Package ‘fusedest’

October 12, 2019

Type Package

Title Block Splitting Algorithm for Estimation with Fused Penalty Functions

Version 1.3.1

Date 2019-10-12

Author Tso-Jung Yen

Maintainer Tso-Jung Yen <tjyen@stat.sinica.edu.tw>

Description Provides methods fusedest_normal() and fusedest_logit() for carrying out block splitting algorithms for fused penalty estimation. For details, please see Tso-Jung Yen (2019) <doi.10618600.2019.1660178>.

License GPL-2

Encoding UTF-8

LazyData true

Imports Rcpp, methods, Matrix, parallel, igraph, stats

Depends

LinkingTo Rcpp, RcppEigen

SystemRequirements C++11

RoxygenNote 6.1.1

NeedsCompilation yes

Repository CRAN

Date/Publication 2019-10-12 09:20:02 UTC

R topics documented:

fusedest-package *Block splitting algorithms for estimation with fused penalty functions*

Description

It provides functions for carrying out block splitting algorithms for fused penalty estimation.

Details

Package:	fusedest
Type:	Package
Version:	1.1
Date:	2019-07-17
License:	GPL-2

Author(s)

Tso-Jung Yen

Maintainer: Tso-Jung Yen <tjyen@stat.sinica.edu.tw>

References

Yen, T.-J. (2019). Solving fused penalty estimation problems via block splitting algorithms.

fusedest_logit *The block splitting algorithm for logistic regression estimation with the fused group lasso penalty function*

Description

A function for computing logistic regression estimation with the fused group lasso penalty function

Value

Return a list of output, e.g. the solution, runtime and iteration error, for the block splitting algorithm. For more details, please see the example below.

Examples

```
library(fusedest)
library(igraph)

######### Functions for data generation ###########


generating_binary_data <- function(beta.true, N, m){


  #### internal functions #######

  logit.prob <- function(X,beta){
    p <- dim(X)[2]

    if(is.null(p)==TRUE){
      eta <- X*beta
    }
    if(is.null(p)==FALSE){
      eta <- X%*%beta
    }
    prob.a <- exp(eta)/(1+exp(eta))
    return(prob.a)
  }

  ################################

  p <- dim(beta.true)[2]
  M <- dim(beta.true)[1]

  label.list <- sample(c(1:M), m, replace = TRUE) ##### Label for data centers
  n.list <- rpois(m, N)
  n.list_pred <- n.list

  ind.strt <- c(1, cumsum(n.list[1:(m-1)])+1)
  ind.end <- cumsum(n.list)

  X <- cbind(rep(1, sum(n.list)), matrix(sample(c(0,1),
  sum(n.list)*(p-1), replace = TRUE, prob = c(0.5, 0.5)),
  nrow = sum(n.list), ncol = p-1))

  X_pred <- cbind(rep(1, sum(n.list)), matrix(sample(c(0,1),
  sum(n.list)*(p-1), replace = TRUE, prob = c(0.5, 0.5)),
  nrow = sum(n.list), ncol = p-1))


  label.dc <- rep(c(1:m), n.list)
  label.dc_pred <- rep(c(1:m), n.list_pred)

  y <- as.numeric(unlist(sapply(c(1:m),
                                function(i){
                                  rbinom(n.list[i], 1,
                                  logit.prob(X[ind.strt[i]:ind.end[i],],
                                  as.numeric(beta.true[label.list[i],])))
                                })))

  y_pred <- as.numeric(unlist(sapply(c(1:m),
                                function(i){
                                  rbinom(n.list[i], 1,
                                  logit.prob(X_pred[ind.strt[i]:ind.end[i],],
                                  as.numeric(beta.true[label.list[i],])))
                                })))

  label.true <- rep(label.list, n.list)
  label.true_pred <- rep(label.list, n.list_pred)

  results <- list(X, X_pred, y, y_pred, n.list, n.list_pred,
  label.dc, label.dc_pred, label.true, label.true_pred)

  names(results) <- c("X", "X_pred", "y", "y_pred", "n.list", "n.list_pred",
                      "label.dc", "label.dc_pred", "label.true", "label.true_pred")
  return(results)
}

generatingEdgelistID <- function(m){

  c1 <- rep(0,m*(m-1)/2)
  c2 <- rep(0,m*(m-1)/2)
  l <- 0
  for(i in 1:(m-1)){
    c1[c((l+1):(l+m-i))] <- rep(i,m-i)
    c2[c((l+1):(l+m-i))] <- c((i+1):m)
    l <- l + m-i
  }

  return(cbind(c1,c2))
}


Blockl2Norm <- function(beta_i, beta_j, p, q_H) {
    .Call('_fusedest_Blockl2Norm', PACKAGE = 'fusedest',
    beta_i, beta_j, p, q_H)
}


IRLSLogisticReg <- function(X, y, a, b, beta_ini, max_iter, tol_err) {
    .Call('_fusedest_IRLSLogisticReg', PACKAGE = 'fusedest',
    X, y, a, b, beta_ini, max_iter, tol_err)
}



########################################
```

```
beta.true <- t(matrix(
  c(c(1,1, rep(c(-0.1,0.1), 4)),
    c(-0.1,0.1, 1,1, rep(c(0.2,-0.2), 3)),
    c(rep(c(-0.1,0.1),2),c(1,1), rep(c(-0.1,0.1),2)),
    c(rep(c(-0.1,0.1),3),c(1,1),c(-0.1,0.1)),
    c(rep(c(-0.1,0.1),4),1, 1)), nrow = 10, ncol = 5
))

###### Setting parameters ##############

no_id <- 1
no.cores <- 1
N_list <-  100 #seq(100, 2000, length = 20)
id_list <- c(1:no_id)
m.total <- 10
p <- dim(beta.true)[2]
no_lambda <- 1

####### Number of data centers ########


result.AIC <- matrix(0, nrow = length(N_list)*no_id, ncol = 13)
result.BIC <- matrix(0, nrow = length(N_list)*no_id, ncol = 13)

l <- 1

for(u in 1:length(N_list)){

  N <- N_list[u]

  for(v in 1:no_id){


    id <- id_list[v]

######## Generating data #######################################

    mydata <- generating_binary_data(beta.true, N, m.total)

    y <- mydata$y
    X <- mydata$X
    label_dc <- mydata$label.dc
    label_true <- mydata$label.true
    n.list <- mydata$n.list

    y_pred <- mydata$y_pred
    X_pred <- mydata$X_pred
    label_dc_pred <- mydata$label.dc_pred
    label.true_pred <- mydata$label.true_pred
    n.list_pred <- mydata$n.list_pred

################## Setting parameters #########################
```

```
    set.seed(2, kind = NULL, normal.kind = NULL)

    rho <- 1
    H <- generatingEdgelistID(m = m.total)
    q_H <- sum(degree(graph_from_edgelist(H, directed = FALSE)))/2

    p <- dim(X)[2]
    n_dc <- as.numeric(unlist(table(label_dc)))
    m.total <- length(n_dc)
    label_true_dc <- tapply(label_true, label_dc, mean)
    beta_true_dc <- beta.true[label_true_dc,]
    n <- sum(n_dc)
    ind_strt <- c(1, cumsum(n_dc[1:(m.total-1)])+1)
    ind_end <- cumsum(n_dc)

################## Computing initial values ###################

    beta_ini <- t(parallel::mcmapply(function(i){

      ind_i <- c(ind_strt[i]:ind_end[i])
      IRLSLogisticReg(X = X[ind_i,], y = y[ind_i], a = 0, b = rep(0, p),
      beta_ini = rep(0, p), max_iter = 1000, tol_err = 10^(-8))$beta},
      c(1:m.total), mc.cores = no.cores))

    beta_i_list <- as.vector(t(beta_ini[H[,1],]))
    beta_j_list <- as.vector(t(beta_ini[H[,2],]))

   l2_norm_dist <- Blockl2Norm(beta_i = beta_i_list, beta_j = beta_j_list, p = p, q_H = q_H)

    max_lambda <- max(l2_norm_dist)
    #lambda_list <- seq(max_lambda, 0.01*max_lambda, length = no_lambda)
    lambda_list <- rev(as.numeric(quantile(l2_norm_dist,
    probs = seq(0.001, 1, length = no_lambda))))
    max_iter <- 10
    tol_err <- 5*10^(-3)

###### Run simulation #########################################

    strt.time <- Sys.time()

    result.uv <- fusedest_logit(X = X, y = y, label_dc = label_dc, H = H,
                         rho = rho, no_lambda = no_lambda, lambda_list = lambda_list,
                            beta_ini = beta_ini, max_iter = max_iter,
                            tol_err = tol_err, no.cores = no.cores)

    beta_list <- result.uv$beta_list
    alpha_list <- result.uv$alpha_list

  end.time <- Sys.time()

  print(difftime(end.time, strt.time, units = "sec"))
    }
}
```

fusedest_normal *The block splitting algorithm for linear regression estimation with the fused group lasso penalty function*

Description

A function for computing linear regression estimation with the fused group lasso penalty function

Value

Return a list of output, e.g. the solution, runtime and iteration error, for the block splitting algorithm. For more details, please see the example below.

Examples

```
library(fusedest)
library(igraph)

####### Functions for data generation #########


generating_normal_data <- function(beta.true, N, m, sigma2.y){

  p <- dim(beta.true)[2]
  M <- dim(beta.true)[1]

  label.list <- sample(c(1:M), m, replace = TRUE)
  n.list <- rpois(m, N)
  X <- matrix(rnorm(sum(n.list)*p, 0, 1), nrow = sum(n.list), ncol = p)

  ind.strt <- c(1, cumsum(n.list[1:(m-1)])+1)
  ind.end <- cumsum(n.list)
  label.dc <- rep(c(1:m), n.list)

  y <- as.numeric(unlist(sapply(c(1:m),
        function(i){
          X[ind.strt[i]:ind.end[i],]%*%as.numeric(beta.true[label.list[i],]) +
          rnorm(n.list[i], 0, sqrt(sigma2.y))
          })))

  label.true <- rep(label.list, n.list)

  results <- list(X, y, n.list, label.dc, label.true)
  names(results) <- c("X", "y", "n.list", "label.dc", "label.true")
  return(results)
}
```

```
generatingEdgelistID03 <- function(m, deg){

  c1 <- NULL
  c2 <- NULL

  if(deg < m-deg){

    c1 <- rep(0, (m-deg)*deg)
    c2 <- rep(0, (m-deg)*deg)

    for(i in 1:(m-deg)){

      ind.i <- c(((i-1)*deg+1):(i*deg))

      c1[ind.i] <- rep(i, deg)
      c2[ind.i] <- c((i+1):(i+deg))
    }

    if(deg > 1){
      c3 <- rep(0, deg*(deg-1)/2)
      c4 <- rep(0, deg*(deg-1)/2)
      l <- 0
      for(i in (m-deg+1):(m-1)){

        c3[c((l+1):(l+m-i))] <- rep(i, m-i)
        c4[c((l+1):(l+m-i))] <- c((i+1):m)
        l <- l + (m-i)
      }

    }
  }

  return(cbind(c(c1,c3),c(c2,c4)))
}



RcppInvGram <- function(X, w, lambda) {
    .Call('_fusedest_RcppInvGram', PACKAGE = 'fusedest', X, w, lambda)
}

RcppXtwy <- function(X, y, w) {
    .Call('_fusedest_RcppXtwy', PACKAGE = 'fusedest', X, y, w)
}

RcppWolsSolver03 <- function(invXtwX, Xtwy, b) {
    .Call('_fusedest_RcppWolsSolver03', PACKAGE = 'fusedest', invXtwX, Xtwy, b)
}



############ Setting true parameters ##########
```

```
p.star <- 10

beta.true <- t(matrix(
  c(rep(c(-2,2), p.star),
    rep(c(2,-2), p.star),
    c(rep(2, p.star),rep(-2,5)),
    c(rep(-2,p.star),rep(2,5)),
    rep(c(-1,3), p.star)), nrow = p.star, ncol = 5
))
N <- 100
m <- 10
p <- dim(beta.true)[2]

########## Generating data ###################

strt <- Sys.time()

mydata <- generating_normal_data(beta.true, N, m, sigma2.y = 1)

end <- Sys.time()
difftime(end, strt, units="sec")

y <- mydata$y
X <- mydata$X
label_dc <- mydata$label.dc
label.true <- mydata$label.true
n.list <- mydata$n.list

sum(n.list)
length(n.list)
length(y)
dim(X)
min(n.list)
max(n.list)
sum(n.list)

###### Run simulation ########################################

no.cores <- 1
m.total <- 10
m.list <- 10
ind.strt <- c(1, cumsum(n.list[1:(m.total-1)])+1)
ind.end <- cumsum(n.list)

no_lambda <- 1
lambda_list <- 0.01

u <- 1
H <- generatingEdgelistID03(m = m.list[u], deg = 2)
q_H <- sum(degree(graph_from_edgelist(H, directed = FALSE)))/2

max_iter <- 10
tol_err <- 10^(-100)
```

```
rho <- 1

set.seed(2, kind = NULL, normal.kind = NULL)

##### Computing initial values ####################################

beta_ini <- t(parallel::mcmapply(function(i){
  W <- rep(1, n.list[i])
  inv_XTX_i <- RcppInvGram(X[ind.strt[i]:ind.end[i],], W, 0)
  XTy_i <- RcppXtwy(X[ind.strt[i]:ind.end[i], ],y[ind.strt[i]:ind.end[i]],W)
  RcppWolsSolver03(inv_XTX_i, XTy_i, rep(0, p))
}, c(1:m.total), mc.cores = no.cores))

beta_ini_norm <- sqrt(apply(beta_ini^2, 1, sum))

####### Running the proposed method ##########################

result.uv <- fusedest_normal(X = X[ind.strt[1]:ind.end[m.list[u]],],
                             y = y[ind.strt[1]:ind.end[m.list[u]]],
                             label_dc = label_dc[ind.strt[1]:ind.end[m.list[u]]], H = H,
                            rho = rho, no_lambda = no_lambda, lambda_list = lambda_list,
                             beta_ini = beta_ini[1:m.list[u],], max_iter = max_iter,
                             tol_err = tol_err, no.cores = no.cores)


result.BS <- result.uv$alg.matrix
```

Index