Package ‘FastLORS’

January 4, 2019

Type Package

Title Joint Modeling for eQTL Mapping in R

Version 0.1.0

Author Jacob Rhyne, Jessie Jeng, and Eric Chi

Maintainer Jacob Rhyne <`jdrhyne2@ncsu.edu`>

Description This package applies FastLORS to perform eQTL mapping for gene expression and SNP data. It can also be used to apply the LORS method of Yang et al. (2013). The package also contains two pre-screening methods to reduce the number of SNPs before joint modeling: (1) HC-Screening: a method that selects the top SNPs based on their higher criticism statistics (Rhyne et al. 2018) and (2) LORS-Screening: which fits a marginal estimate and selects the top SNPs per each gene.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 6.0.1

Imports glmnet,
MASS,
utils,
methods,
stats

Suggests knitr,
rmarkdown

VignetteBuilder knitr

R topics documented:

`Fast_LORS` *Fast_LORS*

Description

`Fast_LORS` is a function for solving the LORS optimization problem in Can Yang et al. (2013) through the proximal gradient method

Usage

```
Fast_LORS(Y, X, rho, lambda, maxiter = 5000, eps = 2.2204e-16,
  tol = 1e-04, verbose = FALSE)
```

Arguments

`Y`	gene expression matrix
`X`	matrix of SNPs
`rho`	parameter for enforcing sparsity of coefficient matrix
`lambda`	parameter for enforcing low-rank structure of hidden factor matrix
`maxiter`	maximum number of iterations
`eps`	constant used when checking the convergence. Ensures no division by 0.
`tol`	tolerance level for convergence
`verbose`	chooses whether details should be printed to console. Default is FALSE.

Value

`B`	The estimated coefficients
`mu`	The estimated intercept
`L`	The estimated matrix of hidden factors
`f_val_vec`	The objective function values
`res_vec`	The relative change in objective function values
`iter`	The number of iterations

Examples

```
##Example

## Generate some data
n <- 50
p <- 200
q <- 100
k <- 10
set.seed(123)
X <- matrix(rbinom(n*p,1,0.5),n,p)
L <- matrix(rnorm(n*k),n,k) %*% t(matrix(rnorm(q*k),q,k))
B <- matrix(0, ncol(X), ncol(L))
activeSNPs <- sort(sample(c(1:nrow(B)), 20))
for(i in 1:length(activeSNPs)){
genes_influenced <- sort(sample(c(1:ncol(B)),5))
B[activeSNPs[i], genes_influenced] <- 2
}
E <- matrix(rnorm(n*q),n,q)
Y <- X %*% B + L + E

rho <- runif(1,3,5)
lambda <- runif(1,3,5)

## Usage
Fast_LORS(Y, X, rho, lambda)
```

`Fast_LORS_Tuning` *Fast_LORS_Tuning*

Description

`Fast_LORS_Tuning` is a function used perform parameter tuning using FastLORS instead of LORS

Usage

```
Fast_LORS_Tuning(Y, X, rho, lambda, Training, Validation, maxiter = 5000,
  eps = 2.2204e-16, tol = 1e-04, B = NULL, mu = NULL, L = NULL)
```

Arguments

`Y`	gene expression matrix
`X`	matrix of SNPs
`rho`	parameter used to enforce sparsity of B
`lambda`	parameter used to enforce low-rank structure of L
`Training`	Boolean matrix for training data
`Validation`	Boolean matrix for validation data
`maxiter`	maximum number of iterates
`eps`	a small constant to prevent dividing by zero when checking relative change in function values.

`tol`	tolerance threshold for convergence
`B`	an estimate for matrix of coefficients. Default is NULL.
`mu`	an estimate for the intercept. Default is NULL.
`L`	an estimate for the hidden factors Default is NULL.

Value

`B`	matrix of coefficients
`L`	matrix of hidden factors
`mu`	vector of intercepts
`Err`	Residual sum of squares of validation data
`f_vals`	Objective function values
`res_vec`	Relative change in objective function values
`iter`	Number of iterations

`GetMaxRho` *GetMaxRho*

Description

`GetMaxRho` is a function used to determine the maximum value as a candidate for rho. See the parmeter tuning section of Yang et al. (2013) Note: This function is adapted from the LORS MATLAB implementation

Usage

```
GetMaxRho(X, Y, L, Omega0)
```

Arguments

`X`	matrix of SNPs
`Y`	gene expression matrix
`L`	matrix of hidden factors
`Omega0`	Boolean matrix of observed entries

Value

MaxRho The maximum value of rho to be used in parameter tuning

Examples

```
##Example

## Generate some data
n <- 50
p <- 200
q <- 100
k <- 10
```

```
set.seed(123)
X <- matrix(rbinom(n*p,1,0.5),n,p)
L <- matrix(rnorm(n*k),n,k) %*% t(matrix(rnorm(q*k),q,k))
B <- matrix(0, ncol(X), ncol(L))
activeSNPs <- sort(sample(c(1:nrow(B)), 20))
for(i in 1:length(activeSNPs)){
genes_influenced <- sort(sample(c(1:ncol(B)),5))
B[activeSNPs[i], genes_influenced] <- 2
}
E <- matrix(rnorm(n*q),n,q)
Y <- X %*% B + L + E
Omega0 <- !(is.na(Y))

## Usage
GetMaxRho(X, Y, L, Omega0)
```

`HC_Screening` *HC_Screening* `HC_Screening` *is a function to apply the HC-Screening screening method of Rhyne et al. (2018) (In progress) Note: HC-Screening ranks SNPs by their higher criticism statistics and selects the top n, where n is the number of samples*

Description

HC_Screening `HC_Screening` is a function to apply the HC-Screening screening method of Rhyne et al. (2018) (In progress) Note: HC-Screening ranks SNPs by their higher criticism statistics and selects the top n, where n is the number of samples

Usage

```
HC_Screening(Y, X)
```

Arguments

`Y`	gene expression matrix
`X`	matrix of SNPs

Value

selectedSNPs The SNPs selected by HC-Screening

Examples

```
## Example

## Generate some data
n <- 50
p <- 200
q <- 100
k <- 10
set.seed(123)
X <- matrix(rbinom(n*p,1,0.5),n,p)
```

```
L <- matrix(rnorm(n*k),n,k) %*% t(matrix(rnorm(q*k),q,k))
B <- matrix(0, ncol(X), ncol(L))
activeSNPs <- sort(sample(c(1:nrow(B)), 20))
for(i in 1:length(activeSNPs)){
genes_influenced <- sort(sample(c(1:ncol(B)),5))
B[activeSNPs[i], genes_influenced] <- 2
}
E <- matrix(rnorm(n*q),n,q)
Y <- X %*% B + L + E

## Usage
HC_Screening(Y, X)
```

InitialEst — *InitialEst* `InitialEst` *is a function to build an initial estimate for B*

Description

InitialEst `InitialEst` is a function to build an initial estimate for B

Usage

```
InitialEst(Y, X, lambda = NULL)
```

Arguments

`Y`	gene expression matrix
`X`	matrix of SNPs
`lambda`	tuning parameter

Value

B An initial estimate of the coefficient matrix

Examples

```
## Example
## Generate some data
n <- 50
p <- 200
q <- 100
k <- 10
set.seed(123)
X <- matrix(rbinom(n*p,1,0.5),n,p)
L <- matrix(rnorm(n*k),n,k) %*% t(matrix(rnorm(q*k),q,k))
B <- matrix(0, ncol(X), ncol(L))
activeSNPs <- sort(sample(c(1:nrow(B)), 20))
for(i in 1:length(activeSNPs)){
  genes_influenced <- sort(sample(c(1:ncol(B)),5))
  B[activeSNPs[i], genes_influenced] <- 2
}
E <- matrix(rnorm(n*q),n,q)
Y <- X %*% B + L + E

Init_est <- InitialEst(Y,X)
```

`linspace` *linspace*

Description

`linspace` is a function to space a sequence linearly from x1 to x2

Usage

```
linspace(x1, x2, n = 100)
```

Arguments

x1	a starting point
x2	an ending point
n	length of sequence

Value

a linear spaced sequence from x1 to x2 of length n

Examples

```
linspace(100,10,5)
```

`logspace` *logspace*

Description

`logspace` is a function to space a sequence evenly on the log scale from x1 to x2

Usage

```
logspace(x1, x2, n = 50)
```

Arguments

x1	a starting point
x2	an ending point
n	length of sequence

Value

a sequence from x1 to x2 of length n spaced evenly on the log scale

Examples

```
##Example
logspace(100,10,5)
```

`LORS0` *LORS0*

Description

`LORS0` is a function for solving the LORS optimization problem through the method described in Can Yang et al. (2013). This function is adapted from the authors MATLAB implementation

Usage

```
LORS0(Y, X, rho, lambda, maxiter = 1000, eps = 2.2204e-16, tol = 1e-04,
  verbose = FALSE)
```

Arguments

`Y`	gene expression matrix
`X`	matrix of SNPs
`rho`	parameter for enforcing sparsity of coefficient matrix
`lambda`	parameter for enforcing low-rank structure of hidden factor matrix
`maxiter`	maximum number of iterations
`eps`	constant used when checking the convergence. Ensures no division by 0.
`tol`	tolerance level for convergence
`verbose`	chooses whether details should be printed to console. Default is FALSE.

Value

`B`	The estimated coefficients
`mu`	The estimated intercept
`L`	The estimated matrix of hidden factors
`f_val_vec`	The objective function values
`res_vec`	The relative change in objective function values

Examples

```
##Example

#' ## Generate some data
n <- 50
p <- 200
q <- 100
k <- 10
set.seed(123)
X <- matrix(rbinom(n*p,1,0.5),n,p)
L <- matrix(rnorm(n*k),n,k) %*% t(matrix(rnorm(q*k),q,k))
B <- matrix(0, ncol(X), ncol(L))
activeSNPs <- sort(sample(c(1:nrow(B)), 20))
for(i in 1:length(activeSNPs)){
genes_influenced <- sort(sample(c(1:ncol(B)),5))
B[activeSNPs[i], genes_influenced] <- 2
}
E <- matrix(rnorm(n*q),n,q)
Y <- X %*% B + L + E

rho <- runif(1,3,5)
lambda <- runif(1,3,5)
LORS0(Y, X, rho, lambda)
```

`LORS2` *LORS2*

Description

`LORS2` is a function used in parameter tuning in LORS. See the parameter tuning section described in Can Yang et al. (2013). This function is adapted from the authors MATLAB implementation

Usage

```
LORS2(Y, X, L, Omega1, Omega2, B, rho, lambda, tol, maxIter = 1000)
```

Arguments

`Y`	gene expression matrix
`X`	matrix of SNPs
`L`	matrix of hidden factors
`Omega1`	Boolean matrix for training data
`Omega2`	Boolean matrix for validation data
`B`	a matrix of coefficients for the SNPs
`rho`	parameter for enforcing sparsity of coefficient matrix
`lambda`	parameter for enforcing low-rank structure of hidden factor matrix
`tol`	tolerance level for convergence
`maxIter`	the maximum number of iterations

Value

`B`	The estimated coefficients
`mu`	The estimated intercept
`L`	The estimated matrix of hidden factors
`Err`	The residual sum of squares on the validation set

Examples

```
##Example

#' ## Generate some data
n <- 50
p <- 200
q <- 100
k <- 10
set.seed(123)
X <- matrix(rbinom(n*p,1,0.5),n,p)
L <- matrix(rnorm(n*k),n,k) %*% t(matrix(rnorm(q*k),q,k))
B <- matrix(0, ncol(X), ncol(L))
activeSNPs <- sort(sample(c(1:nrow(B)), 20))
for(i in 1:length(activeSNPs)){
genes_influenced <- sort(sample(c(1:ncol(B)),5))
B[activeSNPs[i], genes_influenced] <- 2
}
E <- matrix(rnorm(n*q),n,q)
Y <- X %*% B + L + E

Omega0 <- !(is.na(Y))
mask <- matrix(runif(nrow(Y)*ncol(Y)) > 0.5, nrow = nrow(Y), ncol = ncol(Y))
Omega1 <- Omega0 & mask
Omega2 <- Omega0 & !mask
rho <- runif(1,3,5)
lambda <- runif(1,3,5)
tol <- 1e-4

## Usage
LORS2(Y, X, L, Omega1, Omega2, B, rho, lambda, tol)
```

`LORSscreen` *LORSscreen* `LORSscreen` *is a function to solve the LORS-Screening optimization problem in Yang et al. (2013)*

Description

LORSscreen `LORSscreen` is a function to solve the LORS-Screening optimization problem in Yang et al. (2013)

Usage

```
LORSscreen(Y, X, lambda, tol)
```

Arguments

`Y`	gene expression matrix
`X`	a SNP
`lambda`	tuning parameter
`tol`	a tolerance level

Value

B the estimated coefficients for the SNP

L the estimated hidden factors

mu the estimate for the intercept

Examples

```
##Example

## Generate some data
n <- 50
p <- 200
q <- 100
k <- 10
set.seed(123)
X <- matrix(rbinom(n*p,1,0.5),n,p)
L <- matrix(rnorm(n*k),n,k) %*% t(matrix(rnorm(q*k),q,k))
B <- matrix(0, ncol(X), ncol(L))
activeSNPs <- sort(sample(c(1:nrow(B)), 20))
for(i in 1:length(activeSNPs)){
genes_influenced <- sort(sample(c(1:ncol(B)),5))
B[activeSNPs[i], genes_influenced] <- 2
}
E <- matrix(rnorm(n*q),n,q)
Y <- X %*% B + L + E

## Usage to build initial estimate

Bhat_initial <- c()
for(SNP_col in 1:ncol(X)){
   X1 <- matrix(X[,SNP_col], ncol = 1)
   LS <- LORSscreen(Y, X1, lambda = 0.1, 0.01)
   B_row <- LS$B
   Bhat_initial <- rbind(Bhat_initial, B_row)
}
```

`LORS_Screen_Parallel` *LORS_Screen_Parallel* `LORS_Screen_Parallel` *is a function used to run LORS-Screening on a subset of the columns of X. Can be used to perform LORS-Screening in parallel on a cluster.*

Description

LORS_Screen_Parallel `LORS_Screen_Parallel` is a function used to run LORS-Screening on a subset of the columns of X. Can be used to perform LORS-Screening in parallel on a cluster.

Usage

```
LORS_Screen_Parallel(Y, X, chunk)
```

Arguments

`Y`	gene expression matrix
`X`	matrix of SNPs
`chunk`	a group of columns to run the screening on. Done in batches of 1000.

Value

`myB`	matrix of coefficients from LORS-Screening
`lambda`	tuning parameter used in LORS-Screening

Examples

```

##Example

## Generate some data
n <- 20
p <- 50
q <- 30
k <- 4
set.seed(123)
X <- matrix(rbinom(n*p,1,0.5),n,p)
L <- matrix(rnorm(n*k),n,k) %*% t(matrix(rnorm(q*k),q,k))
B <- matrix(0, ncol(X), ncol(L))
activeSNPs <- sort(sample(c(1:nrow(B)), 20))
for(i in 1:length(activeSNPs)){
genes_influenced <- sort(sample(c(1:ncol(B)),5))
B[activeSNPs[i], genes_influenced] <- 2
}
E <- matrix(rnorm(n*q),n,q)
Y <- X %*% B + L + E

## Usage
LORS_Screen_Parallel(Y, X, chunk = 1)
```

`ParamTuneParallel` *ParamTuneParallel* `ParamTuneParallel` *is a function used to run the parameter tuning of either FastLORS or LORS in parallel*

Description

ParamTuneParallel `ParamTuneParallel` is a function used to run the parameter tuning of either FastLORS or LORS in parallel

Usage

```
ParamTuneParallel(Y, X, fold, seed = 123)
```

Arguments

`Y`	gene expression matrix
`X`	matrix of SNPs
`fold`	fold in cross-validation
`seed`	random seed used to create training and validation sets

Value

`myParams`	lambda and a sequence of rho values to use in parameter tuning
`Training`	Training Data
`Validation`	Validation Data

Examples

```
##Example

## Generate some data
n <- 20
p <- 50
q <- 30
k <- 4
set.seed(123)
X <- matrix(rbinom(n*p,1,0.5),n,p)
L <- matrix(rnorm(n*k),n,k) %*% t(matrix(rnorm(q*k),q,k))
B <- matrix(0, ncol(X), ncol(L))
activeSNPs <- sort(sample(c(1:nrow(B)), 20))
for(i in 1:length(activeSNPs)){
genes_influenced <- sort(sample(c(1:ncol(B)),5))
B[activeSNPs[i], genes_influenced] <- 2
}
E <- matrix(rnorm(n*q),n,q)
Y <- X %*% B + L + E

## Usage
ParamTuneParallel(Y, X, fold = 1)
```

`prox_1` *prox_1*

Description

`prox_1` is the soft-thresholding function. Let the entries of b be b_j. The function subtracts tau from b_j for b_j > tau, sets b_j to 0 where abs(b_j) < tau, and adds tau where b_j < -tau.

Usage

```
prox_1(b, tau)
```

Arguments

`b`	a matrix or vector
`tau`	the value to to apply soft thresholding with

Value

prox_b the soft-thresholding function applied to b with threshold tau

---

`rankHC` *rankHC `rankHC` is a function used to rank a Bhat matrix by higher criticism statistics*

Description

rankHC `rankHC` is a function used to rank a Bhat matrix by higher criticism statistics

Usage

```
rankHC(Bhat_standardized)
```

Arguments

`Bhat_standardized`	a standardized coefficient matrix

Value

`index`	The indices of the sorted Higher Criticism values
`HC_vec`	The higher criticism statistics for the standardized Bhat matrix

---

`Run_LORS` *Run_LORS*

Description

`Run_LORS` is a function used to run either FastLORS or LORS

Usage

```
Run_LORS(Y, X, method = "FastLORS", screening = "LORS-Screening",
  tune_method = "FastLORS", seed = 123, maxiter = 10000,
  eps = 2.2204e-16, tol = 1e-04, cross_valid = TRUE)
```

Arguments

`Y`	gene expression matrix
`X`	matrix of SNPs
`method`	chooses with modeling method to run
`screening`	Either "LORS-Screening", "HC-Screening", or "None". The default method, LORS-Screening, is recommended if the number of SNPs is large. HC-Screening of Rhyne et al. (2018) is under development but is included here as an option.
`tune_method`	chooses whether FastLORS should be used for parameter tuning or the original LORS procedure should be used. Default is FastLORS
`seed`	random seed to be used for setting training and validation set. Default is 123.
`maxiter`	maximum number of iterations
`eps`	constant used when checking the convergence. Ensures no division by 0.
`tol`	tolerance level for convergence
`cross_valid`	chooses whether cross-validation should be used in parameter tuning. Default is TRUE.

Value

`LORS_Obj or Fast_LORS_Obj`	A list produced from LORS or FastLORS containing (1) B: estimate of the coefficient matrix (2) L: estimate of the matrix of hidden factors (3) mu: estiamte of the vector of intercepts (4) f_val_vec: objective function values and (5) res_vec: relative change in objective function values
`selectedSNPs`	The SNPs selected by the screening method
`screening_time`	The time (in seconds) spent on screening step
`param_time`	The time (in seconds) spent on the parameter tuning step
`model_time`	The time (in seconds) spent on the joint modeling step
`total_time`	The time (in seconds) spent on the screening, parameter tuning, and joint modeling steps
`rho`	The value of rho chosen through parameter tuning
`lambda`	The value of lambda chosen through parameter tuning

Examples

```
##Example

## Generate some data
n <- 20
p <- 50
q <- 30
k <- 4
set.seed(123)
X <- matrix(rbinom(n*p,1,0.5),n,p)
L <- matrix(rnorm(n*k),n,k) %*% t(matrix(rnorm(q*k),q,k))
B <- matrix(0, ncol(X), ncol(L))
activeSNPs <- sort(sample(c(1:nrow(B)), 20))
for(i in 1:length(activeSNPs)){
genes_influenced <- sort(sample(c(1:ncol(B)),5))
B[activeSNPs[i], genes_influenced] <- 2
}
E <- matrix(rnorm(n*q),n,q)
Y <- X %*% B + L + E

## Usage
Run_LORS(Y, X, method = "FastLORS")
```

Run_LORS_Screening *Run_LORS_Screening* `Run_LORS_Screening` *is a function to to apply the LORS-Screening Algorithm in Yang et al. (2013)*

Description

Run_LORS_Screening `Run_LORS_Screening` is a function to to apply the LORS-Screening Algorithm in Yang et al. (2013)

Usage

```
Run_LORS_Screening(Y, X, lambda = NULL)
```

Arguments

`Y`	gene expression matrix
`X`	matrix of SNPs
`lambda`	tuning parameter

Value

selectedSNPs the SNPs selected by LORS-Screening

Examples

```
##Example

## Generate some data
n <- 50
p <- 200
q <- 100
k <- 10
set.seed(123)
X <- matrix(rbinom(n*p,1,0.5),n,p)
L <- matrix(rnorm(n*k),n,k) %*% t(matrix(rnorm(q*k),q,k))
B <- matrix(0, ncol(X), ncol(L))
activeSNPs <- sort(sample(c(1:nrow(B)), 20))
for(i in 1:length(activeSNPs)){
genes_influenced <- sort(sample(c(1:ncol(B)),5))
B[activeSNPs[i], genes_influenced] <- 2
}
E <- matrix(rnorm(n*q),n,q)
Y <- X %*% B + L + E

selectedSNPs <- Run_LORS_Screening(Y, X)
```

`S`	*S S is a function used internally in rankHC. It calculates empirical cdf's.*

Description

S `S` is a function used internally in rankHC. It calculates empirical cdf's.

Usage

```
S(t, my_matrix)
```

Arguments

`t`	cutoff value of empirical cdf
`my_matrix`	a coefficient matrix

Value

The empirical distribution function of the coefficients evaluated at t

---

`softImpute`	*softImpute*

Description

`softImpute` is a function from Mazudmer et al. (2010). It solves the problem min || X - Z ||_Omega + alpha || Z ||_Nulear and is used in parameter tuning for LORS. Note: This function is adapted from the LORS MATLAB implementation

Usage

```
softImpute(X, Z, Omega0, Omega1, Omega2, alpha0, maxRank)
```

Arguments

`X`	a (possibly) incomplete matrix
`Z`	the target matrix
`Omega0`	Boolean matrix of observed entries
`Omega1`	Boolean matrix of training entries
`Omega2`	Boolean matrix of validation entries
`alpha0`	initial tuning parameter
`maxRank`	maximum rank of the solution

Value

`Z`	Estimate of the target matrix
`Err`	Squared Error of the difference between X and Z on the validation set
`rank_alpha`	The rank of the estimates
`znorm`	The sum of the soft-thresholded singular values of the estimates
`Alpha`	The tuning parameters used

`standardizeBhat` *standardizeBhat* `standardizeBhat` *is a function used to standardize a coefficient matrix*

Description

standardizeBhat `standardizeBhat` is a function used to standardize a coefficient matrix

Usage

```
standardizeBhat(Y, X, Bhat)
```

Arguments

`Y`	gene expression matrix
`X`	matrix of SNPs
`Bhat`	a coefficient matrix

Value

A standardized estimate of the coefficient matrix.

`survival` *survival* `survival` *is a function used internally in rankHC. It calculates the survival function of the standard normal distribution*

Description

survival `survival` is a function used internally in rankHC. It calculates the survival function of the standard normal distribution

Usage

```
survival(t)
```

Arguments

`t`	a cutoff value

Value

s 1 - Pr(Z <= t) where Z is a standard normal random variable

svd_st *svd_st*

Description

svd_st is a function for performing soft-thresholded singular value decomposition of a matrix X

Usage

```
svd_st(X, lambda)
```

Arguments

X	a matrix
lambda	the value to to apply soft thresholding with

Value

L the soft-thresholded singular value decomposition of X.

SVT *SVT*

Description

SVT is a function to perform soft-thresholded singular value decomposition. It is used to get an initial estimate for L. Note: This function is adapted from the LORS MATLAB implementation

Usage

```
SVT(Y, lambda)
```

Arguments

Y	gene expression matrix
lambda	a tuning parameter

Value

L the singular value decomposition of Y, soft-thresholded with lambda.

### Examples

```
##Example
set.seed(123)

Y <-matrix(rnorm(50*100, 7,1), nrow = 50, ncol = 100)
lambda <- runif(1,3,5)
SVT(Y, lambda)
```

Index