigen.DCBM.est, [12](#)

fast.DCBM.est, [13](#)

fast.SBM.est, [13](#)

[12](#), [14](#)

latent.gen, [14](#)

matched.lab, [15](#)

NCV.for.blockmodel, [16](#)

neglog, [16](#)

nscv.f.fold, [17](#)

nscv.random.split, [18](#)

SBM.prob, [19](#)

SBM.spectral.clustering, [20](#)

sparse.RDPG.gen, [20](#)