### Package ‘graphon’

October 13, 2022

**Type** Package

**Title** A Collection of Graphon Estimation Methods

**Version** 0.3.5

**Description** Provides a not-so-comprehensive list of methods for estimating graphon, a symmetric measurable function, from a single or multiple of observed networks. For a detailed introduction on graphon and popular estimation techniques, see the paper by Orbanz, P. and Roy, D.M.(2014) <doi:10.1109/TPAMI.2014.2334607>. It also contains several auxiliary functions for generating sample networks using various network models and graphons.

**License** MIT + file LICENSE

**Encoding** UTF-8

**Suggests** igraph

**Imports** stats, graphics, ROptSpace, utils, Rdpack

**RdMacros** Rdpack

**RoxygenNote** 7.1.1

**NeedsCompilation** no

**Author** Kisung You [aut, cre] (<https://orcid.org/0000-0002-8584-459X>)

**Maintainer** Kisung You <kisungyou@outlook.com>

**Repository** CRAN

**Date/Publication** 2021-08-13 10:10:10 UTC

---

**R topics documented:**

- graphon-package
- cv.SBA
- est.completion
- est.LG
- est.nbdsmooth
- est.SBA
- est.USVT
- gmodel.block
Description

The **graphon** provides a not-so-comprehensive list of methods for estimating graphon, a symmetric measurable function, from a single or multiple of observed networks. It also contains several auxiliary functions for generating sample networks using various network models and graphons.

What is Graphon?

Graphon - graph function - is a symmetric measurable function

\[ W : [0, 1]^2 \rightarrow [0, 1] \]

that arise in studying exchangeable random graph models as well as sequence of dense graphs. In the language of graph theory, it can be understood as a two-stage procedural network modeling that 1) each vertex/node in the graph is assigned an independent random variable \( u_j \) from uniform distribution \( U[0, 1] \), and 2) each edge \((i, j)\) is randomly determined with probability \( W(u_i, u_j) \). Due to such procedural aspect, the term *probability matrix* and *graphon* will be interchangeably used in further documentation.

Composition of the package

The package mainly consists of two types of functions whose names start with 'est' and 'gmodel' for estimation algorithms and graph models, respectively.

The 'est' family has 4 estimation methods at the current version,

- **est.LG** for empirical degree sorting in stochastic blockmodel.
- **est.SBA** for stochastic blockmodel approximation.
- **est.USVT** for universal singular value thresholding.
- **est.nbdsmooth** for neighborhood smoothing.
- **est.completion** for matrix completion from a partially revealed data.

Also, the current release has following graph models implemented,

- **gmodel.P** generates a binary graph given an arbitrary probability matrix.
- **gmodel.ER** is an implementation of Erdos-Renyi random graph models.
- **gmodel.block** is used to generate networks with block structure.
- **gmodel.preset** has 10 exemplary graphon models for simulation.

Author(s)

Kisung You
Cross validation for selecting optimal precision parameter in SBA method.

Description

The performance of Stochastic Blockmodel Approximation (SBA) method is contingent on the number of blocks it finds during estimation process, which is roughly determined by a precision parameter δ. cv.SBA tests multiple of δ values to find the optimal one that minimizes the cross validation risk. Note that the optimal δ is not bound to be a single value.

Usage

```r
cv.SBA(A, vecdelta = seq(0.1, 1, by = 0.1))
```

Arguments

- **A**: either
  - **Case 1.** an \((n \times n)\) binary adjacency matrix, or
  - **Case 2.** a vector containing multiple of \((n \times n)\) binary adjacency matrices.

- **vecdelta**: a vector containing target δ values to be tested.

Value

A named list containing

- **optdelta**: optimal δ values that minimize the cross validation risk \(J\).
- **\(J\)**: cross validation risk values.

References


See Also

est.SBA
Examples

## Not run:
## generate a graphon of type No.8 with 3 clusters
W = gmodel.preset(3,id=8)

## create a probability matrix for 100 nodes
graphW = gmodel.block(W,n=100)
P = graphW$P

## draw 15 observations from a given probability matrix
A = gmodel.P(P,rep=15)

## cross validate SBA algorithm over different deltas
rescv = cv.SBA(A,vecdelta=c(0.1,0.5,0.9))
print(rescv$optdelta)

## End(Not run)

---

**est.completion**

Estimate graphons based on matrix completion scheme

Description

`est.completion` adopts a matrix completion scheme, which is common in missing data or matrix reconstruction studies. When given a multiple of, or a single observation, we consider non-existent edges as missing entries and apply the completion scheme. See `OptSpace` for a more detailed introduction.

Usage

```r
est.completion(
  A,
  rank = NA,
  tolerance = 0.001,
  maxiter = 20,
  progress = FALSE,
  adjust = TRUE
)
```

Arguments

- **A**
  - either
  - **Case 1.** an $(n \times n)$ binary adjacency matrix, or
  - **Case 2.** a list containing multiple of $(n \times n)$ binary adjacency matrices.

- **rank**
  - an estimated rank condition for the matrix; NA for automatic guessing of a rank, or a positive integer for a user-supplied rank assumption.
**est.LG**

Estimate graphons based on empirical degrees

**Description**

*est.LG* takes a 2-stage approach. First it adopts largest gap criterion on empirical degrees to estimate blocks of a given network under Stochastic Blockmodel framework. Then a consistent histogram estimator is utilized to estimate graphons based on estimated blocks in a given network.

**Value**

an \((n \times n)\) corresponding probability matrix.

**References**


**Examples**

```r
## generate a graphon of type No.5 with 3 clusters
W = gmodel.preset(3, id=5)

## create a probability matrix for 100 nodes
graphW = gmodel.block(W, n=100)
P = graphW$P

## draw 10 observations from a given probability matrix
A = gmodel.P(P, rep=10)

## apply the method
res_r3 = est.completion(A, rank=3)  # use rank-3 approximation
res_r9 = est.completion(A, rank=9)  # use rank-9 approximation
res_rN = est.completion(A, adjust=FALSE)  # stop the code if guess works poorly

## visualize
opar = par(no.readonly=TRUE)
par(mfrow=c(1,3), pty="s")
image(res_r3, main="rank 3")
image(res_r9, main="rank 9")
image(res_rN, main="rank is guessed")
par(opar)
```
Usage

est.LG(A, K = 2)

Arguments

A
an \((n \times n)\) binary adjacency matrix.

K
the number of blocks provided by an user.

Value

a named list containing

\(H\) a \((K \times K)\) matrix of 3D histogram.

\(P\) an \((n \times n)\) corresponding probability matrix.

\(B\) a length-\(K\) list where each element is a vector of nodes/indices for each cluster.

References


See Also

est.SBA

Examples

```r
## generate a graphon of type No.5 with 3 clusters
W = gmodel.preset(3, id=10)

## create a probability matrix for 20 nodes
graphW = gmodel.block(W, n=20)
P = graphW$P

## draw 23 observations from a given probability matrix
A = gmodel.P(P, rep=23, symmetric.out=TRUE)

## run LG algorithm with a rough guess for K=2,3,4
res2 = est.LG(A, K=2)
res3 = est.LG(A, K=3)
res4 = est.LG(A, K=4)

## compare true probability matrix and estimated ones
opar = par(no.readonly=TRUE)
par(mfrow=c(2,2), pty="s")
image(P, main="original P matrix")
image(res2$P, main="LG with K=2")
```
est.nbdsmooth

Estimate edge probabilities by neighborhood smoothing

Description

est.nbdsmooth takes the expectation of the adjacency matrix in that it directly aims at estimating network edge probabilities without imposing structural assumptions as of usual graphon estimation requires, such as piecewise lipschitz condition. Note that this method is for symmetric adjacency matrix only, i.e., undirected networks.

Usage

est.nbdsmooth(A)

Arguments

A either

Case 1. an \((n \times n)\) binary adjacency matrix, or

Case 2. a vector containing multiple of \((n \times n)\) binary adjacency matrices.

Value

a named list containing

h a quantile threshold value.

P a matrix of estimated edge probabilities.

References


Examples

```r
## generate a graphon of type No.4 with 3 clusters
W = gmodel.preset(3,id=4)

## create a probability matrix for 100 nodes
graphW = gmodel.block(W,n=100)
P = graphW$P

## draw 5 observations from a given probability matrix
A = gmodel.P(P,rep=5,symmetric.out=TRUE)
```
## run nbdsmooth algorithm
res2 = est.nbdsmooth(A)

## compare true probability matrix and estimated ones
opar = par(no.readonly=TRUE)
par(mfrow=c(1,2), pty="s")
image(P, main="original P matrix")
image(res2$P, main="nbdsmooth estimated P")
par(opar)

---

### est.SBA

**Estimate graphons based on Stochastic Blockmodel approximation**

#### Description

est.SBA takes a 2-stage approach for estimating graphons based on exchangeable random graph models. First, it finds a Stochastic Blockmodel Approximation (SBA) of the graphon. Then, it uses clustering information to estimate graphon using a consistent histogram estimator.

#### Usage

```r
est.SBA(A, delta = 0.5)
```

#### Arguments

- `A` either
  - Case 1. an \((n \times n)\) binary adjacency matrix, or
  - Case 2. a vector containing multiple of \((n \times n)\) binary adjacency matrices.

- `delta` a precision parameter larger than 0.

#### Value

a named list containing

- `H` a \((K \times K)\) matrix for 3D histogram.
- `P` an \((n \times n)\) corresponding probability matrix.
- `B` a length-\(K\) list where each element is a vector of nodes/indices for each cluster.

#### References


est.USVT

See Also

est.LG

Examples

```r
## generate a graphon of type No.6 with 3 clusters
W = gmodel.preset(3, id=6)

## create a probability matrix for 100 nodes
graphW = gmodel.block(W, n=100)
P = graphW$P

## draw 17 observations from a given probability matrix
A = gmodel.P(P, rep=17)

## run SBA algorithm with different deltas (0.2, 0.5, 0.8)
res2 = est.SBA(A, delta=0.2)
res3 = est.SBA(A, delta=0.5)
res4 = est.SBA(A, delta=0.8)

## compare true probability matrix and estimated ones
opar = par(no.readonly=TRUE)
par(mfrow=c(2,2), pty="s")
image(P); title("original P")
image(res2$P); title("SBA with delta=0.2")
image(res3$P); title("SBA with delta=0.5")
image(res4$P); title("SBA with delta=0.8")
par(opar)
```

est.USVT

Estimate graphons based via Universal Singular Value Thresholding

Description

est.USVT is a generic matrix estimation method first proposed for the case where a noisy realization of the matrix is given. Universal Singular Value Thresholding (USVT), as its name suggests, utilizes singular value decomposition of observations in addition to thresholding over singular values achieved from the decomposition.

Usage

```r
est.USVT(A, eta = 0.01)
```

Arguments

- `A` either
  - **Case 1.** an \((n \times n)\) binary adjacency matrix, or
  - **Case 2.** a list containing multiple of \((n \times n)\) binary adjacency matrices.
- `eta` a positive number in \((0, 1)\) to control the level of thresholding.
**Value**

a named list containing

**svs** a vector of sorted singular values.

**thr** a threshold to disregard singular values.

**P** a matrix of estimated edge probabilities.

**References**


**Examples**

```r
## generate a graph on type No.1 with 3 clusters
W = gmodel.preset(3,id=1)

## create a probability matrix for 100 nodes
graphW = gmodel.block(W,n=100)
P = graphW$P

## draw 5 observations from a given probability matrix
A = gmodel.P(P,rep=5,symmetric.out=TRUE)

## run USVT algorithm with different eta values (0.01,0.1)
res2 = est.USVT(A,eta=0.01)
res3 = est.USVT(A,eta=0.1)

## compare true probability matrix and estimated ones
opar = par(no.readonly=TRUE)
par(mfrow=c(1,3), pty="s")
image(P, main="original P matrix")
image(res2$P, main="USVT with eta=0.01")
image(res3$P, main="USVT with eta = 0.1")
par(opar)
```

---

**gmodel.block**

Generate binary random graphs based on stochastic blockmodel

---

**Description**

Given a \((K \times K)\) stochastic blockmodel \(W\), gmodel.block generates an \((n\)-by-\(n)\) binary random graphs. All K blocks have same number of nodes, or almost identical if \(n\) is not a multiple of \(K\). Parameter \(noloop\) controls whether generated observations have an edge from a node to itself, called a loop.
gmodel.block

Usage

gmodel.block(W, n, rep = 1, noloop = TRUE)

Arguments

W       a \((K \times K)\) blockmodel matrix.
n       the number of nodes for each observation.
rep     the number of observations to be generated.
noloop a logical value; TRUE for graphs without self-loops, FALSE otherwise.

Value

a named list containing

\(G\) depending on \(rep\) value,

\((\text{rep}=1)\) an \((n \times n)\) observation, or

\((\text{rep}>1)\) a length-\(rep\) list where each element is an observation is an \((n \times n)\) realization from the model.

\(P\) an \((n \times n)\) probability matrix of generating each edge.

See Also

gmodel.P

Examples

```r
## set inputs
W = matrix(c(0.9,0.2,0.2,0.7),nr=2)
n = 200

## generate 2 observations without self-loops.
out <- gmodel.block(W,n,rep=2,noloop=TRUE)

## visualize generated graphs
opar = par(no.readonly=TRUE)
par(mfrow=c(1,2), pty="s")
image(out$G[[1]]); title("Observation 1")
image(out$G[[2]]); title("Observation 2")
par(opar)
```
**gmodel.ER**

*Observations from Erdos-Renyi random graph model*

**Description**

Erdos-Renyi random graph model is one of the most popular and fundamental examples in modeling networks. Given n nodes, gmodel.ER generates edges randomly from Bernoulli distribution with a globally specified probability.

**Usage**

```r
gmodel.ER(n, mode = "prob", par = 0.5, rep = 1)
```

**Arguments**

- `n` the number of nodes to be generated
- `mode` 'prob' (default) for edges to be drawn from Bernoulli distribution independently, or 'num' for a graph to have a fixed number of edges placed randomly
- `par` a real number \( \in [0, 1] \) for mode='prob', or a positive integer \( \in [1, n \times (n - 1)/2] \) for mode='num'
- `rep` the number of observations to be generated.

**Details**

In network science, 'ER' model is often interchangeably used in where we have fixed number of edges to be placed at random. The original use of edge-generating probability is from Gilbert (1959). Therefore, we set this algorithm to be flexible in that user can create either a fixed number of edges placed at random or set global edge-generating probability and draw independent observations following Bernoulli distribution.

**Value**

depending on `rep` value, either

- (rep=1) an \((n \times n)\) observation matrix, or
- (rep>1) a length-rep list where each element is an observation is an \((n \times n)\) realization from the model.

**References**


### gmodel.P

**Generate graphs given a probability matrix**

**Description**

Given an \((n \times n)\) probability matrix \(P\), `gmodel.P` generates binary observation graphs corresponding to Bernoulli distribution whose parameter matches to the element of \(P\).

**Usage**

```r
gmodel.P(P, rep = 1, noloop = TRUE, symmetric.out = FALSE)
```

**Arguments**

- **P**: an \((n \times n)\) probability matrix.
- **rep**: the number of observations to be generated.
- **noloop**: a logical value; TRUE for graphs without self-loops, FALSE otherwise.
- **symmetric.out**: a logical value; FALSE for generated graphs to be nonsymmetric, TRUE otherwise. Note that TRUE is supported only if the input matrix \(P\) is symmetric.

**Value**

depending on `rep` value, either

- (rep=1) an \((n\times n)\) observation matrix, or
- (rep>1) a length-\(\text{rep}\) list where each element is an observation is an \((n\times n)\) realization from the model.
Examples

```r
## set inputs
modelP <- matrix(runif(16),nrow=4)

## generate 3 observations without self-loops.
out <- gmodel.P(modelP,rep=3,noloop=TRUE)

## visualize generated graphs
opar = par(no.readonly=TRUE)
par(mfrow=c(1,3), pty="s")
image(out[[1]], main="1st sample")
image(out[[2]], main="2nd sample")
image(out[[3]], main="3rd sample")
par(opar)
```

---

gmodel.preset

*Generate one of pre-specified graphons.*

Description

gmodel.preset generates one of pre-specified graphons of size \( n \times n \). Users can select one of 10 different graphons by their id, an integer from 1 to 10. The table of available graphons follows that of the reference article given below.

Usage

gmodel.preset(n, id = 1, sort = TRUE)

Arguments

- **n**: the number of nodes for a graphon to be generated.
- **id**: an integer from 1 to 10, each corresponding to a specific graphon model.
- **sort**: a logical value; TRUE to sort in an decreasing order of degree, FALSE otherwise.

Value

- an \((n \times n)\) graphon matrix.

References

Examples

## Not run:
## Generate 3 random graphons of nodal size 100.
  n = 100
  r3 = sample(1:10,3)
  W1 = gmodel.preset(n,id=r3[1])
  W2 = gmodel.preset(n,id=r3[2])
  W3 = gmodel.preset(n,id=r3[3])

## Generate corresponding observations and plot them
  A1 = gmodel.P(W1)
  A2 = gmodel.P(W2)
  A3 = gmodel.P(W3)

\dontshow{
  for (i in 1:10){
    W = gmodel.preset(100,id=i)
  }
}

## End(Not run)
Index

cv.SBA, 3
est.completion, 2, 4
est.LG, 2, 5, 9
est.nbdsmooth, 2, 7
est.SBA, 2, 3, 6, 8
est.USVT, 2, 9

gmodel.block, 2, 10
gmodel.ER, 2, 12
gmodel.P, 2, 11, 13
gmodel.preset, 2, 14
graphon-package, 2

OptSpace, 4