Package ‘divideconquer’

May 18, 2017

Type Package

Title Divide-and-conquer adaptive lasso using least square approximation

Version 0.1.0

Author Yan Wang, Tianxi Cai

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

Description divideconquer package reduces the computational burden for fitting adaptive lasso when n>>p, through the combinatorial use of divide and conquer, least square approximation, and one-step estimator.

License

Encoding UTF-8

LazyData true

RoxygenNote 6.0.1

R topics documented:

`divideconquer-package` *Divide-and-Conquer adaptive lasso with least square approximation*

Description

Package:	divideconquer
Type:	Package
Version:	0.1
Date:	2017-05-18
License:	GPL (>= 2)
LazyLoad:	yes

Applies divide-and-conquer adaptive lasso based on least square approximation

Usage

```
loadpack()
```

Author(s)

Yan Wang `<yaw719@mail.harvard.edu>`, Tianxi Cai `<tcai@hsph.harvard.edu>`

References

the paper

`A.fun` *Calculate negative information (A) for logistic regression.*

Description

`A.fun` evaluates negative information (A) for logistic regression.

Usage

```
A.fun(bet, dat)
```

Arguments

`bet`	is the beta for evaluation
`dat`	is the dataset

cumsum2 *Cumulative sum*

Description

`cumsum2` calculates cumulative sum

Usage

```
cumsum2(mydat)
```

dg.logit *Inner function for one-step estimator*

Description

`g.logit` evaluates the derivative of logit function.

Usage

```
dg.logit(xx)
```

Est.ALASSO.Approx.GLMNET *Least square approximated adaptive lasso*

Description

`Est.ALASSO.Approx.GLMNET` fits adaptive lasso based on least square approximation (LSA)

Usage

```
Est.ALASSO.Approx.GLMNET(ynew, xnew, bini, N.adj, lambda.grid)
```

Arguments

`ynew`	a p by 1 vector of LSA-based outcome
`xnew`	a p by p matrix of LSA-based covariates
`bini`	1/bini is used as the penalty for adaptive lasso
`N.adj`	log(N.adj) is used to penalize BIC, where BIC = -2loglik + df*log(N.adj), and modified BIC, where modBIC = -2loglik + df*N.adj^0.1
`lambda.grid`	the grid of lambda to put into glmnet

Value

bhat.BIC adaptive lasso estimator using BIC and bhat.modBIC adaptive lasso estimator using modified BIC. lambda.BIC and lambda.modBIC return the optimal lambda chosen by BIC and modified BIC.

Author(s)

Yan Wang, Tianxi Cai

Examples

```
Est.ALASSO.Approx.GLMNET(ynew,xnew,bini,N.adj)
```

`Est.ALASSO.GLMNET` *A wrapper for adaptive lasso*

Description

`Est.ALASSO.GLMNET` fits adaptive lasso based on glmnet. The best lambda (penalizing factor) is chosen by BIC or modified BIC.

Usage

```
Est.ALASSO.GLMNET(data, BIC.factor = 0.1, fam0 = "binomial", w.b = NULL,
  lambda.grid)
```

Arguments

`data`	matrix or data frame. For non-survival outcome, first column = response, rest = design matrix. For continuous-time survival outcome, first column = time, second column = delta (0/1), rest = design matrix.
`BIC.factor`	factor in modified BIC, BIC = -2 loglikelihood + df * N^BIC.factor
`fam0`	family of the response, taking "binomial", "Poisson", "Cox"
`w.b`	w.b used to penalize adaptive lasso. If null, a glm/Cox model will be fitted and 1/abs(coefficients) will be used as w.b
`lambda.grid`	the grid of lambda to put into glmnet

Value

bhat.BIC adaptive lasso estimator using BIC and bhat.modBIC adaptive lasso estimator using modified BIC. lambda.BIC and lambda.modBIC return the optimal lambda chosen by BIC and modified BIC.

Author(s)

Yan Wang, Tianxi Cai

Examples

```
Est.ALASSO.Approx.GLMNET(ynew,xnew,bini,N.adj)
```

`Est.ALASSO.GLMNET.CV` *A modified wrapper for adaptive lasso with the optimal penalty chosen by cross-validation*

Description

`Est.ALASSO.GLMNET.CV` fits adaptive lasso based on cv.glmnet. The best lambda (penalizing factor) is chosen by 10-fold cross-validation.

Usage

```
Est.ALASSO.GLMNET.CV(data, fam0 = "binomial", w.b = NULL, lambda.grid,
  chunksize = 50)
```

Arguments

`data`	matrix or data frame. For non-survival outcome, first column = response, rest = design matrix. For continuous-time survival outcome, first column = time, second column = delta (0/1), rest = design matrix.
`fam0`	family of the response, taking "binomial", "Poisson", "Cox"
`w.b`	w.b used to penalize adaptive lasso. If null, a glm/Cox model will be fitted and 1/abs(coefficients) will be used as w.b
`lambda.grid`	the grid of lambda to put into glmnet
`chunksize`	The prediction step in cv.glmnet could take a large amount of memory. chunksize specifies how many chunks you would like to split the prediction step. The predition step will be run in a loop if chunksize>1.

Value

a list containing two arguments: bhat.cv adaptive lasso estimator using 10-fold cross-validation; lambda.cv is the optimal lambda chosen by cross-validation.

Author(s)

Yan Wang, Tianxi Cai

Examples

```
Est.ALASSO.GLMNET.CV(data, fam0="binomial", w.b = NULL, lambda.grid, chunksize = 50)
```

`Est.ALASSO.GLMNET.TANGXIE`
A wrapper for divide-and-conquer adaptive lasso proposed by Chen and Xie (2014) and Tang et al. (2016).

Description

`Est.ALASSO.GLMNET.TANGXIE` fits adaptive lasso based on cv.glmnet. The best lambda (penalizing factor) is chosen by 10-fold cross-validation.

Usage

```
Est.ALASSO.GLMNET.TANGXIE(dat.list, K, BIC.factor = 0.1, fam0 = "binomial",
  lambda.grid, mvpct = 0.5)
```

Arguments

`dat.list`	a list of matrices. Each element of the list is a sub-dataset. In each sub-dataset, for non-survival outcome, first column = response, rest = design matrix. In each sub-dataset, for continuous-time survival outcome, first column = time, second column = delta (0/1), rest = design matrix.
`K`	Number of sub-datasets in dat.list
`BIC.factor`	factor in modified BIC, BIC = -2 loglikelihood + df * N^BIC.factor
`fam0`	family of the response, taking "binomial", "Poisson", "Cox"
`lambda.grid`	the grid of lambda to put into glmnet
`mvpct`	majority voting percentage used for Chen and Xie (2014)
`w.b`	w.b used to penalize adaptive lasso. If null, a glm/Cox model will be fitted and 1/abs(coefficients) will be used as w.b

Value

a list containing two arguments: bhat.cv adaptive lasso estimator using 10-fold cross-validation; lambda.cv is the optimal lambda chosen by cross-validation.

Author(s)

Yan Wang, Tianxi Cai

References

Chen, Xueying, and Min-ge Xie. "A split-and-conquer approach for analysis of extraordinarily large data." Statistica Sinica (2014): 1655-1684.

Tang, Lu, Ling Zhou, and Peter X-K. Song. "Method of Divide-and-Combine in Regularised Generalised Linear Models for Big Data." arXiv preprint arXiv:1611.06208 (2016).

Examples

```
Est.ALASSO.GLMNET.TANGXIE(dat.list,K,BIC.factor=0.1,fam0="binomial",lambda.grid,mvpct = 0.5)
```

g.logit — *Inner function for one-step estimator*

Description

`g.logit` evaluates logit function.

Usage

```
g.logit(xx)
```

iteration.fun — *One-step iteration function for divide-and-conquer logistic model*

Description

`iteration.fun` estimates a one-step for a logistic regression for each subset.

Usage

```
iteration.fun(dat.list, bini, kk.list)
```

Arguments

`dat.list`	list of subsets (after dividing). In each subset, first column = outcome, rest = design matrix
`bini`	initial estimator as starting point
`kk.list`	which subsets of dat.list get one-step update

Value

a list with b.k a matrix of one step estimator and Ahat the negative information matrix

Author(s)

Yan Wang, Tianxi Cai

Examples

```
iteration.fun(dat.list=dat.list,bini=bini,kk.list=2:K)
```

`iteration.fun.cox` *One-step iteration function for divide-and-conquer Cox model*

Description

`iteration.fun.cox` estimates a one-step for a Cox regression for each subset.

Usage

```
iteration.fun.cox(dat.list, bini, kk.list, rtn = "Score+A+Approx")
```

Arguments

`dat.list`	list of subsets (after dividing). In each subset, first column = U, second column = delta (0/1), rest = design matrix
`bini`	initial estimator as starting point
`kk.list`	which subsets of dat.list get one-step update
`rtn`	return score, score+negative information, score+negative information using -SS' approximation

Value

a list with b.k a matrix of one-step estimators and with Ahat the negative information matrix

Author(s)

Yan Wang, Tianxi Cai

Examples

```
iteration.fun.cox(dat.list=dat.list,bini=bini,kk.list=2:K,rtn='Score+A+Approx')
```

`logitlik.fun` *Calculate log-likelihood for logistic regression.*

Description

`logitlik.fun` evaluates log-likelihood for logistic regression.

Usage

```
logitlik.fun(bet.mat, dat)
```

Arguments

`bet.mat`	is the beta matrix for evaluation
`dat`	is the dataset

`mycv.coxnet` *Inner function for Est.ALASSO.GLMNET.CV*

Description

`mycv.coxnet` is a modified function of cv.coxnet, such that the prediction step can be split and run.

Usage

```
mycv.coxnet(outlist, lambda, x, y, weights, offset, foldid, type.measure,
  grouped, keep = FALSE)
```

`mycv.glmnet` *Inner function for Est.ALASSO.GLMNET.CV*

Description

`mycv.glmnet` is a modified function of cv.glmnet, such that the prediction step can be split and run.

Usage

```
mycv.glmnet(x, y, weights, offset = NULL, lambda = NULL,
  type.measure = c("mse", "deviance", "class", "auc", "mae"), nfolds = 10,
  foldid, grouped = TRUE, keep = FALSE, parallel = FALSE, ...)
```

`mycv.lognet` *Inner function for Est.ALASSO.GLMNET.CV*

Description

`mycv.lognet` is a modified function of cv.lognet, such that the prediction step can be split and run.

Usage

```
mycv.lognet(outlist, lambda, x, y, weights, offset, foldid, type.measure,
  grouped, keep = FALSE)
```

`MySum` *Inner function for one-step estimator*

Description

`MySum` is an inner function to calculate score and negative information.

Usage

```
MySum(yy, FUN, Yi, Vi = NULL)
```

PI.k.FUN — *Inner function for one-step estimator*

Description

`Score.A.FUN` is an inner function to calculate score and negative information.

Usage

```
PI.k.FUN(tt, ebzi, xi, zi, k0 = 0, vi = NULL)
```

Score.A.FUN — *Calculate score and negative information (A) matrix for a Cox model*

Description

`Score.A.FUN` evaluates score and negative information (A)

Usage

```
Score.A.FUN(data, betahat, rtn = "score")
```

Arguments

`data`	a matrix with first column = U, second column = delta (0/1), rest = design
`betahat`	at which coefficient level to evaluate score and negative information
`rtn`	'Score' returns only the score function, 'Score+A' returns score and negative information (second derivative of PL), 'Score+A+Approx' returns score and negative information (approximated by -SS')

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

Description

`SIM.FUN` generates binary, count, and 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 binary and count data, it returns a matrix with the first column=response, rest = design matrix, without intercept. 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 Cai

Examples

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

U.fun *Calculate score for logistic regression.*

Description

U.fun evaluates scores for logistic regression.

Usage

```
U.fun(bet, dat)
```

Arguments

bet	is the beta for evaluation
dat	is the dataset

VTM *Vector to matrix*

Description

VTM Replicate vector vc by dm times and create a dm by length(vc) matrix

Usage

```
VTM(vc, dm)
```

Index