Package ‘PRIMAL’

January 20, 2025

Type Package

Title Parametric Simplex Method for Sparse Learning

Version 1.0.2

Date 2020-01-21

Author Zichong Li, Qianli Shen

Maintainer Zichong Li <zichongli5@gmail.com>

LinkingTo Rcpp, RcppEigen

Description Implements a unified framework of parametric simplex method for a variety of sparse learning problems (e.g., Dantzig selector (for linear regression), sparse quantile regression, sparse support vector machines, and compressive sensing) combined with efficient hyper-parameter selection strategies. The core algorithm is implemented in C++ with Eigen3 support for portable high performance linear algebra. For more details about parametric simplex method, see Haotian Pang (2017) <https://papers.nips.cc/paper/6623-parametric-simplex-method-for-sparse-learning.pdf>.

Imports Matrix

License GPL (>= 2)

NeedsCompilation yes

RoxygenNote 6.1.1

Repository CRAN

Date/Publication 2020-01-22 11:10:02 UTC

Contents

`PRIMAL-package` *Parametric Simplex Method for Sparse Learning*

Description

A package for parametric simplex method for sparse learning

Details

Package:	PRIMAL
Type:	Package
Version:	1.0.0
Date:	2019-08-15

The package "PRIMAL" provides 5 main functions:
(1) The dantzig selector solver applying simplex method. Please refer to `Dantzig_solver`.
(2) The sparse SVM solver applying simplex method. Please refer to `SparseSVM_solver`.
(3) The compressed sensing solver. Please refer to `CompressedSensing_solver`.
(4) The quantile regression solver. Please refer to `QuantileRegression_solver`.
(5) The solver for standard formulation of parametric simplex method. Please refer to `PSM_solver`.

Author(s)

Qianli Shen, Zichong Li

See Also

`plot.primal`, `print.primal`, `coef.primal`

`coef.primal` *Coef function for S3 class "primal"*

Description

Print the estimated solution correspond to a specific parameter.

Usage

```
## S3 method for class 'primal'
coef(object, n = NULL, ...)
```

Arguments

`object`	An object with S3 class `"primal"`.
`n`	The index of the wanted parameter.
`...`	System reserved (No specific usage)

See Also

`Dantzig_solver`, `SparseSVM_solver`

`CompressedSensing_solver`
Solve given compressed sensing problem in parametric simplex method

---

Description

Solve given compressed sensing problem in parametric simplex method

Usage

```
CompressedSensing_solver(X, y, max_it = 50, lambda_threshold = 0.01)
```

Arguments

`X`	`x` is an `n` by `d` data matrix
`y`	`y` is a length `n` response vector
`max_it`	This is the number of the maximum path length one would like to achieve. The default length is `50`.
`lambda_threshold`	The parametric simplex method will stop when the calculated parameter is smaller than lambda. The default value is `0.01`.

Value

An object with S3 class `"primal"` is returned:

`data`	The `n` by `d` data matrix from the input
`response`	The length `n` response vector from the input
`beta`	A matrix of regression estimates whose columns correspond to regularization parameters for parametric simplex method.
`df`	The degree of freedom (number of nonzero coefficients) along the solution path.
`value`	The sequence of optimal value of the object function corresponded to the sequence of lambda.
`iterN`	The number of iteration in the program.
`lambda`	The sequence of regularization parameters `lambda` obtained in the program.
`type`	The type of the problem, such as `Dantzig` and `SparseSVM`.

See Also

primal-package, Dantzig_solver

Examples

```
## Compressed Sensing
## We set X to be standard normal random matrix and generate Y using gaussian noise.
## Generate the design matrix and coefficient vector
n = 100 # sample number
d = 250 # sample dimension
c = 0.5 # correlation parameter
s = 20  # support size of coefficient
set.seed(1024)
X = scale(matrix(rnorm(n*d),n,d)+c*rnorm(n))/sqrt(n-1)*sqrt(n)
beta = c(rnorm(s), rep(0, d-s))
## Generate response using Gaussian noise, and solve the solution path
noise = rnorm(n)
Y = X%*%beta + noise
## Compressed Sensing solved with parametric simplex method
fit.compressed = CompressedSensing_solver(X, Y, max_it = 100, lambda_threshold = 0.01)
###lambdas used
print(fit.compressed$lambda)
## number of nonzero coefficients for each lambda
print(fit.compressed$df)
## Visualize the solution path
plot(fit.compressed)
```

Dantzig_solver — *Solve given Dantzig selector problem in parametric simplex method*

Description

Solve given Dantzig selector problem in parametric simplex method

Usage

```
Dantzig_solver(X, y, max_it = 50, lambda_threshold = 0.01)
```

Arguments

`X`	x is an n by d data matrix
`y`	y is a length n response vector
`max_it`	This is the number of the maximum path length one would like to achieve. The default length is 50.
`lambda_threshold`	The parametric simplex method will stop when the calculated parameter is smaller than lambda. The default value is 0.01.

Value

An object with S3 class "primal" is returned:

data	The n by d data matrix from the input
response	The length n response vector from the input
beta	A matrix of regression estimates whose columns correspond to regularization parameters for parametric simplex method.
df	The degree of freedom (number of nonzero coefficients) along the solution path.
value	The sequence of optimal value of the object function corresponded to the sequence of lambda.
iterN	The number of iteration in the program.
lambda	The sequence of regularization parameters lambda obtained in the program.
type	The type of the problem, such as Dantzig and SparseSVM.

See Also

primal-package

Examples

```
## Dantzig
## We set X to be standard normal random matrix and generate Y using gaussian noise.
## Generate the design matrix and coefficient vector
n = 100 # sample number
d = 250 # sample dimension
c = 0.5 # correlation parameter
s = 20  # support size of coefficient
set.seed(1024)
X = scale(matrix(rnorm(n*d),n,d)+c*rnorm(n))/sqrt(n-1)*sqrt(n)
beta = c(rnorm(s), rep(0, d-s))
## Generate response using Gaussian noise, and solve the solution path
noise = rnorm(n)
Y = X%*%beta + noise
## Dantzig selection solved with parametric simplex method
fit.dantzig = Dantzig_solver(X, Y, max_it = 100, lambda_threshold = 0.01)
###lambdas used
print(fit.dantzig$lambda)
## number of nonzero coefficients for each lambda
print(fit.dantzig$df)
## Visualize the solution path
plot(fit.dantzig)
```

plot.primal	*Plot function for S3 class "primal"*

Description

Plot regularization path and parameter obtained from the algorithm.

Usage

```
## S3 method for class 'primal'
plot(x, n = NULL, ...)
```

Arguments

x	An object with S3 class "primal"
n	If n = NULL, three graph will be shown together. If n is a number, then the corresponding graph will be shown.
...	System reserved (No specific usage)

See Also

Dantzig_solver, SparseSVM_solver

print.primal	*Print function for S3 class "primal"*

Description

Print the information about the model usage, the parameter path, degree of freedom of the solution path.

Usage

```
## S3 method for class 'primal'
print(x, ...)
```

Arguments

x	An object with S3 class "primal".
...	System reserved (No specific usage)

See Also

Dantzig_solver, SparseSVM_solver

PSM_solver *Solve given problem in parametric simplex method*

Description

Solve given problem in parametric simplex method

Usage

```
PSM_solver(A, b, b_bar, c, c_bar, B_init = NULL, max_it = 50,
  lambda_threshold = 0.01)
```

Arguments

A	A is an n by d data matrix
b	b is a length n response vector
b_bar	b_bar is a length n vector time to parameter in constraints.
c	c is a length d vector in target function.
c_bar	c_bar is a length d vector time to parameter in target function
B_init	B_init is the index of initial basic colume.
max_it	This is the number of the maximum path length one would like to achieve. The default length is 50.
lambda_threshold	The parametric simplex method will stop when the calculated parameter is smaller than lambda. The default value is 0.01.

Value

An object with S3 class "primal" is returned:

data	The n by d data matrix from the input
response	The length n response vector from the input
beta	A matrix of regression estimates whose columns correspond to regularization parameters for parametric simplex method.
beta0	A vector of regression estimates whose index correspond to regularization parameters for parametric simplex method.
df	The degree of freecom (number of nonzero coefficients) along the solution path.
value	The sequence of optimal value of the object function corresponded to the sequence of lambda.
iterN	The number of iteration in the program.
lambda	The sequence of regularization parameters lambda obtained in the program.
type	The type of the problem, such as Dantzig and SparseSVM.

See Also

primal-package

Examples

```
## This example show how to use PSM_solver() to solve dantzig problem.
## Generate the design matrix and coefficient vector
n = 100 # sample number
d = 250 # sample dimension
c = 0.5 # correlation parameter
s = 20  # support size of coefficient
set.seed(1024)
X = scale(matrix(rnorm(n*d),n,d)+c*rnorm(n))/sqrt(n-1)*sqrt(n)
beta = c(rnorm(s), rep(0, d-s))
## Generate response using Gaussian noise, and solve the solution path
noise = rnorm(n)
Y = X%*%beta + noise
## Define parameters for dantzig problem
XtX = t(X)%*%X
A = cbind(cbind(rbind(XtX,-XtX),-rbind(XtX,-XtX)),diag(rep(1,2*d)))
b = rbind(t(X)%*%Y,-t(X)%*%Y)
c = c(rep(-1,2*d),rep(0,2*d))
c_bar = rep(0,4*d)
b_bar = rep(1,2*d)
B_init = seq(2*d,4*d-1)
## Dantzig selection solved with parametric simplex method
fit.dantzig = PSM_solver(A, b, b_bar, c, c_bar, B_init, max_it = 50, lambda_threshold = 0.01)
###lambdas used
print(fit.dantzig$lambda)
## number of nonzero coefficients for each lambda
print(fit.dantzig$df)
## Visualize the solution path
plot(fit.dantzig)
```

`QuantileRegression_solver` *Solve given quantile regression problem in parametric simplex method*

Description

Solve given quantile regression problem in parametric simplex method

Usage

```
QuantileRegression_solver(X, y, max_it = 50, lambda_threshold = 0.01,
  tau = 0.5)
```

Arguments

`X`	`x` is an `n` by `d` data matrix
`y`	`y` is a length `n` response vector
`max_it`	This is the number of the maximum path length one would like to achieve. The default length is `50`.
`lambda_threshold`	The parametric simplex method will stop when the calculated parameter is smaller than lambda. The default value is `0.01`.
`tau`	The quantile number you want. The default quantile is `0.5`

Value

An object with S3 class `"primal"` is returned:

`data`	The `n` by `d` data matrix from the input
`response`	The length `n` response vector from the input
`beta`	A matrix of regression estimates whose columns correspond to regularization parameters for parametric simplex method.
`df`	The degree of freedom (number of nonzero coefficients) along the solution path.
`value`	The sequence of optimal value of the object function corresponded to the sequence of lambda.
`iterN`	The number of iteration in the program.
`lambda`	The sequence of regularization parameters `lambda` obtained in the program.
`type`	The type of the problem, such as `Dantzig` and `SparseSVM`.

See Also

`primal-package`, `Dantzig_solver`

Examples

```
## Quantile Regression
## We set X to be standard normal random matrix and generate Y using gaussian noise
## with default quantile number to be 0.5.
## Generate the design matrix and coefficient vector
n = 100 # sample number
d = 250 # sample dimension
c = 0.5 # correlation parameter
s = 20  # support size of coefficient
set.seed(1024)
X = scale(matrix(rnorm(n*d),n,d)+c*rnorm(n))/sqrt(n-1)*sqrt(n)
beta = c(rnorm(s), rep(0, d-s))
## Generate response using Gaussian noise, and solve the solution path
noise = rnorm(n)
Y = X%*%beta + noise
## Quantile Regression problem solved with parametric simplex method
fit.quantile = QuantileRegression_solver(X, Y, max_it = 100, lambda_threshold = 0.01)
```

```
###lambdas used
print(fit.quantile$lambda)
## number of nonzero coefficients for each lambda
print(fit.quantile$df)
## Visualize the solution path
plot(fit.quantile)
```

`SparseSVM_solver` *Solve given Sparse SVM problem in parametric simplex method*

Description

Solve given Sparse SVM problem in parametric simplex method

Usage

```
SparseSVM_solver(X, y, max_it = 50, lambda_threshold = 0.01)
```

Arguments

`X`	`x` is an `n` by `d` data matrix
`y`	`y` is a length `n` response vector
`max_it`	This is the number of the maximum path length one would like to achieve. The default length is `50`.
`lambda_threshold`	The parametric simplex method will stop when the calculated parameter is smaller than lambda. The default value is `0.01`.

Value

An object with S3 class `"primal"` is returned:

`data`	The `n` by `d` data matrix from the input
`response`	The length `n` response vector from the input
`beta`	A matrix of regression estimates whose columns correspond to regularization parameters for parametric simplex method.
`beta0`	A vector of regression estimates whose index correspond to regularization parameters for parametric simplex method.
`df`	The degree of freecom (number of nonzero coefficients) along the solution path.
`value`	The sequence of optimal value of the object function corresponded to the sequence of lambda.
`iterN`	The number of iteration in the program.
`lambda`	The sequence of regularization parameters `lambda` obtained in the program.
`type`	The type of the problem, such as `Dantzig` and `SparseSVM`.

See Also

primal-package

Examples

```
## SparseSVM
## We set the X matrix to be normal random matrix and Y is a vector consists of -1 and 1
## with the number of iteration to be 1000.
## Generate the design matrix and coefficient vector
n = 200 # sample number
d = 100 # sample dimension
c = 0.5 # correlation parameter
s = 20  # support size of coefficient
set.seed(1024)
X = matrix(rnorm(n*d),n,d)+c*rnorm(n)
## Generate response and solve the solution path
Y <- sample(c(-1,1),n,replace = TRUE)
## Sparse SVM solved with parametric simplex method
fit.SVM = SparseSVM_solver(X, Y, max_it = 1000, lambda_threshold = 0.01)
## lambdas used
print(fit.SVM$lambda)
## Visualize the solution path
plot(fit.SVM)
```

Index