Package 'loggle'

April 16, 2018

Title Local Group Graphical Lasso Estimation

Version 1.0

Date 2018-04-11

Author Jilei Yang, Jie Peng

Maintainer Jilei Yang <jlyang@ucdavis.edu>

Description Provides a set of methods that learn time-varying graphical models based on data measured over a temporal grid. The underlying statistical model is motivated by the needs to describe and understand evolving interacting relationships among a set of random variables in many real applications, for instance the study of how stocks interact with each other and how such interactions change over time. The time-varying graphical models are estimated under the assumption that the graph topology changes gradually over time. For more details on estimating time-varying graphical models, please refer to: Yang, J. & Peng, J. (2018) <arXiv:1804.03811>.

Depends R (>= 3.0.2)

Imports Matrix (>= 1.2), doParallel (>= 1.0.8), foreach (>= 1.2.0), igraph (>= 0.7), glasso (>= 1.8), sm

Suggests sparseMVN, matrixcalc, XML, RCurl, quantmod

License GPL (>= 2)

NeedsCompilation yes

URL https://github.com/jlyang1990/loggle

Repository CRAN

Date/Publication 2018-04-16 09:16:59 UTC

R topics documented:

`example` *An example dataset of time-varying graphs*

Description

This dataset contains an example of time-varying graphs including precision matrices and observations, with number of variables being 15.

Usage

```
data(example)
```

Format

This dataset consists of two parts:

1. X - 15x1001: a data matrix containing 1001 observations of 15 variables on an equally-spaced time grid ranging from 0 to 1.

2. Omega.true - a list of length 1001 with each element being a 15x15 matrix: a list of precision matrices on an equally-spaced time grid ranging from 0 to 1.

Details

This dataset can be used for evaluating method performance via comparing the estimated time-varying graphs with the underlying true time-varying graphs. See examples in loggle, loggle.cv.vote, loggle.cv, loggle.cv.h for more details.

Author(s)

Yang, J. and Peng, J.

Source

The source code is in example_source.

References

Yang, J. & Peng, J. (2018), 'Estimating Time-Varying Graphical Models', arXiv preprint arXiv:1804.03811

example_source *Source code for generating time-varying graphs*

Description

This is the source code for generating time-varying graphs including precision matrices and observations in example.

Author(s)

Yang, J. and Peng, J.

References

Yang, J. & Peng, J. (2018), 'Estimating Time-Varying Graphical Models', arXiv preprint arXiv:1804.03811

Examples

```
library(Matrix)
library(sparseMVN)
library(matrixcalc)

# function to generate time-varying graphs and
# observations
loggle.graph <- function(p, N, alpha = 0.28) {

  done <- FALSE

  while(!done) {

    # generate time-varying lower triangular matrices
    t <- seq(0, 1, length = N)
    B1 <- matrix(rnorm(p^2, 0, 1/2), p, p)
    B2 <- matrix(rnorm(p^2, 0, 1/2), p, p)
    B3 <- matrix(rnorm(p^2, 0, 1/2), p, p)
    B4 <- matrix(rnorm(p^2, 0, 1/2), p, p)
    B1[upper.tri(B1)] <- 0
    B2[upper.tri(B2)] <- 0
    B3[upper.tri(B3)] <- 0
    B4[upper.tri(B4)] <- 0
    uni <- runif(4, -0.5, 0.5)

    Omega.true <- vector("list", N)
    X <- matrix(0, p, N)

    for(i in 1:N) {

      # generate raw precision matrix at time point i
      G <- (B1*sin(pi*t[i]/2+uni[1])
            + B2*cos(pi*t[i]/2+uni[2])
```

```
            + B3*sin(pi*t[i]/4+uni[3])
            + B4*cos(pi*t[i]/4+uni[4]))/2
        Omega <- G %*% t(G)
        Omega <- Omega %*% diag(1/sqrt((1:p)))
        Omega[upper.tri(Omega)] <- 0
        Omega <- Omega + t(Omega)
        Omega.diag <- diag(Omega)/(2*sqrt(1:p))+log(p,10)/4

        # implement soft-thresholding to off-diagonals in
        # precision matrix
        Omega.t1 <- matrix(1, p, p) - alpha/abs(Omega)
        Omega.t2 <- matrix(1, p, p) - alpha/(2*abs(Omega))
        Omega <- Omega.t2 * (Omega.t1 > 0) * Omega
        diag(Omega) <- Omega.diag

        if(!is.positive.definite(Omega)) {
          break
        }

        Omega.true[[i]] <- Matrix(Omega, sparse = TRUE)

        # generate observations from precision matrix
        X[, i] <- rmvn.sparse(1, rep(0, p),
        Cholesky(Omega.true[[i]]))

        if(i == N) {
          done = TRUE
        }
      }
    }

    result <- list(Omega.true = Omega.true, X = X)
    return(result)
  }

  # generate an example dataset of time-varying graphs
  # via loggle.graph
  set.seed(10)
  example <- loggle.graph(p = 15, N = 1001)
```

---

loggle *A function to learn time-varying graphical models*

Description

This function is to efficiently learn time-varying graphical models for a given set of tuning parameters.

Usage

```
loggle(X, pos = 1:ncol(X), h = 0.8*ncol(X)^(-1/5),
d = 0.2, lambda = 0.25, fit.type = "pseudo",
refit = TRUE, epi.abs = 1e-5, epi.rel = 1e-3,
max.step = 500, detrend = TRUE, fit.corr = TRUE,
pos.corr = 1:ncol(X), num.thread = 1,
print.detail = TRUE)
```

Arguments

X	a p by N data matrix containing observations on a time grid ranging from 0 to 1: p – number of variables, N – number of time points. The nominal time for the kth time point is (k-1)/(N-1)
pos	a vector constitutes a subset of {1, 2, ..., N}: indices of time points where graphs are estimated, default = 1:N
h	a scalar between 0 and 1: bandwidth in kernel smoothed sample covariance/correlation matrix, default = 0.8*N^(-1/5)
d	a scalar or a vector of the same length as pos with values between 0 and 1: width of neighborhood centered at each time point specified by pos, default = 0.2
lambda	a scalar or a vector of the same length as pos: tuning parameter of lasso penalty at each time point specified by pos, default = 0.25
fit.type	a string: "likelihood" – likelihood estimation, "pseudo" – pseudo likelihood estimation, or "space" – sparse partial correlation estimation, default = "pseudo"
refit	logic: if TRUE, conduct model refitting given learned graph structures, default = TRUE
epi.abs	a scalar: absolute tolerance in ADMM stopping criterion, default = 1e-5
epi.rel	a scalar: relative tolerance in ADMM stopping criterion, default = 1e-3
max.step	an integer: maximum iteration steps in ADMM iteration, default = 500
detrend	logic: if TRUE, subtract kernel weighted moving average for each variable in data matrix (i.e., detrending), if FALSE, subtract overall average for each variable in data matrix (i.e., centering), default = TRUE
fit.corr	logic: if TRUE, use sample correlation matrix in model fitting, if FALSE, use sample covariance matrix in model fitting, default = TRUE
pos.corr	a vector constitutes a subset of {1, 2, ..., N}: indices of time points used to get kernel smoothed sample covariance/correlation matrix, default = 1:N
num.thread	an integer: number of threads used in parallel computing, default = 1
print.detail	logic: if TRUE, print details in model fitting procedure, default = TRUE

Details

The model fitting method based on pseudo-likelihood (`fit.type = "pseudo"` or `fit.type = "space"`) is usually less computationally intensive than that based on likelihood (`fit.type = "likelihood"`), with similar model fitting performance.

`refit = TRUE` is recommended to prevent over-shrinkage of the estimated coefficients in time-varying graphs (precision matrices).

If no pre-processing has been done to the data matrix `X`, `detrend = TRUE` is recommended to detrend each variable in data matrix by subtracting corresponding kernel weighted moving average.

`fit.corr = TRUE` is recommended such that all the variables are of the same scale. If `fit.corr = FALSE` is used, the default value of `lambda` may need to be changed accordingly.

An ADMM (alternating directions method of multipliers) algorithm is implemented in learning time-varying graphical models.

Value

`Omega`	a list of estimated precision matrices at time points specified by `pos`
`edge.num`	a vector of numbers of edges at time points specified by `pos`
`edge`	a list of edges at time points specified by `pos`

Author(s)

Yang, J. and Peng, J.

References

Yang, J. & Peng, J. (2018), 'Estimating Time-Varying Graphical Models', arXiv preprint arXiv:1804.03811

See Also

loggle.cv for learning time-varying graphical models via cross validation, loggle.cv.select for model selection based on cross validation results, loggle.cv.vote for learning time-varying graphical models using cv.vote, loggle.cv.h for learning time-varying graphical models via cross validation (with `h` fixed), loggle.cv.select.h for model selection based on cross validation results (with `h` fixed), loggle.refit for model refitting based on estimated graphs, example for description of an example dataset of time-varying graphs, example_source for generating the example dataset of time-varying graphs, stockdata for description of stock price dataset of S&P 500 companies, stockdata_source for obtaining stock price dataset of S&P 500 companies.

Examples

```
data(example)  # load example dataset
# data matrix and true precision matrices
X <- example$X
Omega.true <- example$Omega.true
dim(X)  # dimension of data matrix
p <- nrow(X)  # number of variables

# positions of time points to estimate graphs
pos <- round(seq(0.1, 0.9, length=9)*(ncol(X)-1)+1)
K <- length(pos)
# estimate time-varying graphs
# num.thread can be set as large as number of cores
# on a multi-core machine (however when p is large,
```

```
# memory overflow should also be taken caution of)
ts <- proc.time()
result <- loggle(X, pos, h = 0.1, d = 0.15,
lambda = 0.25, fit.type = "pseudo", refit = TRUE,
num.thread = 1)
te <- proc.time()
sprintf("Time used for loggle: %.2fs", (te-ts)[3])

# number of edges at each time point
print(cbind("time" = seq(0.1, 0.9, length=9),
"edge.num" = result$edge.num))

# graph at each time point
library(igraph)
par(mfrow = c(3, 3))
for(k in 1:length(pos)) {
  adj.matrix <- result$Omega[[k]] != 0
  net <- graph.adjacency(adj.matrix, mode =
  "undirected", diag = FALSE)
  set.seed(0)
  plot(net, vertex.size = 10, vertex.color =
  "lightblue", vertex.label = NA, edge.color =
  "black", layout = layout.circle)
  title(main = paste("t =",
  round(pos[k]/(ncol(X)-1), 2)), cex.main = 0.8)
}

# false discovery rate (FDR) and power based on
# true precision matrices
edge.num.true <- sapply(1:K, function(i)
(sum(Omega.true[[pos[i]]]!=0)-p)/2)
edge.num.overlap <- sapply(1:K, function(i)
(sum(result$Omega[[i]] & Omega.true[[pos[i]]])-p)/2)
perform.matrix <- cbind(
"FDR" = 1 - edge.num.overlap / result$edge.num,
"power" = edge.num.overlap / edge.num.true)
print(apply(perform.matrix, 2, mean))
```

`loggle.cv` *A function to learn time-varying graphical models via cross validation*

Description

This function is to efficiently conduct model selection via cross validation for learning time-varying graphical models. It conducts grid search for optimal `h` (bandwidth in kernel smoothed sample covariance/correlation matrix), `d` (width of neighborhood centered at each position where a graph is estimated) and `lambda` (tuning parameter of lasso penalty at each position where a graph is estimated).

Usage

```
loggle.cv(X, pos = 1:ncol(X),
h.list = seq(0.1, 0.3, 0.05), d.list = c(0, 0.001, 0.01,
0.025, 0.05, 0.075, 0.1, 0.15, 0.2, 0.25, 0.3, 1),
lambda.list = seq(0.15, 0.35, 0.02), cv.fold = 5,
fit.type = "pseudo", return.select = TRUE,
select.type = "all_flexible", cv.vote.thres = 0.8,
early.stop.thres = 5, epi.abs = 1e-4, epi.rel = 1e-2,
max.step = 500, detrend = TRUE, fit.corr = TRUE,
h.correct = TRUE, num.thread = 1, print.detail = TRUE)
```

Arguments

`X`	a p by N data matrix containing observations on a time grid ranging from 0 to 1: p – number of variables, N – number of time points. The nominal time for the kth time point is (k-1)/(N-1)
`pos`	a vector constitutes a subset of {1, 2, ..., N}: indices of time points where graphs are estimated, default = 1:N
`h.list`	a vector with values between 0 and 1: a grid of bandwidths in kernel smoothed sample covariance/correlation matrix, default = seq(0.1, 0.3, 0.05)
`d.list`	a vector with values between 0 and 1: a grid of widths of neighborhood centered at each time point specified by `pos`, default = c(0, 0.001, 0.01, 0.025, 0.05, 0.075, 0.1, 0.15, 0.2, 0.25, 0.3, 1)
`lambda.list`	a vector: a grid of tuning parameters of lasso penalty at each time point specified by `pos`, default = seq(0.15, 0.35, 0.02)
`cv.fold`	a scalar: number of cross-validation folds, default = 5
`fit.type`	a string: "likelihood" – likelihood estimation, "pseudo" – pseudo likelihood estimation, or "space" – sparse partial correlation estimation, default = "pseudo"
`return.select`	logic: if TRUE, return model selection result, default = TRUE
`select.type`	a string: "all_flexible" – optimal `d` and `lambda` can vary across time points specified by `pos`, "d_fixed" – optimal `d` is fixed across time points specified by `pos` and optimal `lambda` can vary across time points specified by `pos`, "all_fixed" – optimal `d` and `lambda` are fixed across time points specified by `pos`, default = "all_flexible"
`cv.vote.thres`	a scalar between 0 and 1: an edge is kept after cv.vote if and only if it exists in no less than `cv.vote.thres`*`cv.fold` cv folds, default = 0.8
`early.stop.thres`	a scalar: stopping criterion in grid search – grid search stops when the ratio between number of detected edges and number of variables exceeds `early.stop.thres`, default = 5
`epi.abs`	a scalar or a vector of the same length as `d.list`: absolute tolerance in ADMM stopping criterion, default = 1e-4.
`epi.rel`	a scalar or a vector of the same length as `d.list`: relative tolerance in ADMM stopping criterion, default = 1e-2.

max.step	an integer: maximum iteration steps in ADMM iteration, default = 500
detrend	logic: if TRUE, subtract kernel weighted moving average for each variable in data matrix (i.e., detrending), if FALSE, subtract overall average for each variable in data matrix (i.e., centering), default = TRUE
fit.corr	logic: if TRUE, use sample correlation matrix in model fitting, if FALSE, use sample covariance matrix in model fitting, default = TRUE
h.correct	logic: if TRUE, apply bandwidth adjustment for validation sets due to sample size difference, default = TRUE
num.thread	an integer: number of threads used in parallel computing, default = 1
print.detail	logic: if TRUE, print details in model fitting procedure, default = TRUE

Details

This function conducts grid search for optimal h, d and lambda. It is a wrapper of loggle.cv.h and calls loggle.cv.h for each candidate h.

The model fitting method based on pseudo-likelihood (fit.type = "pseudo" or fit.type = "space") is usually less computationally intensive than that based on likelihood (fit.type = "likelihood"), with similar model fitting performance.

select.type = "all_flexible" is for the situation where we expect both the extent of structure smoothness (controlled by d) and the extent of graph sparsity (controlled by lambda) vary across time points. If only the extent of graph sparsity varies across time points, select.type = "d_fixed" should be used. If both of them are expected to be similar across time points, select.type = "all_fixed" should be used.

cv.vote.thres controls the tradeoff between false discovery rate and power in model selection. A large value of cv.vote.thres would decrease false discovery rate but also hurt power.

early.stop.thres is used as an early stopping criterion. With limited sample size, a very complicated model seldom leads to a competitive cv score. Therefore in grid search for lambda, when the estimated model is already large determined by early.stop.thres, we stop fitting models for smaller lambda's (as they often lead to more complicated models). In this case, the output corresponding to the remaining lambda's will be NA.

If no pre-processing has been done to the data matrix X, detrend = TRUE is recommended to detrend each variable in data matrix by subtracting corresponding kernel weighted moving average.

fit.corr = TRUE is recommended such that all the variables are of the same scale. If fit.corr = FALSE is used, the default value of lambda may need to be changed accordingly.

h.correct = TRUE is recommended in calculating kernel smoothed sample covariance matrix for validation sets. This is because bandwidth h should reflect the difference in sample sizes between training and validation sets.

Value

cv.result.h	a list of model fitting results from loggle.cv.h for each h
cv.select.result	results from loggle.cv.select if return.select = TRUE

Author(s)

Yang, J. and Peng, J.

References

Yang, J. & Peng, J. (2018), 'Estimating Time-Varying Graphical Models', arXiv preprint arXiv:1804.03811

See Also

loggle for learning time-varying graphical models, loggle.cv.select for model selection based on cross validation results, loggle.cv.vote for learning time-varying graphical models using cv.vote, loggle.cv.h for learning time-varying graphical models via cross validation (with h fixed), loggle.cv.select.h for model selection based on cross validation results (with h fixed), loggle.refit for model refitting based on estimated graphs, example for description of an example dataset of time-varying graphs, example_source for generating the example dataset of time-varying graphs, stockdata for description of stock price dataset of S&P 500 companies, stockdata_source for obtaining stock price dataset of S&P 500 companies.

Examples

```
data(example)  # load example dataset
# data matrix and true precision matrices
X <- example$X
Omega.true <- example$Omega.true
dim(X)  # dimension of data matrix
p <- nrow(X)  # number of variables

# positions of time points to estimate graphs
pos <- round(seq(0.1, 0.9, length=9)*(ncol(X)-1)+1)
K <- length(pos)
# estimate time-varying graphs and conduct model
# selection via cross-validation
# num.thread can be set as large as number of cores
# on a multi-core machine (however when p is large,
# memory overflow should also be taken caution of)
ts <- proc.time()
result <- loggle.cv(X, pos, h.list = c(0.2, 0.25),
d.list = c(0, 0.05, 0.15, 1), lambda.list
= c(0.2, 0.25), cv.fold = 3, fit.type = "pseudo",
cv.vote.thres = 1, num.thread = 1)
te <- proc.time()
sprintf("Time used for loggle.cv: %.2fs", (te-ts)[3])

# optimal values of h, d and lambda, and number of
# selected edges at each time point
select.result <- result$cv.select.result
print(cbind("time" = seq(0.1, 0.9, length=9),
"h.opt" = rep(select.result$h.opt, K),
"d.opt" = select.result$d.opt,
"lambda.opt" = select.result$lambda.opt,
"edge.num.opt" = select.result$edge.num.opt))
```

```
# false discovery rate (FDR) and power based on
# true precision matrices for selected model
edge.num.opt <- select.result$edge.num.opt
edge.num.true <- sapply(1:K, function(i)
(sum(Omega.true[[pos[i]]]!=0)-p)/2)
edge.num.overlap <- sapply(1:K, function(i)
(sum(select.result$adj.mat.opt[[i]]
& Omega.true[[pos[i]]])-p)/2)
perform.matrix <- cbind(
"FDR" = 1 - edge.num.overlap / edge.num.opt,
"power" = edge.num.overlap / edge.num.true)
print(apply(perform.matrix, 2, mean))
```

`loggle.cv.h` *A function to learn time-varying graphical models via cross validation (with* `h` *fixed)*

Description

This function is to efficiently conduct model selection via cross validation for learning time-varying graphical models. Different from loggle.cv, `h` (bandwidth in kernel smoothed sample covariance/correlation matrix) should be pre-specified.

Usage

```
loggle.cv.h(X, pos = 1:ncol(X),
h = 0.8*ncol(X)^(-1/5), d.list = c(0, 0.001, 0.01,
0.025, 0.05, 0.075, 0.1, 0.15, 0.2, 0.25, 0.3, 1),
lambda.list = seq(0.15, 0.35, 0.02), cv.fold = 5,
fit.type = "pseudo", return.select = TRUE,
select.type = "all_flexible", cv.vote.thres = 0.8,
early.stop.thres = 5, epi.abs = 1e-4, epi.rel = 1e-2,
max.step = 500, detrend = TRUE, fit.corr = TRUE,
h.correct = TRUE, num.thread = 1, print.detail = TRUE)
```

Arguments

`X`	a p by N data matrix containing observations on a time grid ranging from 0 to 1: p – number of variables, N – number of time points. The nominal time for the kth time point is (k-1)/(N-1)
`pos`	a vector constitutes a subset of {1, 2, ..., N}: indices of time points where graphs are estimated, default = 1:N
`h`	a scalar between 0 and 1: bandwidth in kernel smoothed sample covariance/correlation matrix, default = 0.8*N^(-1/5)
`d.list`	a vector with values between 0 and 1: a grid of widths of neighborhood centered at each time point specified by `pos`, default = c(0, 0.001, 0.01, 0.025, 0.05, 0.075, 0.1, 0.15, 0.2, 0.25, 0.3, 1)

`lambda.list`	a vector: a grid of tuning parameters of lasso penalty at each time point specified by `pos`, default = seq(0.15, 0.35, 0.02)
`cv.fold`	a scalar: number of cross-validation folds, default = 5
`fit.type`	a string: "likelihood" – likelihood estimation, "pseudo" – pseudo likelihood estimation, or "space" – sparse partial correlation estimation, default = "pseudo"
`return.select`	logic: if TRUE, return model selection result, default = TRUE
`select.type`	a string: "all_flexible" – optimal `d` and `lambda` can vary across time points specified by `pos`, "d_fixed" – optimal `d` is fixed across time points specified by `pos` and optimal `lambda` can vary across time points specified by `pos`, "all_fixed" – optimal `d` and `lambda` are fixed across time points specified by `pos`, default = "all_flexible"
`cv.vote.thres`	a scalar between 0 and 1: an edge is kept after cv.vote if and only if it exists in no less than `cv.vote.thres`*`cv.fold` cv folds, default = 0.8
`early.stop.thres`	a scalar: stopping criterion in grid search – grid search stops when the ratio between number of detected edges and number of variables exceeds `early.stop.thres`, default = 5
`epi.abs`	a scalar or a vector of the same length as `d.list`: absolute tolerance in ADMM stopping criterion, default = 1e-4.
`epi.rel`	a scalar or a vector of the same length as `d.list`: relative tolerance in ADMM stopping criterion, default = 1e-2.
`max.step`	an integer: maximum iteration steps in ADMM iteration, default = 500
`detrend`	logic: if TRUE, subtract kernel weighted moving average for each variable in data matrix (i.e., detrending), if FALSE, subtract overall average for each variable in data matrix (i.e., centering), default = TRUE
`fit.corr`	logic: if TRUE, use sample correlation matrix in model fitting, if FALSE, use sample covariance matrix in model fitting, default = TRUE
`h.correct`	logic: if TRUE, apply bandwidth adjustment for validation sets due to sample size difference, default = TRUE
`num.thread`	an integer: number of threads used in parallel computing, default = 1
`print.detail`	logic: if TRUE, print details in model fitting procedure, default = TRUE

Details

This function conducts grid search for optimal `d` and `lambda`, while `h` is pre-specified. To select the optimal `h` as well, one can use loggle.cv or apply this function to different candidate `h`'s.

The model fitting method based on pseudo-likelihood (`fit.type = "pseudo"` or `fit.type = "space"`) is usually less computationally intensive than that based on likelihood (`fit.type = "likelihood"`), with similar model fitting performance.

`select.type = "all_flexible"` is for the situation where we expect both the extent of structure smoothness (controlled by `d`) and the extent of graph sparsity (controlled by `lambda`) vary across time points. If only the extent of graph sparsity varies across time points, `select.type = "d_fixed"` should be used. If both of them are expected to be similar across time points, `select.type = "all_fixed"` should be used.

`cv.vote.thres` controls the tradeoff between false discovery rate and power in model selection. A large value of `cv.vote.thres` would decrease false discovery rate but also hurt power.

`early.stop.thres` is used as an early stopping criterion. With limited sample size, a very complicated model seldom leads to a competitive cv score. Therefore in grid search for `lambda`, when the estimated model is already large determined by `early.stop.thres`, we stop fitting models for smaller `lambda`'s (as they often lead to more complicated models). In this case, the output corresponding to the remaining `lambda`'s will be `NA`.

If no pre-processing has been done to the data matrix `X`, `detrend = TRUE` is recommended to detrend each variable in data matrix by subtracting corresponding kernel weighted moving average.

`fit.corr = TRUE` is recommended such that all the variables are of the same scale. If `fit.corr = FALSE` is used, the default value of `lambda` may need to be changed accordingly.

`h.correct = TRUE` is recommended in calculating kernel smoothed sample covariance matrix for validation sets. This is because bandwidth `h` should reflect the difference in sample sizes between training and validation sets.

Value

`cv.score`	an array of cv scores for each combination of d, lambda, time point and cv fold
`cv.result.fold`	a list of model fitting results for each cv fold, including a list of estimated precision matrices for each combination of d, lambda and time point, an array of numbers of edges for each combination of d, lambda and time point, and a list of edges for each combination of d, lambda and time point
`cv.select.result`	results from loggle.cv.select.h if `return.select = TRUE`

Author(s)

Yang, J. and Peng, J.

References

Yang, J. & Peng, J. (2018), 'Estimating Time-Varying Graphical Models', arXiv preprint arXiv:1804.03811

See Also

loggle for learning time-varying graphical models, loggle.cv for learning time-varying graphical models via cross validation, loggle.cv.select for model selection based on cross validation results.

Examples

```
data(example)  # load example dataset
# data matrix and true precision matrices
X <- example$X
Omega.true <- example$Omega.true
dim(X)  # dimension of data matrix
p <- nrow(X)  # number of variables

# positions of time points to estimate graphs
pos <- round(seq(0.1, 0.9, length=9)*(ncol(X)-1)+1)
```

```
K <- length(pos)
# estimate time-varying graphs and conduct model
# selection via cross-validation
# num.thread can be set as large as number of cores
# on a multi-core machine (however when p is large,
# memory overflow should also be taken caution of)
ts <- proc.time()
result <- loggle.cv.h(X, pos, h = 0.2,
d.list = c(0, 0.05, 0.15, 1), lambda.list
= c(0.2, 0.25), cv.fold = 3, fit.type = "pseudo",
cv.vote.thres = 1, num.thread = 1)
te <- proc.time()
sprintf("Time used for loggle.cv.h: %.2fs", (te-ts)[3])

# optimal values of d and lambda, and number of
# selected edges at each time point
select.result <- result$cv.select.result
print(cbind("time" = seq(0.1, 0.9, length=9),
"d.opt" = select.result$d.opt,
"lambda.opt" = select.result$lambda.opt,
"edge.num.opt" = select.result$edge.num.opt))

# false discovery rate (FDR) and power based on
# true precision matrices for selected model
edge.num.opt <- select.result$edge.num.opt
edge.num.true <- sapply(1:K, function(i)
(sum(Omega.true[[pos[i]]]!=0)-p)/2)
edge.num.overlap <- sapply(1:K, function(i)
(sum(select.result$adj.mat.opt[[i]]
& Omega.true[[pos[i]]])-p)/2)
perform.matrix <- cbind(
"FDR" = 1 - edge.num.overlap / edge.num.opt,
"power" = edge.num.overlap / edge.num.true)
print(apply(perform.matrix, 2, mean))
```

`loggle.cv.select` *A function to conduct model selection based on cross validation results*

Description

This function is to conduct model selection for time-varying graphical models based on cross validation results from loggle.cv.

Usage

```
loggle.cv.select(cv.result, select.type = "all_flexible",
cv.vote.thres = 0.8)
```

Arguments

cv.result	a list: results from loggle.cv
select.type	a string: "all_flexible" – optimal d and lambda can vary across time points specified by pos, "d_fixed" – optimal d is fixed across time points specified by pos and optimal lambda can vary across time points specified by pos, "all_fixed" – optimal d and lambda are fixed across time points specified by pos, default = "all_flexible"
cv.vote.thres	a scalar between 0 and 1: an edge is kept after cv.vote if and only if it exists in no less than cv.vote.thres*cv.fold cv folds, default = 0.8

Details

select.type = "all_flexible" is for the situation where we expect both the extent of structure smoothness (controlled by d) and the extent of graph sparsity (controlled by lambda) vary across time points. If only the extent of graph sparsity varies across time points, select.type = "d_fixed" should be used. If both of them are expected to be similar across time points, select.type = "all_fixed" should be used.

cv.vote.thres controls the tradeoff between false discovery rate and power in model selection. A large value of cv.vote.thres would decrease false discovery rate but also hurt power.

Value

h.opt	optimal value of h
d.opt	a vector of optimal values of d for each estimated graph
lambda.opt	a vector of optimal values of lambda for each estimated graph
cv.score.opt	optimal cv score (averaged over time points and cv folds)
edge.num.opt	a vector of numbers of edges for each estimated graph
edge.opt	a list of edges for each estimated graph
adj.mat.opt	a list of adjacency matrices for each estimated graph

Author(s)

Yang, J. and Peng, J.

References

Yang, J. & Peng, J. (2018), 'Estimating Time-Varying Graphical Models', arXiv preprint arXiv:1804.03811

See Also

loggle for learning time-varying graphical models, loggle.cv for learning time-varying graphical models via cross validation, loggle.cv.vote for learning time-varying graphical models using cv.vote, loggle.cv.h for learning time-varying graphical models via cross validation (with h fixed), loggle.cv.select.h for model selection based on cross validation results (with h fixed), loggle.refit for model refitting based on estimated graphs.

Examples

```
data(example)  # load example dataset
X <- example$X  # data matrix
dim(X)  # dimension of data matrix

# positions of time points to estimate graphs
pos <- round(seq(0.1, 0.9, length=9)*(ncol(X)-1)+1)
# estimate time-varying graphs via cross-validation
result <- loggle.cv(X, pos, h.list = c(0.2, 0.25),
d.list = c(0, 0.05, 0.15, 1), lambda.list
= c(0.2, 0.25), cv.fold = 3, fit.type = "pseudo",
cv.vote.thres = 1, num.thread = 1)

# conduct model selection using cross-validation results
select.result <- loggle.cv.select(cv.result = result,
select.type = "all_flexible", cv.vote.thres = 0.8)

# optimal values of h, d and lambda, and number of
# selected edges at each time point
print(cbind("time" = seq(0.1, 0.9, length=9),
"h.opt" = rep(select.result$h.opt, length(pos)),
"d.opt" = select.result$d.opt,
"lambda.opt" = select.result$lambda.opt,
"edge.num.opt" = select.result$edge.num.opt))
```

`loggle.cv.select.h` *A function to conduct model selection based on cross validation results (with* `h` *fixed)*

Description

This function is to conduct model selection for time-varying graphical models based on cross validation results from loggle.cv.h.

Usage

```
loggle.cv.select.h(cv.result, select.type = "all_flexible",
cv.vote.thres = 0.8)
```

Arguments

`cv.result`	a list: results from loggle.cv.h
`select.type`	a string: "all_flexible" – optimal `d` and `lambda` can vary across time points specified by `pos`, "d_fixed" – optimal `d` is fixed across time points specified by `pos` and optimal `lambda` can vary across time points specified by `pos`, "all_fixed" – optimal `d` and `lambda` are fixed across time points specified by `pos`, default = "all_flexible"
`cv.vote.thres`	a scalar between 0 and 1: an edge is kept after cv.vote if and only if it exists in no less than `cv.vote.thres*cv.fold` cv folds, default = 0.8

Details

`select.type = "all_flexible"` is for the situation where we expect both the extent of structure smoothness (controlled by `d`) and the extent of graph sparsity (controlled by `lambda`) vary across time points. If only the extent of graph sparsity varies across time points, `select.type = "d_fixed"` should be used. If both of them are expected to be similar across time points, `select.type = "all_fixed"` should be used.

`cv.vote.thres` controls the tradeoff between false discovery rate and power in model selection. A large value of `cv.vote.thres` would decrease false discovery rate but also hurt power.

Value

`d.opt`	a vector of optimal values of `d` for each estimated graph
`lambda.opt`	a vector of optimal values of `lambda` for each estimated graph
`cv.score.opt`	optimal cv score (averaged over time points and cv folds)
`edge.num.opt`	a vector of numbers of edges for each estimated graph
`edge.opt`	a list of edges for each estimated graph
`adj.mat.opt`	a list of adjacency matrices for each estimated graph

Author(s)

Yang, J. and Peng, J.

References

Yang, J. & Peng, J. (2018), 'Estimating Time-Varying Graphical Models', arXiv preprint arXiv:1804.03811

See Also

loggle for learning time-varying graphical models, loggle.cv for learning time-varying graphical models via cross validation, loggle.cv.select for model selection based on cross validation results.

Examples

```
data(example)  # load example dataset
X <- example$X  # data matrix
dim(X)  # dimension of data matrix

# positions of time points to estimate graphs
pos <- round(seq(0.1, 0.9, length=9)*(ncol(X)-1)+1)
# estimate time-varying graphs via cross-validation
result <- loggle.cv.h(X, pos, h = 0.2,
d.list = c(0, 0.05, 0.15, 1), lambda.list
= c(0.2, 0.25), cv.fold = 3, fit.type = "pseudo",
cv.vote.thres = 1, num.thread = 1)

# conduct model selection using cross-validation results
select.result <- loggle.cv.select.h(cv.result = result,
select.type = "all_flexible", cv.vote.thres = 0.8)
```

```
# optimal values of d and lambda, and number of
# selected edges at each time point
print(cbind("time" = seq(0.1, 0.9, length=9),
"d.opt" = select.result$d.opt,
"lambda.opt" = select.result$lambda.opt,
"edge.num.opt" = select.result$edge.num.opt))
```

`loggle.cv.vote` *A function to learn time-varying graphical models using cv.vote*

Description

This function is to efficiently learn time-varying graphical models for a given set of tuning parameters. Different from loggle, cv.vote is also applied in estimating time-varying graphs.

Usage

```
loggle.cv.vote(X, pos = 1:ncol(X),
h = 0.8*ncol(X)^(-1/5), d = 0.2, lambda = 0.25,
cv.fold = 5, fit.type = "pseudo", refit = TRUE,
cv.vote.thres = 0.8, epi.abs = 1e-5, epi.rel = 1e-3,
max.step = 500, detrend = TRUE, fit.corr = TRUE,
num.thread = 1, print.detail = TRUE)
```

Arguments

`X`	a p by N data matrix containing observations on a time grid ranging from 0 to 1: p – number of variables, N – number of time points. The nominal time for the kth time point is (k-1)/(N-1)
`pos`	a vector constitutes a subset of {1, 2, ..., N}: indices of time points where graphs are estimated, default = 1:N
`h`	a scalar between 0 and 1: bandwidth in kernel smoothed sample covariance/correlation matrix, default = 0.8*N^(-1/5)
`d`	a scalar or a vector of the same length as `pos` with values between 0 and 1: width of neighborhood centered at each time point specified by `pos`, default = 0.2
`lambda`	a scalar or a vector of the same length as `pos`: tuning parameter of lasso penalty at each time point specified by `pos`, default = 0.25
`cv.fold`	a scalar: number of cross-validation folds, default = 5
`fit.type`	a string: "likelihood" – likelihood estimation, "pseudo" – pseudo likelihood estimation, or "space" – sparse partial correlation estimation, default = "pseudo"
`refit`	logic: if TRUE, conduct model refitting given learned graph structures, default = TRUE
`cv.vote.thres`	a scalar between 0 and 1: an edge is kept after cv.vote if and only if it exists in no less than `cv.vote.thres`*`cv.fold` cv folds, default = 0.8

epi.abs	a scalar: absolute tolerance in ADMM stopping criterion, default = 1e-5
epi.rel	a scalar: relative tolerance in ADMM stopping criterion, default = 1e-3
max.step	an integer: maximum iteration steps in ADMM iteration, default = 500
detrend	logic: if TRUE, subtract kernel weighted moving average for each variable in data matrix (i.e., detrending), if FALSE, subtract overall average for each variable in data matrix (i.e., centering), default = TRUE
fit.corr	logic: if TRUE, use sample correlation matrix in model fitting, if FALSE, use sample covariance matrix in model fitting, default = TRUE
num.thread	an integer: number of threads used in parallel computing, default = 1
print.detail	logic: if TRUE, print details in model fitting procedure, default = TRUE

Details

The idea of cross-validation is implemented in this function, where loggle is applied on each cross-validation fold to get fold-wise estimated time-varying graphs, and cv.vote is then applied across cross-validation folds to get the final version of estimated time-varying graphs.

The model fitting method based on pseudo-likelihood (`fit.type = "pseudo"` or `fit.type = "space"`) is usually less computationally intensive than that based on likelihood (`fit.type = "likelihood"`), with similar model fitting performance.

`cv.vote.thres` controls the tradeoff between false discovery rate and power. A large value of `cv.vote.thres` would decrease false discovery rate but also hurt power.

If no pre-processing has been done to the data matrix `X`, `detrend = TRUE` is recommended to detrend each variable in data matrix by subtracting corresponding kernel weighted moving average.

`fit.corr = TRUE` is recommended such that all the variables are of the same scale. If `fit.corr = FALSE` is used, the default value of `lambda` may need to be changed accordingly.

Value

result.fold	a list of model fitting results from loggle for each cv fold
Omega	a list of estimated precision matrices at time points specified by pos
edge.num	a vector of numbers of edges at time points specified by pos
edge	a list of edges at time points specified by pos

Author(s)

Yang, J. and Peng, J.

References

Yang, J. & Peng, J. (2018), 'Estimating Time-Varying Graphical Models', arXiv preprint arXiv:1804.03811

See Also

loggle for learning time-varying graphical models, loggle.cv for learning time-varying graphical models via cross validation, loggle.cv.select for model selection based on cross validation results.

Examples

```
data(example)  # load example dataset
# data matrix and true precision matrices
X <- example$X
Omega.true <- example$Omega.true
dim(X)  # dimension of data matrix
p <- nrow(X)  # number of variables

# positions of time points to estimate graphs
pos <- round(seq(0.1, 0.9, length=9)*(ncol(X)-1)+1)
K <- length(pos)
# estimate time-varying graphs
# num.thread can be set as large as number of cores
# on a multi-core machine (however when p is large,
# memory overflow should also be taken caution of)
ts <- proc.time()
result <- loggle.cv.vote(X, pos, h = 0.1, d = 0.15,
lambda = 0.25, cv.fold = 3, fit.type = "pseudo",
refit = TRUE, cv.vote.thres = 0.8, num.thread = 1)
te <- proc.time()
sprintf("Time used for loggle.cv.vote: %.2fs", (te-ts)[3])

# number of edges at each time point
print(cbind("time" = seq(0.1, 0.9, length=9),
"edge.num" = result$edge.num))

# graph at each time point
library(igraph)
par(mfrow = c(3, 3))
for(k in 1:length(pos)) {
  adj.matrix <- result$Omega[[k]] != 0
  net <- graph.adjacency(adj.matrix, mode =
  "undirected", diag = FALSE)
  set.seed(0)
  plot(net, vertex.size = 10, vertex.color =
  "lightblue", vertex.label = NA, edge.color =
  "black", layout = layout.circle)
  title(main = paste("t =",
  round(pos[k]/(ncol(X)-1), 2)), cex.main = 0.8)
}

# false discovery rate (FDR) and power based on
# true precision matrices
edge.num.true <- sapply(1:K, function(i)
(sum(Omega.true[[pos[i]]]!=0)-p)/2)
edge.num.overlap <- sapply(1:K, function(i)
(sum(result$Omega[[i]] & Omega.true[[pos[i]]])-p)/2)
perform.matrix <- cbind(
"FDR" = 1 - edge.num.overlap / result$edge.num,
"power" = edge.num.overlap / edge.num.true)
print(apply(perform.matrix, 2, mean))
```

`loggle.refit` *A function to conduct model refitting given learned graph structures*

Description

This function is to efficiently conduct model refitting given learned time-varying graph structures. Time-varying graph structures can be learned from loggle, loggle.cv.vote, loggle.cv or loggle.cv.h.

Usage

```
loggle.refit(X, pos, adj.mat, h = 0.8*ncol(X)^(-1/5),
print.detail = TRUE)
```

Arguments

`X`	a p by N data matrix containing observations on a time grid ranging from 0 to 1: p – number of variables, N – number of time points. The nominal time for the kth time point is (k-1)/(N-1)
`pos`	a vector constitutes a subset of {1, 2, ..., N}: indices of time points where graphs are estimated, default = 1:N
`adj.mat`	a list of the same length as `pos`: adjacency matrices at time points specified by `pos`
`h`	a scalar between 0 and 1: bandwidth in kernel smoothed sample covariance/correlation matrix, default = 0.8*N^(-1/5)
`print.detail`	logic: if TRUE, print details in model refitting procedure, default = TRUE

Details

Function "glasso" in R package `glasso` is applied in model refitting.

Value

`Omega`	a list of estimated precision matrices at time points specified by `pos`

Author(s)

Yang, J. and Peng, J.

References

Yang, J. & Peng, J. (2018), 'Estimating Time-Varying Graphical Models', arXiv preprint arXiv:1804.03811

See Also

loggle for learning time-varying graphical models, loggle.cv for learning time-varying graphical models via cross validation, loggle.cv.select for model selection based on cross validation results.

Examples

```
data(example)  # load example dataset
X <- example$X  # data matrix
dim(X)  # dimension of data matrix

# positions of time points to estimate graphs
pos <- round(seq(0.1, 0.9, length=9)*(ncol(X)-1)+1)
# estimate time-varying graphs and conduct model
# selection via cross-validation
result <- loggle.cv.h(X, pos, h = 0.2,
d.list = c(0, 0.05, 0.15, 1), lambda.list
= c(0.2, 0.25), cv.fold = 3, fit.type = "pseudo",
cv.vote.thres = 1, num.thread = 1)

# estimated adjacency matrices at each time point
adj.mat.opt <- result$cv.select.result$adj.mat.opt
# estimated precision matrices at each time point
# via model refitting
Omega.opt <- loggle.refit(X, pos, adj.mat.opt)
```

---

stockdata *Stock prices of S&P 500 companies from 2007 to 2016*

Description

This dataset contains stock closing prices and company information of S&P 500 companies (updated on 2017-01-01).

Usage

```
data(stockdata)
```

Format

This dataset consists of two matrices:

1. data - 2518x455: 455 stocks' closing prices at 2518 trading days from 2007 to 2016. Trading dates are represented as row name in data matrix and company names are represented as column name in data matrix.

2. info - 455x3: 1st column: ticker symbol of each company. 2nd column: Global Industry Classification Standard (GICS) sector of each company. 3rd column: full name of each company.

Details

This dataset can be used in time-varying graph estimation to reveal the dynamic relationships between S&P 500 companies from 2007 to 2016.

Author(s)

Yang, J. and Peng, J.

Source

Data are publicly available at www.yahoo.com, and R package quantmod can be used to extract the stock information. The source code is in stockdata_source.

References

Yang, J. & Peng, J. (2018), 'Estimating Time-Varying Graphical Models', arXiv preprint arXiv:1804.03811

Examples

```
data(stockdata)

date.index <- rownames(stockdata$data)
stock.sector <- stockdata$info[, "sector"]

# select stocks from sector Materials and
# Telecommunications Services
sp.m <- t(stockdata$data[date.index < "2011-01-01",
stock.sector == "Materials"])
sp.t <- t(stockdata$data[date.index < "2011-01-01",
stock.sector == "Telecommunications Services"])
sp <- rbind(sp.m, sp.t)

# construct data matrix by taking log ratio of prices
# between adjacent time points
p <- dim(sp)[1]
N <- dim(sp)[2]-1
X <- matrix(0, p, N)
for(i in 1:p) {
  X[i, ] <- scale(log(sp[i, -1] / sp[i, -(N+1)]))
}
dim(X)  # dimension of data matrix

# positions of time points to estimate graphs
pos <- round(seq(0.1, 0.9, length=9)*(N-1)+1)
# estimate time-varying graphs and conduct model
# selection via cross-validation
result <- loggle.cv.h(X, pos, h = 0.1,
d.list = c(0, 0.02, 0.05),
lambda.list = 10 ^ c(-0.5, -0.3), cv.fold = 3,
fit.type = "pseudo", num.thread = 1)

# graphs at specified time points in selected model
library(igraph)
par(mfrow = c(1, 1))
pos.plot <- round(seq(2, length(pos)-1, length = 4))
for(k in 1:length(pos.plot)) {
  adj.matrix <-
```

```
  result$cv.select.result$adj.mat.opt[[pos.plot[k]]]
  net <- graph.adjacency(adj.matrix, mode = "undirected",
  diag = FALSE)
  set.seed(0)
  V(net)$color <- rep(rainbow(2), times = c(nrow(sp.m),
  nrow(sp.t)))
  sp.ind <- rep(1:2, times = c(nrow(sp.m), nrow(sp.t)))
  E(net)$color <- apply(get.edgelist(net), 1, function(x)
  ifelse(sp.ind[x[1]] == sp.ind[x[2]], "black", "gray"))
  E(net)$width <- apply(get.edgelist(net), 1, function(x)
  ifelse(sp.ind[x[1]] == sp.ind[x[2]], 1, 0.5))
  plot(net, vertex.size = 3, vertex.label = NA, layout =
  layout.circle, main = date.index[pos[pos.plot[k]]])
  legend("bottomright", c("Materials", "Tele. Services"),
  pch = 21, pt.bg = rainbow(2), cex = 0.6)
  legend("bottomleft", c("within sector",
  "cross sector"), lwd = c(1, 0.4), col = c("black",
  "gray"), cex = 0.6)
}
```

`stockdata_source` *Source code for stock prices of S&P 500 companies from 2007 to 2016*

Description

This is the source code for obtaining the stock closing prices and company information of S&P 500 companies in stockdata.

Author(s)

Yang, J. and Peng, J.

References

Yang, J. & Peng, J. (2018), 'Estimating Time-Varying Graphical Models', arXiv preprint arXiv:1804.03811

Examples

```
library(XML)
library(RCurl)
library(quantmod)

# select one stock of interest
stock.ticker <- "MSFT"

# extract stock information from wiki
url <- "https://en.wikipedia.org/wiki/List_of_S%26P_500_companies"
tabs <- getURL(url)
stock.info <- readHTMLTable(tabs, stringsAsFactors = FALSE)[[1]]
stock.ticker.all <- stock.info$`Ticker symbol`
```

```
stock.name.all <- stock.info$Security
stock.sector.all <- stock.info$`GICS Sector`
stock.index <- which(stock.ticker.all == stock.ticker)
stock.name <- stock.name.all[stock.index]
stock.sector <- stock.sector.all[stock.index]

# extract stock closing prices and indices of dates
stock.price <- getSymbols(stock.ticker, auto.assign=FALSE,
from='2007-01-01', to="2017-01-01")[, 4]
date.index <- as.character(index(stock.price))

# summary of selected stock
cat(sprintf("Ticker: %s\nName: %s\nSector: %s\nMin Price: %.2f (%s)
Max Price: %.2f (%s)", stock.ticker, stock.name, stock.sector,
min(stock.price), date.index[which.min(stock.price)],
max(stock.price), date.index[which.max(stock.price)]))
```

Index