dcalasso

March 3, 2019

Type Package

Title Divide-and-conquer adaptive lasso for big data

Version 0.1.0

Author Yan Wang <yaw719@mail.harvard.edu>,
Tianxi Cai <tcai@hsph.harvard.edu>,
Chuan Hong <Chuan_Hong@hms.harvard.edu>

Maintainer Yan Wang <yaw719@mail.harvard.edu>

Description divideconquer package reduces the computational burden of fitting large adaptive lasso for Cox model when n>>p, by divide and conquer, least square approximation, and one-step estimation.

License

Depends survival, mgcv, glmnet, doParallel, foreach, MASS

Encoding UTF-8

LazyData true

RoxygenNote 6.1.1

R topics documented:

coef.dcalasso *Extract coefficients from dcalasso objects*

Description

`coef.dcalasso` extracts coefficients from dcalasso objects

Usage

```
## S3 method for class 'dcalasso'
coef(object, unpen = F, ...)
```

Arguments

`object`	a dcalasso object
`unpen`	whether to switch to the unpenalized coefficients

Author(s)

Yan Wang, Tianxi Cai, Chuan Hong

dataprocess *Process data frame*

Description

`dataprocess` processes the original dataset into useful elements: Y, X, strata, weights, offset.

Usage

```
dataprocess(mf, Terms, family)
```

Author(s)

Yan Wang, Tianxi Cai, Chuan Hong

dcalasso — *Divide-and-conquer method for the fitting of adaptive lasso model with big data*

Description

`dcalasso` fits adaptive lasso for big datasets using multiple linearization methods, including one-step estimation and least square approximation. This function is able to fit the adaptive lasso model either when the dataset is being loaded as a whole into `data` or when the datasets are splitted a priori and saved into multiple `rds` files. The algorithm uses a divide-and-conquer one-step estimator as the initial estimator and uses a least square approximation to the partial likelihood, which reduces the computation cost. The algorithm currently supports adaptive lasso with Cox proportional hazards model with or without time-dependent covariates. Ties in survival data analysis are handled by Efron's method. The first half of the routine computes an initial estimator (n^1/2 consistent estimator). It first obtains a warm-start by fitting coxph to the first subset (first random split of data or first data file indicated by data.rds) and then uses one-step estimation with iter.os rounds to update the warm-start. The one-step estimation loops through each subset and gathering scores and information matrices. The second half of the routine then shrinks the initial estimator using a least square approximation-based adaptive lasso step.

Usage

```
dcalasso(formula, family = cox.ph(), data = NULL, data.rds = NULL,
  weights, subsets, na.action, offset, lambda = 10^seq(-10, 3, 0.01),
  gamma = 1, K = 20, iter.os = 2, ncores = 1)
```

Arguments

`formula`	a formula specifying the model. For Cox model, the outcome should be specified as the Surv(start, stop, status) or Surv(start, status) object in the survival package.
`family`	For Cox model, family should be cox.ph(), or "cox.ph".
`data`	data frame containing all variables.
`data.rds`	when the dataset is too big to load as a whole into the RAM, one can specify `data.rds` which are the full paths of all randomly splitted subsets of the full data, saved into multiple `.rds` format.
`weights`	a prior weights on each observation
`na.action`	how to handle NA
`offset`	an offset term with a fixed coefficient of one
`lambda`	tuning parameter for the adaptive lasso penalty. penalty = lambda * sum_j |beta_j|/|beta_j initial|^gamma
`gamma`	exponent of the adaptive penalty. penalty = lambda * sum_j |beta_j|/|beta_j initial|^gamma
`K`	number of division of the full dataset. It will be overwritten to `length(data.rds)` if data.rds is given.
`iter.os`	number of iterations for one-step updates
`ncores`	number of cores to use. The iterations will be paralleled using `foreach` if ncores>1.
`subset`	an expression indicating subset of rows of data used in model fitting

Value

`coefficients.pen`
: adaptive lasso shrinkage estimation

`coefficients.unpen`
: initial unregularized estimator

`cov.unpen`	variance-covariance matrix of unpenalized model
`cov.pen`	variance-covariance matrix of penalized model
`BIC`	sequence of BIC evaluation at each lambda
`n.pen`	number use to penalize the degrees of freedom in BIC.
`n`	number of used rows of the data
`idx.opt`	index for the optimal BIC
`BIC.opt`	minimal BIC
`family`	family object of the model
`lamba.opt`	optimal lambda to minimize BIC
`df`	degrees of freedom at each lambda
`p`	number of covariates
`iter`	number of one-step iterations
`Terms`	term object of the model

Author(s)

Yan Wang <yaw719@mail.harvard.edu>, Tianxi Cai <tcai@hsph.harvard.edu>, Chuan Hong <Chuan_Hong@hms.harvard.edu>

References

Wang, Yan, Chuan Hong, Nathan Palmer, Qian Di, Joel Schwartz, Isaac Kohane, and Tianxi Cai. "A Fast Divide-and-Conquer Sparse Cox Regression." arXiv preprint arXiv:1804.00735 (2018).

Examples

```
##### Time-independent #####
set.seed(1)
N = 1e5; p.x = 50; K = 100; n = N/K;  cor = 0.2;
bb = c(rep(0.4,4),rep(0.2,4),rep(0.1,4),rep(0.05,4))
beta0 = c(1, bb, rep(0, p.x - length(bb)))
dat.mat0 = as.data.frame(SIM.FUN(N, p.x = p.x, cor = cor, family='Cox',beta0 = beta0))
dat.mat0[,'strat'] = rep(1:20, each = N/20)

## Without strata
# unicore
mod = dcalasso(as.formula(paste0('Surv(u,delta)~',paste(paste0('V',3:52),collapse='+'))),
               family = 'cox.ph',data = dat.mat0,
               K = 10, iter.os = 2)
sum.mod = summary(mod)
print(sum.mod, unpen = T)
plot(mod)
pred.link = predict(mod, newdata = dat.mat0)
pred.term = predict(mod, newdata = dat.mat0, type = 'terms')
pred.response = predict(mod, newdata = dat.mat0, type = 'response')
```

```
# parallel
modp = dcalasso(as.formula(paste0('Surv(u,delta)~',paste(paste0('V',3:52),collapse='+'))),
                family = 'cox.ph',data = dat.mat0,
                K = 10, iter.os = 4, ncores = 2)
sum.modp = summary(modp)
print(sum.modp, unpen = T)
plot(modp)

# Standard
std = coxph(as.formula(paste0('Surv(u,delta)~',paste(paste0('V',3:52),collapse='+'))),
            data = dat.mat0)

plot(mod$coefficients.unpen, std$coefficients)
plot(modp$coefficients.unpen, std$coefficients)


## With strata
# unicore
mod = dcalasso(as.formula(paste0('Surv(u,delta)~strata(strat)+',paste(paste0('V',3:52),collapse='+'))),
               family = 'cox.ph',data = dat.mat0,
               K = 10, iter.os = 2)
sum.mod = summary(mod)
print(sum.mod, unpen = T)
plot(mod)

# parallel
modp = dcalasso(as.formula(paste0('Surv(u,delta)~strata(strat)+',paste(paste0('V',3:52),collapse='+'))),
                family = 'cox.ph',data = dat.mat0,
                K = 10, iter.os = 2, ncores = 2)
sum.modp = summary(modp)
print(sum.modp, unpen = T)
plot(modp)

# Standard
std = coxph(as.formula(paste0('Surv(u,delta)~strata(strat)+',paste(paste0('V',3:52),collapse='+'))),
            data = dat.mat0)


plot(mod$coefficients.unpen, std$coefficients)
plot(modp$coefficients.unpen, std$coefficients)




##### Time-independent with separate file saving #####
set.seed(1)
N = 1e5; p.x = 50; K = 100; n = N/K;  cor = 0.2;
bb = c(rep(0.4,4),rep(0.2,4),rep(0.1,4),rep(0.05,4))
beta0 = c(1, bb, rep(0, p.x - length(bb)))
dat.mat0 = as.data.frame(SIM.FUN(N, p.x = p.x, cor = cor, family='Cox',beta0 = beta0))
dat.mat0[,'strat'] = rep(1:20, each = N/20)
dir = "C:/"
ll = split(1:N, factor(1:10))
for (kk in 1: 10){
  df = dat.mat0[ll[[kk]],]
  saveRDS(df, file = paste0(dir,'dataTI',kk,'.rds'))
```

```
}

## Without strata
# unicore
mod = dcalasso(as.formula(paste0('Surv(u,delta)~',paste(paste0('V',3:52),collapse='+'))),
               family = 'cox.ph',
               data.rds = paste0(dir,'dataTI',1:10,'.rds'), iter.os = 2)
sum.mod = summary(mod)
print(sum.mod, unpen = T)
plot(mod)

# parallel
modp = dcalasso(as.formula(paste0('Surv(u,delta)~',paste(paste0('V',3:52),collapse='+'))),
                family = 'cox.ph',
                data.rds = paste0(dir,'dataTI',1:10,'.rds'), iter.os = 2, ncores = 2)
sum.modp = summary(modp)
print(sum.modp, unpen = T)
plot(modp)

# Standard
std = coxph(as.formula(paste0('Surv(u,delta)~',paste(paste0('V',3:52),collapse='+'))),
            data = dat.mat0)

plot(mod$coefficients.unpen, std$coefficients)
plot(modp$coefficients.unpen, std$coefficients)


## With strata
# unicore
mod = dcalasso(as.formula(paste0('Surv(u,delta)~strata(strat)+',paste(paste0('V',3:52),collapse='+'))),
               family = 'cox.ph',
               data.rds = paste0(dir,'dataTI',1:10,'.rds'), K = 10, iter.os = 2)
sum.mod = summary(mod)
print(sum.mod, unpen = T)
plot(mod)

# parallel
modp = dcalasso(as.formula(paste0('Surv(u,delta)~strata(strat)+',paste(paste0('V',3:52),collapse='+'))),
                family = 'cox.ph',
             data.rds = paste0(dir,'dataTI',1:10,'.rds'), K = 10, iter.os = 2, ncores = 2)
sum.modp = summary(modp)
print(sum.modp, unpen = T)
plot(modp)

# Standard
std = coxph(as.formula(paste0('Surv(u,delta)~strata(strat)+',paste(paste0('V',3:52),collapse='+'))),
            data = dat.mat0)

plot(mod$coefficients.unpen, std$coefficients)
plot(modp$coefficients.unpen, std$coefficients)


########### Time-dependent loading as a whole ####################
set.seed(1)
n.subject = 1e5; p.ti = 50; p.tv = 50; K = 20; n = n.subject/K; cor = 0.2;  lambda.grid = 10^seq(-10,3,0.01);
beta0.ti = NULL
beta0.tv = NULL
```

```
dat.mat0 = as.data.frame(SIM.FUN.TVC(p.ti, p.tv, n.subject, cor, beta0.ti, beta0.tv))
dat.mat0[,'strat'] = dat.mat0[,dim(dat.mat0)[2]]%%(n.subject/20)
dat.mat0 = dat.mat0[,-(dim(dat.mat0)[2]-1)]

## Without strata
# unicore
mod = dcalasso(as.formula(paste0('Surv(t0,t1,status)~',paste(paste0('V',4:103),collapse='+'))),
               family = 'cox.ph',data = dat.mat0,
               K = 10, iter.os = 2)
sum.mod = summary(mod)
print(sum.mod, unpen = T)
plot(mod)

# parallel
modp = dcalasso(as.formula(paste0('Surv(t0,t1,status)~',paste(paste0('V',4:103),collapse='+'))),
                family = 'cox.ph',data = dat.mat0,
                K = 10, iter.os = 2, ncores = 2)
sum.modp = summary(modp)
print(sum.modp, unpen = T)
plot(modp)

# Standard
std = coxph(as.formula(paste0('Surv(t0,t1,status)~',paste(paste0('V',4:103),
                                                          collapse='+'))),
            data = dat.mat0)
plot(mod$coefficients.unpen, std$coefficients)
plot(modp$coefficients.unpen, mod$coefficients.unpen)

# With strata
# unicore
mod = dcalasso(as.formula(paste0('Surv(t0,t1,status)~strata(strat)+',paste(paste0('V',4:103),collapse='+'))
               family = 'cox.ph',data = dat.mat0,
               K = 10, iter.os = 4)
sum.mod = summary(mod)
print(sum.mod, unpen = T)
plot(mod)

# parallel
modp = dcalasso(as.formula(paste0('Surv(t0,t1,status)~strata(strat)+',paste(paste0('V',4:103),collapse='+')
                family = 'cox.ph',data = dat.mat0,
                K = 10, iter.os = 4, ncores = 2)
sum.modp = summary(modp)
print(sum.modp, unpen = T)
plot(modp)

# Standard
std = coxph(as.formula(paste0('Surv(t0,t1,status)~strata(strat)+',paste(paste0('V',4:103),
                                                                        collapse='+'))),
            data = dat.mat0)
plot(mod$coefficients.unpen, std$coefficients)
plot(modp$coefficients.unpen, mod$coefficients.unpen)



########### Time-dependent separate file saving ###################
set.seed(1)
```

```
n.subject = 1e5; p.ti = 50; p.tv = 50; K = 20; n = n.subject/K; cor = 0.2;  lambda.grid = 10^seq(-10,3,0.01);
beta0.ti = NULL
beta0.tv = NULL
dat.mat0 = as.data.frame(SIM.FUN.TVC(p.ti, p.tv, n.subject, cor, beta0.ti, beta0.tv))
dat.mat0[,'strat'] = dat.mat0[,dim(dat.mat0)[2]]%%(n.subject/20)
dat.mat0 = dat.mat0[,-(dim(dat.mat0)[2]-1)]
ll = split(1:dim(dat.mat0)[1], factor(1:10))
for (kk in 1: 10){
  df = dat.mat0[ll[[kk]],]
  saveRDS(df, file = paste0(dir,'dataTV',kk,'.rds'))
}


## Without strata
# unicore
mod = dcalasso(as.formula(paste0('Surv(t0,t1,status)~',paste(paste0('V',4:103),collapse='+'))),
               family = 'cox.ph',
               data.rds = paste0(dir,'dataTV',1:10,'.rds'), K = 10, iter.os = 2)
sum.mod = summary(mod)
print(sum.mod, unpen = T)
plot(mod)

# parallel
modp = dcalasso(as.formula(paste0('Surv(t0,t1,status)~',paste(paste0('V',4:103),collapse='+'))),
                family = 'cox.ph',
             data.rds = paste0(dir,'dataTV',1:10,'.rds'), K = 10, iter.os = 2, ncores = 2)
sum.modp = summary(modp)
print(sum.modp, unpen = T)
plot(modp)

# Standard
std = coxph(as.formula(paste0('Surv(t0,t1,status)~',paste(paste0('V',4:103),
                                                          collapse='+'))),
            data = dat.mat0)
plot(mod$coefficients.unpen, std$coefficients)
plot(modp$coefficients.unpen, mod$coefficients.unpen)

# With strata
# unicore
mod = dcalasso(as.formula(paste0('Surv(t0,t1,status)~strata(strat)+',paste(paste0('V',4:103),collapse='+')))
               family = 'cox.ph',
               data.rds = paste0(dir,'dataTV',1:10,'.rds'), K = 10, iter.os = 4)
sum.mod = summary(mod)
print(sum.mod, unpen = T)
plot(mod)

# parallel
modp = dcalasso(as.formula(paste0('Surv(t0,t1,status)~strata(strat)+',paste(paste0('V',4:103),collapse='+')
                family = 'cox.ph',
             data.rds = paste0(dir,'dataTV',1:10,'.rds'), K = 10, iter.os = 4, ncores = 2)
sum.modp = summary(modp)
print(sum.modp, unpen = T)
plot(modp)

# Standard
std = coxph(as.formula(paste0('Surv(t0,t1,status)~strata(strat)+',paste(paste0('V',4:103),
                                                                        collapse='+'))),
```

```
            data = dat.mat0)
plot(mod$coefficients.unpen, std$coefficients)
plot(modp$coefficients.unpen, mod$coefficients.unpen)
```

`plot.dcalasso` *Plot BIC paths for dcalasso objects*

Description

`plot.dcalasso` summarizes output of dcalasso fit

Usage

```
## S3 method for class 'dcalasso'
plot(object, ...)
```

Arguments

`object`	a dcalasso object

Author(s)

Yan Wang, Tianxi Cai, Chuan Hong

`predict.dcalasso` *Prediction of dcalasso object*

Description

`predict.dcalasso` makes prediction of a dcalasso object based on the adaptive lasso estimation.

Usage

```
## S3 method for class 'dcalasso'
predict(object, newdata, type = "link")
```

Arguments

`object`	a dcalasso object
`newdata`	a new data frame
`type`	"terms", "link", "response" same as predict.glm

Author(s)

Yan Wang, Tianxi Cai, Chuan Hong

print.dcalasso *Print dcalasso objects*

Description

`print.dcalasso` summarizes output of dcalasso fit

Usage

```
## S3 method for class 'dcalasso'
print(object, ...)
```

Arguments

`object`	a dcalasso object

Author(s)

Yan Wang, Tianxi Cai, Chuan Hong

print.summary.dcalasso *Print summary for dcalasso objects*

Description

`print.summary.dcalasso` summarizes output of dcalasso fit

Usage

```
## S3 method for class 'summary.dcalasso'
print(object, unpen = F, ...)
```

Arguments

`object`	a summary.dcalasso object
`unpen`	whether to print out the unpenalized result

Author(s)

Yan Wang, Tianxi Cai, Chuan Hong

scoreinfofun — *Computes score and information given subset*

Description

`scoreinfofun` computes information and scores.

Usage

```
scoreinfofun(Y, X, stratas, weights, offset, family, bini, idx)
```

Arguments

`Y`	outcome vector or matrix
`X`	design matrix
`stratas`	strata vector
`weights`	weight for each observation
`offset`	offset for each observation
`family`	family of the outcome
`bini`	initial coefficient estimate
`idx`	indices for evaluation

Author(s)

Yan Wang, Tianxi Cai, Chuan Hong

SIM.FUN — *Generate simulation data to test adaptive lasso*

Description

`SIM.FUN` generates continuous-time survival response data that are associated with design matrix. The design matrix comes from a correlated multivariate normal. The default signals (beta0) are sparse.

Usage

```
SIM.FUN(nn, p.x = 50, cor = 0.2, family = c("binary", "count",
  "Cox"), beta0 = NULL)
```

Arguments

`nn`	sample size
`p.x`	number of covariates
`cor`	correlation of covariates
`family`	the family of response data taking c('binary','count','Cox')
`beta0`	the coefficients for the design, including intercept

Value

For survival data, it returns a matrix with the first column U, second column delta (0,1), and rest = design matrix.

Author(s)

Yan Wang, Tianxi, Chuan Hong

Examples

```
SIM.FUN(nn = 1e6, p.x = 50, family = 'binary')
```

SIM.FUN.TVC *Generate simulation data to test time-dependent Cox model with adaptive lasso*

Description

SIM.FUN.TVC generates time-dependent survival response with four time-intervals 0-1, 1-2, 2-3, 3-4 for each subject data. All subjects are administratively censored at 4, if T>4. T comes from a Weibull distribution with shape of 2. The design matrix comes from a correlated multivariate normal. The default signals (beta0) are sparse.

Usage

```
SIM.FUN.TVC(p.ti = 50, p.tv = 50, n.subject = 1e+06, cor = 0.2,
  beta0.ti = NULL, beta0.tv = NULL)
```

Arguments

p.ti	number of time-invariant covariates
p.tv	number of time-varying covariates
n.subject	number of subjects
cor	correlation between time-varying and each interval's time-varying covairates
beta.ti	coefficient for time-invariant covariates
beta.tv	coefficients for time-varying covariates

Value

a matrix with the first column starting time, second column ending time, third column event (0,1), and rest = design matrix + ID for subject.

Author(s)

Yan Wang, Tianxi Cai, Chuan Hong

References

Section 3.3 in Austin, P.C., 2012. Generating survival times to simulate Cox proportional hazards models with time-varying covariates. Statistics in medicine, 31(29), pp.3946-3958.

Examples

```
SIM.FUN.tvc()
```

`summary.dcalasso` *Summary method for dcalasso objects*

Description

`summary.dcalasso` summarizes output of dcalasso fit

Usage

```
## S3 method for class 'dcalasso'
summary(object, unpen = F, ...)
```

Arguments

`object`	a dcalasso fit
`unpen`	whether to print out the unpenalized result

Author(s)

Yan Wang, Tianxi Cai, Chuan Hong

`vcov.dcalasso` *Extract variance covariance from a dcalasso objects*

Description

`vcov.dcalasso` extracts variance covariance objects

Usage

```
## S3 method for class 'dcalasso'
vcov(object, unpen = F, ...)
```

Arguments

`object`	a dcalasso object
`unpen`	whether to switch to the unpenalized variance covariance

Author(s)

Yan Wang, Tianxi Cai, Chuan Hong

`warmfit` *Warm-start fit wrapper*

Description

`warmfit` is to obtain an warm-start of the initial estimator using a subset of data.

Usage

```
warmfit(Y, X, strata, weights, offset, family, idx)
```

Arguments

`Y`	outcome vector or matrix
`X`	design matrix
`strata`	strata vector
`weights`	weight for each observation
`offset`	offset for each observation
`family`	family of the outcome
`idx`	indices for the subset to fit

Author(s)

Yan Wang, Tianxi Cai, Chuan Hong

Index