Package ‘PACLasso’

April 29, 2019

Type Package

Title Penalized and Constrained Lasso Optimization

Version 1.0.0

Date 2019-4-11

Maintainer Courtney Paulson <cpaulson@rhsmith.umd.edu>

Description An implementation of both the equality and inequality constrained lasso functions for the algorithm described in ``Penalized and Constrained Optimization'' by James, Paulson, and Rusmevichientong (Journal of the American Statistical Association, 2019; see <http://www-bcf.usc.edu/~gareth/research/PAC.pdf> for a full-text version of the paper). The algorithm here is designed to allow users to define linear constraints (either equality or inequality constraints) and use a penalized regression approach to solve the constrained problem. The functions here are used specifically for constraints with the lasso formulation, but the method described in the PaC paper can be used for a variety of scenarios. In addition to the simple examples included here with the corresponding functions, complete code to entirely reproduce the results of the paper is available online through the Journal of the American Statistical Association.

Depends R (>= 3.3.0), methods (>= 3.4.4), penalized (>= 0.9)

Imports MASS (>= 7.3), lars (>= 1.2), quadprog (>= 1.5), limSolve (>= 1.5.5.3)

License GPL-3

URL http://www-bcf.usc.edu/~gareth/research/PAC.pdf

Repository CRAN

Encoding UTF-8

LazyData true

RoxygenNote 6.1.1

NeedsCompilation no

Author Courtney Paulson [aut, cre],
Gareth James [ctb],
Paat Rusmevichientong [ctb]

Date/Publication 2019-04-29 21:30:16 UTC

R topics documented:

`generate.data` *Function to Randomly Generate Data (with Constraints)*

Description

This function is primarily used for reproducibility. It will generate a data set of a given size with a given number of constraints for testing function code.

Usage

```
generate.data(n = 1000, p = 10, m = 5, cov.mat = NULL, s = 5,
  sigma = 1, glasso = F, err = 0)
```

Arguments

`n`	number of rows in randomly-generated data set (default is 1000)
`p`	number of variables in randomly-generated data set (default is 10)
`m`	number of constraints in randomly-generated constraint matrix (default is 5)
`cov.mat`	a covariance matrix applied in the generation of data to impose a correlation structure. Default is NULL (no correlation)
`s`	number of true non-zero elements in coefficient vector beta1 (default is 5)
`sigma`	standard deviation of noise in response (default is 1, indicating standard normal)
`glasso`	should the generalized Lasso be used (TRUE) or standard Lasso (FALSE). Default is FALSE
`err`	error to be introduced in random generation of coefficient values. Default is no error (err = 0)

Value

x generated x data

y generated response y vector

C.full generated full constraint matrix (with constraints of the form C.full*beta=b)

b generated constraint vector b

b.run if error was included, the error-adjusted value of b

beta the complete beta vector, including generated beta1 and beta2

References

Gareth M. James, Courtney Paulson, and Paat Rusmevichientong (JASA, 2019) "Penalized and Constrained Optimization." (Full text available at http://www-bcf.usc.edu/~gareth/research/PAC.pdf)

Examples

```
random_data = generate.data(n = 500, p = 20, m = 10)
dim(random_data$x)
head(random_data$y)
dim(random_data$C.full)
random_data$beta
```

---

lars.c — *Constrained LARS Coefficient Function (Equality Constraints)*

Description

This function computes the PaC constrained LASSO coefficient paths following the methodology laid out in the PaC paper. This function could be called directly as a standalone function, but the authors recommend using lasso.c for any implementation. This is because lasso.c has additional checks for errors across the coefficient paths and allows for users to go forwards and backwards through the paths if the paths are unable to compute in a particular direction for a particular run.

Usage

```
lars.c(x, y, C.full, b, l.min = -2, l.max = 6, step = 0.2,
  beta0 = NULL, verbose = F, max.it = 12, intercept = T,
  normalize = T, forwards = T)
```

Arguments

x	independent variable matrix of data to be used in calculating PaC coefficient paths
y	response vector of data to be used in calculating PaC coefficient paths
C.full	complete constraint matrix C (with constraints of the form C.full*beta=b)
b	constraint vector b

l.min	lowest value of lambda to consider (used as 10^l.min). Default is -2
l.max	largest value of lambda to consider (used as 10^l.max). Default is 6
step	step size increase in lambda attempted at each iteration (by a factor of 10^step). Default is 0.2
beta0	initial guess for beta coefficient vector. Default is NULL (indicating initial vector should be calculated by algorithm)
verbose	should function print output at each iteration (TRUE) or not (FALSE). Default is FALSE
max.it	maximum number of times step size is halved before the algorithm terminates and gives a warning. Default is 12
intercept	should intercept be included in modeling (TRUE) or not (FALSE). Default is TRUE.
normalize	should x data be normalized. Default is TRUE
forwards	if forwards = F, then the algorithm starts at 10^l.max and moves backwards (without the forward step). If forwards = T, algorithm starts at 10^l.min and works forward. Default is FALSE

Value

coefs A p by length(lambda) matrix with each column corresponding to the beta estimate for that lambda

lambda the grid of lambdas used to calculate the coefficients on the coefficient path

intercept vector with each element corresponding to intercept for corresponding lambda

error did the algorithm terminate due to too many iterations (TRUE or FALSE)

b2index the index of the beta2 values identified by the algorithm at each lambda

References

Gareth M. James, Courtney Paulson, and Paat Rusmevichientong (JASA, 2019) "Penalized and Constrained Optimization." (Full text available at http://www-bcf.usc.edu/~gareth/research/PAC.pdf)

Examples

```
random_data = generate.data(n = 500, p = 20, m = 10)
lars_fit = lars.c(random_data$x, random_data$y, random_data$C.full, random_data$b)
lars_fit$lambda
lars_fit$error
### The coefficients for the first lambda value
lars_fit$coefs[1,]
### Example of code where path is unable
### to be finished (only one iteration)
lars_err = lars.c(random_data$x, random_data$y, random_data$C.full,
random_data$b, max.it = 1)
lars_err$error
lars_err$lambda
```

lars.ineq *Constrained LARS Coefficient Function with Inequality Constraints*

Description

This function computes the PaC constrained LASSO coefficient paths following the methodology laid out in the PaC paper but with inequality constraints. This function could be called directly as a standalone function, but the authors recommend using `lasso.ineq` for any implementation. This is because `lasso.ineq` has additional checks for errors across the coefficient paths and allows for users to go forwards and backwards through the paths if the paths are unable to compute in a particular direction for a particular run.

Usage

```
lars.ineq(x, y, C.full, b, l.min = -2, l.max = 6, step = 0.2,
  beta0 = NULL, verbose = F, max.it = 12, intercept = T,
  normalize = T, forwards = T)
```

Arguments

`x`	independent variable matrix of data to be used in calculating PaC coefficient paths
`y`	response vector of data to be used in calculating PaC coefficient paths
`C.full`	complete inequality constraint matrix C (with inequality constraints of the form `C.full*beta` >= `b`))
`b`	constraint vector b
`l.min`	lowest value of lambda to consider (used as 10^`l.min`). Default is -2
`l.max`	largest value of lambda to consider (used as 10^`l.max`). Default is 6
`step`	step size increase in lambda attempted at each iteration (by a factor of 10^`step`). Default is 0.2
`beta0`	initial guess for `beta` coefficient vector. Default is NULL (indicating initial vector should be calculated by algorithm)
`verbose`	should function print output at each iteration (TRUE) or not (FALSE). Default is FALSE
`max.it`	maximum number of times step size is halved before the algorithm terminates and gives a warning. Default is 12
`intercept`	should intercept be included in modeling (TRUE) or not (FALSE). Default is TRUE.
`normalize`	should `x` data be normalized. Default is TRUE
`forwards`	if `forwards` = F, then the algorithm starts at 10^`l.max` and moves backwards (without the forward step). If `forwards` = T, algorithm starts at 10^`l.min` and works forward. Default is FALSE

Value

`coefs` A `p` by length(`lambda`) matrix with each column corresponding to the beta estimate for that lambda

`lambda` the grid of lambdas used to calculate the coefficients on the coefficient path

`intercept` vector with each element corresponding to intercept for corresponding lambda

`error` did the algorithm terminate due to too many iterations (TRUE or FALSE)

`b2index` the index of the `beta2` values identified by the algorithm at each lambda

References

Gareth M. James, Courtney Paulson, and Paat Rusmevichientong (JASA, 2019) "Penalized and Constrained Optimization." (Full text available at http://www-bcf.usc.edu/~gareth/research/PAC.pdf)

Examples

```
random_data = generate.data(n = 500, p = 20, m = 10)
lars_fit = lars.ineq(random_data$x, random_data$y, random_data$C.full, random_data$b)
lars_fit$lambda
lars_fit$error
### The coefficients for the first lambda value
lars_fit$coefs[1,]
### Example of code where path is unable to be finished
### (only one iteration)
lars_err = lars.ineq(random_data$x, random_data$y, random_data$C.full,
random_data$b, max.it = 1)
lars_err$error
lars_err$lambda
```

`lasso.c` *Complete Run of Constrained LASSO Path Function (Equality Constraints)*

Description

This is a wrapper function for the `lars.c` PaC constrained Lasso function. `lasso.c` controls the overall path, providing checks for the path and allowing the user to control how the path is computed (and what to do in the case of a stopped path).

Usage

```
lasso.c(x, y, C.full, b, l.min = -2, l.max = 6, step = 0.2,
  beta0 = NULL, verbose = F, max.it = 12, intercept = T,
  normalize = T, backwards = F)
```

Arguments

- `x` independent variable matrix of data to be used in calculating PaC coefficient paths
- `y` response vector of data to be used in calculating PaC coefficient paths
- `C.full` complete constraint matrix C (with constraints of the form `C.full*beta=b`)
- `b` constraint vector b
- `l.min` lowest value of lambda to consider (used as 10^`l.min`). Default is -2
- `l.max` largest value of lambda to consider (used as 10^`l.max`). Default is 6
- `step` step size increase in lambda attempted at each iteration (by a factor of 10^`step`). Default is 0.2
- `beta0` initial guess for beta coefficient vector. Default is NULL (indicating initial vector should be calculated by algorithm)
- `verbose` should function print output at each iteration (TRUE) or not (FALSE). Default is FALSE
- `max.it` maximum number of times step size is halved before the algorithm terminates and gives a warning. Default is 12
- `intercept` should intercept be included in modeling (TRUE) or not (FALSE). Default is TRUE.
- `normalize` should X data be normalized. Default is TRUE
- `backwards` which direction should algorithm go, backwards from lambda = 10^`l.max` (TRUE) or forwards from 10^`l.max` and then backwards if algorithm gets stuck (FALSE). Default is FALSE.

Value

`coefs` A `p` by length(`lambda`) matrix with each column corresponding to the beta estimate for that lambda

`lambda` vector of values of lambda that were fit

`intercept` vector with each element corresponding to intercept for corresponding lambda

`error` Indicator of whether the algorithm terminated early because max.it was reached

References

Gareth M. James, Courtney Paulson, and Paat Rusmevichientong (JASA, 2019) "Penalized and Constrained Optimization." (Full text available at http://www-bcf.usc.edu/~gareth/research/PAC.pdf)

Examples

```
random_data = generate.data(n = 500, p = 20, m = 10)
lasso_fit = lasso.c(random_data$x, random_data$y, random_data$C.full, random_data$b)
lasso_fit$lambda
lasso_fit$error
### The coefficients for the first lambda value
lasso_fit$coefs[1,]
```

```
### Example of code where path is unable to be finished
### (only one iteration), so both directions will be tried
lasso_err = lasso.c(random_data$x, random_data$y, random_data$C.full,
random_data$b, max.it = 1)
lasso_err$error
lasso_err$lambda
```

`lasso.ineq` *Complete Run of Constrained LASSO Path Function with Inequality Constraints*

Description

This is a wrapper function for the `lars.c` PaC constrained Lasso function. `lasso.c` controls the overall path, providing checks for the path and allowing the user to control how the path is computed (and what to do in the case of a stopped path).

Usage

```
lasso.ineq(x, y, C.full, b, l.min = -2, l.max = 6, step = 0.2,
  beta0 = NULL, verbose = F, max.it = 12, intercept = T,
  normalize = T, backwards = F)
```

Arguments

`x`	independent variable matrix of data to be used in calculating PaC coefficient paths
`y`	response vector of data to be used in calculating PaC coefficient paths
`C.full`	complete constraint matrix C (with inequality constraints of the form `C.full*beta >= b`))
`b`	constraint vector b
`l.min`	lowest value of lambda to consider (used as 10^`l.min`). Default is -2
`l.max`	largest value of lambda to consider (used as 10^`l.max`). Default is 6
`step`	step size increase in lambda attempted at each iteration (by a factor of 10^`step`). Default is 0.2
`beta0`	initial guess for beta coefficient vector. Default is NULL (indicating initial vector should be calculated by algorithm)
`verbose`	should function print output at each iteration (TRUE) or not (FALSE). Default is FALSE
`max.it`	maximum number of times step size is halved before the algorithm terminates and gives a warning. Default is 12
`intercept`	should intercept be included in modeling (TRUE) or not (FALSE). Default is TRUE.
`normalize`	should X data be normalized. Default is TRUE
`backwards`	which direction should algorithm go, backwards from lambda = 10^`l.max` (TRUE) or forwards from 10^`l.max` and then backwards if algorithm gets stuck (FALSE). Default is FALSE.

Value

`coefs` A p by length(`lambda`) matrix with each column corresponding to the beta estimate for that lambda

`lambda` vector of values of lambda that were fit

`intercept` vector with each element corresponding to intercept for corresponding lambda

`error` Indicator of whether the algorithm terminated early because max.it was reached

References

Gareth M. James, Courtney Paulson, and Paat Rusmevichientong (JASA, 2019) "Penalized and Constrained Optimization." (Full text available at http://www-bcf.usc.edu/~gareth/research/PAC.pdf)

Examples

```
random_data = generate.data(n = 500, p = 20, m = 10)
lasso_fit = lasso.ineq(random_data$x, random_data$y, random_data$C.full, random_data$b)
lasso_fit$lambda
lasso_fit$error
### The coefficients for the first lambda value
lasso_fit$coefs[1,]
### Example of code where path is unable to be finished
### (only one iteration), so both directions will be tried
lasso_err = lasso.ineq(random_data$x, random_data$y, random_data$C.full,
random_data$b, max.it = 1)
lasso_err$error
lasso_err$lambda
```

`lin.int` *Initialize Linear Programming Fit (Equality Constraints)*

Description

This function is called internally by `lars.c` to get the linear programming initial fit if the user requests implementation of the algorithm starting at the largest lambda value and proceeding backwards.

Usage

```
lin.int(C.full, b)
```

Arguments

`C.full`	complete constraint matrix C (with constraints of the form `C.full*beta=b`)
`b`	constraint vector b

Value

`beta` the initial beta vector of coefficients to use for the PaC algorithm

Examples

```
random_data = generate.data(n = 500, p = 20, m = 10)
lin_start = lin.int(random_data$C.full, random_data$b)
lin_start
```

---

lin.int.ineq *Initialize Linear Programming Fit with Inequality Constraints*

Description

This function is called internally by `lars.ineq` to get the linear programming initial fit if the user requests implementation of the algorithm starting at the largest lambda value and proceeding backwards.

Usage

```
lin.int.ineq(C.full, b)
```

Arguments

`C.full`	complete constraint matrix C (with inequality constraints of the form `C.full*beta >= b`))
`b`	constraint vector b

Value

`beta` the initial beta vector of coefficients to use for the PaC algorithm

Examples

```
random_data = generate.data(n = 500, p = 20, m = 10)
lin_start = lin.int.ineq(random_data$C.full, random_data$b)
lin_start
```

---

quad.int *Initialize Quadratic Programming Fit (Equality Constraints)*

Description

This function is called internally by `lars.c` to get the quadratic programming fit if the user requests implementation of the algorithm starting at the smallest lambda value and proceeding forwards.

Usage

```
quad.int(x, y, C.full, b, lambda, d = 10^-7)
```

Arguments

x	independent variable matrix of data to be used in calculating PaC coefficient paths
y	response vector of data to be used in calculating PaC coefficient paths
C.full	complete constraint matrix C (with constraints of the form `C.full*beta=b`)
b	constraint vector b
lambda	value of lambda
d	very small diagonal term to allow for SVD (default 10^-7)

Value

`beta` the initial beta vector of coefficients to use for the PaC algorithm

Examples

```
random_data = generate.data(n = 500, p = 20, m = 10)
quad_start = quad.int(random_data$x, random_data$y, random_data$C.full,
random_data$b, lambda = 0.01)
quad_start
```

---

quad.int.ineq *Initialize Quadratic Programming Fit with Inequality Constraints*

Description

This function is called internally by `lars.ineq` to get the quadratic programming fit if the user requests implementation of the algorithm starting at the smallest lambda value and proceeding forwards.

Usage

```
quad.int.ineq(x, y, C.full, b, lambda, d = 10^-5)
```

Arguments

x	independent variable matrix of data to be used in calculating PaC coefficient paths
y	response vector of data to be used in calculating PaC coefficient paths
C.full	complete constraint matrix C (with inequality constraints of the form `C.full*beta >= b`))
b	constraint vector b
lambda	value of lambda
d	very small diagonal term to allow for SVD (default 10^-7)

Value

`beta` the initial beta vector of coefficients to use for the PaC algorithm

Examples

```
random_data = generate.data(n = 500, p = 20, m = 10)
quad_start = quad.int.ineq(random_data$x, random_data$y,
random_data$C.full, random_data$b, lambda = 0.01)
quad_start
```

`transformed` *Transform Data to Fit PaC Implementation (Equality Constraints)*

Description

This function is called internally by `lars.c` to compute the transformed versions of the X, Y, and constraint matrix data, as shown in the PaC paper.

Usage

```
transformed(x, y, C.full, b, lambda, beta0, eps = 10^-8)
```

Arguments

`x`	independent variable matrix of data to be used in calculating PaC coefficient paths
`y`	response vector of data to be used in calculating PaC coefficient paths
`C.full`	complete constraint matrix C (with constraints of the form `C.full*beta=b`)
`b`	constraint vector b
`lambda`	value of lambda
`beta0`	initial guess for beta coefficient vector
`eps`	value close to zero used to verify SVD decomposition. Default is 10^-8

Value

`x` transformed x data to be used in the PaC algorithm

`y` transformed y data to be used in the PaC algorithm

`Y_star` transformed Y* value to be used in the PaC algorithm

`a2` index of A used in the calculation of beta2 (the non-zero coefficients)

`beta1` beta1 values

`beta2` beta2 values

`C` constraint matrix

`C2` subset of constraint matrix corresponding to non-zero coefficients

`active.beta` index of non-zero coefficient values

`beta2.index` index of non-zero coefficient values

References

Gareth M. James, Courtney Paulson, and Paat Rusmevichientong (JASA, 2019) "Penalized and Constrained Optimization." (Full text available at http://www-bcf.usc.edu/~gareth/research/PAC.pdf)

Examples

```
random_data = generate.data(n = 500, p = 20, m = 10)
transform_fit = transformed(random_data$x, random_data$y, random_data$C.full,
random_data$b, lambda = 0.01, beta0 = rep(0,20))
dim(transform_fit$x)
head(transform_fit$y)
dim(transform_fit$C)
transform_fit$active.beta
```

`transformed.ineq` *Transform Data to Fit PaC Implementation for Inequality Constraints*

Description

This function is called internally by `lars.c` to compute the transformed versions of the X, Y, and constraint matrix data, as shown in the PaC paper.

Usage

```
transformed.ineq(x, y, C.full, b, lambda, beta0, eps = 10^-8)
```

Arguments

`x`	independent variable matrix of data to be used in calculating PaC coefficient paths
`y`	response vector of data to be used in calculating PaC coefficient paths
`C.full`	complete constraint matrix C (with inequality constraints of the form `C.full*beta` >= b))
`b`	constraint vector b
`lambda`	value of lambda
`beta0`	initial guess for beta coefficient vector
`eps`	value close to zero used to verify SVD decomposition. Default is 10^-8

Value

`x` transformed x data to be used in the PaC algorithm

`y` transformed y data to be used in the PaC algorithm

`Y_star` transformed Y* value to be used in the PaC algorithm

`a2` index of A used in the calculation of beta2 (the non-zero coefficients) `beta1` beta1 values

`beta2` beta2 values

`C` constraint matrix

`C2` subset of constraint matrix corresponding to non-zero coefficients

`active.beta` index of non-zero coefficient values

`beta2.index` index of non-zero coefficient values

References

Gareth M. James, Courtney Paulson, and Paat Rusmevichientong (JASA, 2019) "Penalized and Constrained Optimization." (Full text available at http://www-bcf.usc.edu/~gareth/research/PAC.pdf)

Examples

```
random_data = generate.data(n = 500, p = 20, m = 10)
transform_fit = transformed.ineq(random_data$x, random_data$y,
random_data$C.full, random_data$b, lambda = 0.01, beta0 = rep(0,20))
dim(transform_fit$x)
head(transform_fit$y)
dim(transform_fit$C)
transform_fit$active.beta
```

Index