Package 'bestsubset'

February 10, 2017

Type Package

Title Tools for best subset selection in regression

Version 1.0.0

Date 2016-01-20

Author Trevor Hastie, Rob Tibshirani, Ryan Tibshirani

Maintainer Ryan Tibshirani <ryantibs@stat.cmu.edu>

Depends glmnet, gurobi

Description An implementation of best subset selection in regression based on a mixed integer quadratic program formulation of the subset selection problem, and the Gurobi mixed integer program optimizer; also, tools for running simulations comparing best subset selection to other common sparse regression estimators such as the lasso and forward stepwise.

License GPL-2

RoxygenNote 5.0.1

R topics documented:

bestsubset-package *Simulations for best subset selection in regression*

Description

Tools for running simulations that compare best subset selection in regression to other common sparse regression estimators such as the lasso and forward stepwise.

Details

The simulation setup is based on the paper: "Best subset selection via a modern optimization lens" by Dimitris Bertsimas, Angela King, and Rahul Mazumder, Annals of Statistics, 44(2), 813-852, 2016.

bs *Best subset selection.*

Description

Compute best subset selection solutions.

Usage

```
bs(x, y, k = 1:min(nrow(x), ncol(x)), intercept = TRUE, time.limit = 100,
  nruns = 50, maxiter = 1000, tol = 1e-04, polish = TRUE,
  verbose = FALSE)
```

Arguments

x	Matrix of predictors, of dimension (say) n x p.
y	Vector of responses, of length (say) n.
k	Sparsity level, i.e., number of nonzero coefficients to allow in the subset regression model; can be a vector, in which case the best subset selection problem is solved for every value of the sparsity level. Default is 1:min(n,p).
intercept	Should an intercept be included in the regression model? Default is TRUE.
verbose	Should intermediate progress be printed out? Default is FALSE.

Details

This function solves best subset selection program:

$$\min_{\beta} \|Y - X\beta\|_2^2 \ \text{ s.t. } \ \|\beta\|_0 \leq k$$

for a response vector Y and predictor matrix X. It uses projected gradient descent to find an approximate solution to the above nonconvex program, and then calls Gurobi's MIO (mixed integer optimization) solver with this approximate solution as a warm start. See references below for the paper by Bertsimas, King, and Mazumder (2016), that describes this algorithm.

Value

A list with the following components:

- beta: matrix of regression coefficients, one column per sparsity level
- status: vector of status strings returned by Gurobi's MIO solver, one element for each sparsity level
- k: vector of sparsity levels
- x, y: the passed x and y
- bx, by: the means of the columns of x, and the mean of y
- intercept: was an intercept included?

Author(s)

Ryan Tibshirani

References

This function utilizes the MIO formulation for subset selection as described in "Best subset selection via a modern optimization lens" by Dimitris Bertsimas, Angela King, and Rahul Mazumder, Annals of Statistics, 44(2), 813-852, 2016. This R implementation is based on Matlab code written by Rahul Mazumder.

Examples

```
# Simulate some simple regression data with the first 5 coefficients
# being nonzero
set.seed(3)
n = 100
p = 20
ntest = 10000
xy.obj = sim.xy(n,p,nval=0,ntest=ntest,s=5,beta.type=2,snr=1)
x = xy.obj$x
y = xy.obj$y
xtest = xy.obj$xtest
mutest = xy.obj$mutest

# Run forward stepwise regression for 8 steps
fs.obj = fs(x,y,intercept=FALSE,maxsteps=8,verbose=TRUE)
fs.beta = coef(fs.obj)
fs.supp = apply(fs.beta != 0, 2, which)

# Solve best subset selection for 8 sparsity levels
bs.obj = bs(x,y,intercept=FALSE,k=1:8,verbose=TRUE)
bs.beta = coef(bs.obj)
bs.supp = apply(bs.beta != 0, 2, which)

# Compare supports of the solutions with 5 and 8 variables
fs.supp[[5]]; bs.supp[[5]]
fs.supp[[8]]; bs.supp[[8]]

# Predict on test data and record risk
fs.pred = predict(fs.obj,newx=xtest)
bs.pred = predict(fs.obj,newx=xtest)
colMeans((fs.pred - mutest)^2)
colMeans((bs.pred - mutest)^2)
```

coef.bs *Coefficient function for bs object.*

Description

Compute coefficients at a particular sparsity level of the best subset selection model.

Usage

```
## S3 method for class 'bs'
coef(object, s, ...)
```

Arguments

object	The bs object, as produced by the bs function.
s	The sparsity level (or vector of sparsity levels) at which coefficients should be computed. If missing, then the default is use all sparsity levels of the passed bs object.
...	Other arguments (currently not used).

coef.fs *Coefficient function for fs object.*

Description

Compute coefficients at a particular step of the forward stepwise path.

Usage

```
## S3 method for class 'fs'
coef(object, s, ...)
```

Arguments

object	The fs object, as produced by the fs function.
s	The step (or vector of steps) of the path at which coefficients should be computed. Can be fractional, in which case interpolation is performed. If missing, then the default is use all steps of the passed fs object.
...	Other arguments (currently not used).

Details

Note that at s = 1, there is one nonzero coefficient, at s = 2, there are two nonzero coefficients, etc. (This differs from the parametrization used in the `coef.fs` function in the R package `selectiveInference`, as the latter function delivers s-1 nonzero coefficients at step s, and was written to be consistent with the natural parametrization for the least angle regression path.)

coef.lasso *Coef function for lasso object.*

Description

Coef function for lasso object.

Usage

```
## S3 method for class 'lasso'
coef(object, s = NULL)
```

fs *Forward stepwise regression.*

Description

Compute the forward stepwise regression path.

Usage

```
fs(x, y, maxsteps = min(ncol(x), 2000), intercept = TRUE,
  normalize = TRUE, verbose = FALSE)
```

Arguments

x	Matrix of predictors, of dimension (say) n x p.
y	Vector of responses, of length (say) n.
maxsteps	Maximum number of steps of the forward stepwise path to compute. Default is min(p,2000).
intercept, normalize	Should an intercept be included in the regression model? Should the predictors be normalized before computing the path? Default is TRUE for both.
verbose	Should intermediate progress be printed out? Default is FALSE.

Details

This function implements forward stepwise regression, adding the predictor at each step that maximizes the absolute correlation between the predictors—once orthogonalized with respect to the current model—and the residual. This entry criterion is standard, and is equivalent to choosing the variable that achieves the biggest drop in RSS at each step; it is used, e.g., by the `step` function in R. Note that, for example, the `lars` package implements a stepwise option (with type="step"), but uses a (mildly) different entry criterion, based on maximal absolute correlation between the original (non-orthogonalized) predictors and the residual.

Value

A list with the following components:

- action, sign: vectors that give the index of the variable added at each step, and the sign of this variable's correlation with the residual upon entry df
- df: vector that gives the (naive) degrees of freedom of the model at each step, i.e., the number of active predictor variables (+ 1 if there is an intercept included)
- beta: matrix of regression coefficients for each step along the path, one column per step
- completepath: a boolean indicating whether the forward stepwise path was run to completion (as opposed to being stopped early because the max number of steps was achieved)
- bls: if the complete path was computed, this is a vector that gives the least squares coefficients of the full regression model
- x, y: the passed x and y
- bx, by: the means of the columns of x, and the mean of y
- intercept, normalize: the passed values for intercept and normalize

Author(s)

Ryan Tibshirani

Examples

```
# Simulate some simple regression data with the first 5 coefficients
# being nonzero
set.seed(3)
n = 100
p = 20
ntest = 10000
xy.obj = sim.xy(n,p,nval=0,ntest=ntest,s=5,beta.type=2,snr=1)
x = xy.obj$x
y = xy.obj$y
xtest = xy.obj$xtest
mutest = xy.obj$mutest

# Run forward stepwise regression for 8 steps
fs.obj = fs(x,y,intercept=FALSE,maxsteps=8,verbose=TRUE)
fs.beta = coef(fs.obj)
fs.supp = apply(fs.beta != 0, 2, which)

# Solve best subset selection for 8 sparsity levels
bs.obj = bs(x,y,intercept=FALSE,k=1:8,verbose=TRUE)
bs.beta = coef(bs.obj)
bs.supp = apply(bs.beta != 0, 2, which)

# Compare supports of the solutions with 5 and 8 variables
fs.supp[[5]]; bs.supp[[5]]
fs.supp[[8]]; bs.supp[[8]]

# Predict on test data and record risk
fs.pred = predict(fs.obj,newx=xtest)
bs.pred = predict(fs.obj,newx=xtest)
colMeans((fs.pred - mutest)^2)
colMeans((bs.pred - mutest)^2)
```

lasso *Lasso and friends.*

Description

This is just a simple wrapper function around the glmnet function in the R package of the same name. Its purpose is to provide a version where the associated coef and predict methods always produce coefficients and predictions at exactly nlambda values, by default. (The glmnet function may produce a path with less than nlambda lambda values, depending on the data.)

Usage

```
lasso(x, y, alpha = 1, nrelax = 10, nlambda = 50,
  lambda.min.ratio = ifelse(nrow(x) < ncol(x), 0.01, 1e-04), lambda = NULL,
  intercept = TRUE, standardize = TRUE)
```

Details

Compute the lasso, ridge regression, or elastic net solutions in regression.

plot.many.sims *Plot the results over several simulation settings.*

Description

Plot the results over several sets of simulations, where the same methods are run over different simulations settings.

Usage

```
## S3 method for class 'many.sims'
plot(file.list, grouping, snr.vec, method.nums = NULL,
  method.names = NULL, type = c("ave", "med"), std = TRUE,
  tuning = c("validation", "oracle"), fig.dir = ".", file.name = NULL,
  w = 5.5, h = 5.5, mar = NULL, pve = TRUE, cols = 1:8, main = NULL,
  cex.main = 1.25, legend.pos = "bottomright")
```

Arguments

file.list	vector of strings that point to saved sim objects (each object produced by a call to sim.master).
grouping	integer or factor vector indicating the grouping to use for the simulations. Within each group, the relative test error achieved by each method is plotted across the available SNR levels.
snr.vec	Vector giving the SNR levels considered within each group.
method.nums	the indices of the methods that should be plotted. Default is NULL, in which case all methods are plotted.
method.names	the names of the methods that should be plotted. Default is NULL, in which case the names are extracted from the sim objects.

type	Either "ave" or "med", indicating whether the average or median of the relative test error metric should be displayed. Default is "ave".
std	Should standard errors be displayed (in parantheses)? When type is set to "med", the median absolute deviations are shown in place of the standard errors. Default is TRUE.
tuning	one of "validation" or "oracle", indicating whether the tuning parameter for each method in each simulation setting should be chosen according to minimizing validation error, or according to minimizing risk on test set. Default is "validation".
fig.dir	The figure directory to use. Default is ".".
file.name	Vector of strings, giving the file names to use for the saved figures. Default is NULL, in which case the names "sim1", "sim2", etc. are used. (Extensions of "pdf" are always appended to the given file names.)
w, h	the width and height (in inches) for the plots. Defaults are 5.5 for both.
mar	the margins to use for the plots. Default is NULL, in which case the margins are set automatically (depending on whether not main is NULL).
pve	Should the (population) proportion of variance explained be shown, corresponding to each SNR level under consideration? This is snr/(1+snr). Default is TRUE.
cols, main, cex.main, legend.pos	graphical parameters.

`plot.sim` *Plot function for sim object.*

Description

Plot the results of a set of simulations, stored in an object of class sim (produced by `sim.master`).

Usage

```
## S3 method for class 'sim'
plot(x, method.nums = 1:length(x$err.rel.ave),
  method.names = NULL, type = c("ave", "med"), std = TRUE, cols = 1:8,
  main = NULL, cex.main = 1.25, legend.pos = c("topright"),
  make.pdf = FALSE, fig.dir = ".", file.name = "sim", w = 5.5,
  h = 5.5, mar = NULL)
```

Arguments

x	The sim object.
method.nums	the indices of the methods that should be plotted. Default is to 1:length(x$err.rel.ave), which plots all methods.
method.names	the names of the methods that should be plotted. Default is NULL, in which case the names are extracted from the sim object.
type	Either "ave" or "med", indicating whether the average or median of the relative test error metric should be displayed. Default is "ave".

`std` Should standard errors be displayed (in parantheses)? When type is set to "med", the median absolute deviations are shown in place of the standard errors. Default is TRUE.

`cols, main, cex.main, legend.pos`
graphical parameters.

`make.pdf` Should a pdf be produced? Default is FALSE.

`fig.dir, file.name`
The figure directory and file name to use, only when make.pdf is TRUE. Defaults are "." and "sim". (An extension of "pdf" is always appended to the given file name.)

`w, h` the width and height (in inches) for the plot, used only when make.pdf is TRUE. Defaults are 5.5 for both.

`mar` the margins to use for the plot. Default is NULL, in which case the margins are set automatically (depending on whether not main is NULL).

`predict.bs` *Predict function for bs object.*

Description

Predict the response from a new set of predictor variables, using the coefficients from a particular step of the forward stepwise path.

Usage

```
## S3 method for class 'bs'
predict(object, newx, s, ...)
```

Arguments

`object` The vs path object, as produced by the vs function.

`newx` Matrix of new predictor variables at which predictions should be made; if missing, the original (training) predictors are used.

`s` The sparsity level (or vector of sparsity levels) at which coefficients should be computed. If missing, then the default is use all sparsity levels of the passed bs object.

`...` Other arguments (currently not used).

predict.fs *Predict function for fs object.*

Description

Predict the response from a new set of predictor variables, using the coefficients from a particular step of the forward stepwise path.

Usage

```
## S3 method for class 'fs'
predict(object, newx, s, ...)
```

Arguments

object	The fs path object, as produced by the fs function.
newx	Matrix of new predictor variables at which predictions should be made; if missing, the original (training) predictors are used.
s	The step (or vector of steps) of the path at which coefficients should be computed. Can be fractional, in which case interpolation is performed. If missing, then the default is use all steps of the passed fs object.
...	Other arguments (currently not used).

Details

Note that at s = 1, there is one nonzero coefficient, at s = 2, there are two nonzero coefficients, etc. (This differs from the parametrization used in the `coef.fs` function in the R package `selectiveInference`, as the latter function delivers s-1 nonzero coefficients at step s, and was written to be consistent with the natural parametrization for the least angle regression path.)

predict.lasso *Predict function for lasso object.*

Description

Predict function for lasso object.

Usage

```
## S3 method for class 'lasso'
predict(object, newx, s = NULL, type = "link")
```

print.sim *Print function for sim object.*

Description

Summarize and print the results of a set of simulations, stored an object of class sim (produced by sim.master).

Usage

```
## S3 method for class 'sim'
print(x, type = c("ave", "med"), std = TRUE, digits = 3,
  ...)
```

Arguments

x	The sim object.
type	Either "ave" or "med", indicating whether the average or median of the relative test error metric should be displayed. Default is "ave".
std	Should standard errors be displayed (in parantheses)? When type is set to "med", the median absolute deviations are shown in place of the standard errors. Default is TRUE.
digits	Number of digits to display. Default is 3.
...	Other arguments (currently not used).

print.tex *Print function for latex-style tables.*

Description

Print a given table in format digestable by latex.

Usage

```
## S3 method for class 'tex'
print(tab, tab.se = NULL, digits = 3, file = NULL,
  align = "l")
```

`sim.master` *Master function for running simulations.*

Description

Run a set of simulations with the specified configuration.

Usage

```
sim.master(n, p, nval, ntest, reg.funs, nrep = 50, seed = NULL,
  verbose = FALSE, file = NULL, file.rep = 5, rho = 0, s = 5,
  beta.type = 1, snr = 1)
```

Arguments

`n, p`	The number of training observations, and the number of predictors.
`nval, ntest`	The number of validation observations, and the number of testing observations.
`reg.funs`	This is a list of functions, representing the regression procedures to be used (evaluated) in the simulation. Each element of the list must be a function that takes x, y (the training predictor matrix and response vector) as its only two (mandatory) arguments, and must return an object with associated coef and predict methods. The coef method must take obj (the returned object) and return a matrix of coefficients, with one column per tuning parameter value inherent to the regression method. The predict method must take obj, newx (the returned object and a new predictor matrix) and return a matrix of predictions, again with one column per tuning parameter value inherent to the regression method.
`seed`	Seed to be set for the overall random number generation, i.e., set before repetitions are begun (for reproducibility of the simulation results). Default is NULL, which effectively sets no seed.
`verbose`	Should intermediate progress be printed out? Default is FALSE.
`file, file.rep`	Name of a file to which simulation results are saved (using saveRDS), and a number of repetitions after which intermediate results are saved. Setting file to NULL is interpreted to mean that no simulations results should be saved; setting file.rep to 0 is interpreted to mean that simulations results should be saved at the very end, i.e., no intermediate saving. Defaults are NULL and 5, respectively.
`rho, s, beta.type, snr`	Arguments to pass to `sim.xy`; see the latter's help file for details.
`Number`	of repetitions of which to average the results. Default is 50.

Value

A list with components err.train, err.val, err.test, err.rel, risk, nzs, opt for the training error, validation error, test error, relative test error (test error divided by sigma^2), risk, number of selected nonzero coefficients, and optimism (difference in test and training errors). These are each lists of length N, where N is the number of regression methods under consideration (the length of reg.funs). The ith element of each list is then a matrix of dimension nrep x m, where m the number of tuning parameters inherent to the ith method. The returned components err.train.ave, err.val.ave, err.test.ave, err.rel.ave, risk.ave, nzs.ave, opt.ave return the averages of the training error, validation error, etc. over the nrep repetitions. Similarly for the components with postfixes .std, .med, .mad, which return

the standard deviation, median, and median absolute deviation, respectively. The returned components with postfixes .tun.val and .tun.orc are matrices of dimension N x nrep, which return the training error, validation error, etc. when the tuning parameter for each regression method in each repetition is chosen by validation tuning (best validation error) or by oracle tuning (best average risk across all the repetitions).

Author(s)

Trevor Hastie, Robert Tibshirani, Ryan Tibshirani

References

The structure of this simulation code based on that from the `conformalInference` package.

See Also

`sim.xy`

Examples

```
# Simulate in simple regression setting with the first 5 coefficients
# being nonzero
set.seed(0)
n = 100
p = 20
nval = n
ntest = 10000

# Check for gurobi package
if (!require("gurobi",quietly=TRUE)) {
  stop("Package gurobi not installed (required here)!")
}

# Regression functions: lasso, forward stepwise, and best subset selection
reg.funs = list()
reg.funs[["Lasso"]] = function(x,y) lasso(x,y,intercept=FALSE,nlam=50)
reg.funs[["Stepwise"]] = function(x,y) fs(x,y,intercept=FALSE)
reg.funs[["Best subset"]] = function(x,y) bs(x,y,intercept=FALSE)

# Run the master simulation function, then print results
sim.obj.hisnr = sim.master(n,p,nval,ntest,reg.funs=reg.funs,nrep=10,seed=0,
                     beta.type=2,snr=1,verbose=TRUE)
sim.obj.hisnr

# Repeat, but now for a lower signal-to-noise ratio
sim.obj.losnr = sim.master(n,p,nval,ntest,reg.funs=reg.funs,nrep=10,
                           seed=0,beta.type=2,snr=0.1,verbose=TRUE)
sim.obj.losnr

# Plot simulation results side by side
par(mfrow=c(1,2))
plot(sim.obj.hisnr, main="SNR = 1", legend.pos="topright")
plot(sim.obj.losnr, main="SNR = 0.1", legend.pos="topleft")
```

`sim.xy` *Predictors and responses generation.*

Description

Generate a predictor matrix x, and response vector y, following a specified setup. Actually, three pairs of predictors and responses are generated: one for training, one for validation, and one for testing.

Usage

```
sim.xy(n, p, nval, ntest, rho = 0, s = 5, beta.type = 1, snr = 1)
```

Arguments

`n, p`	The number of training observations, and the number of predictors.
`nval, ntest`	The number of validation observations, and the number of testing observations.
`rho`	Parameter that drives pairwise correlations of the predictor variables; specifically, predictors i and j have population correlation rho^abs(i-j). Default is 0.
`s`	number of nonzero coefficients in the underlying regression model. Default is 5. (Ignored if beta.type is 4, in which case the number of nonzero coefficients is 6; and if beta.type is 5, it is interpreted as a the number of strongly nonzero coefficients in a weak sparsity model.)
`beta.type`	Integer taking values in between 1 and 5, used to specify the pattern of nonzero coefficients in the underlying regression model; see details below. Default is 1.
`snr`	Desired signal-to-noise ratio (SNR), i.e., var(mu)/sigma^2 where mu is mean and sigma^2 is the error variance. The error variance is set so that the given SNR is achieved. Default is 1.

Details

The predictors are normal with covariance sigma^2 * Sigma, where sigma^2 is set according to the desired signal-to-noise ratio, and Sigma has (i,j)th entry rho^abs(i-j). The first 4 options for the nonzero pattern of the underlying regression coefficients beta follow the simulation setup in Bertsimas, King, and Mazumder (2016), and the last is a weak sparsity option:

- 1: beta has s components of 1, occurring at (roughly) equally-spaced indices in between 1 and p
- 2: beta has its first s components equal to 1
- 3: beta has its first s components taking nonzero values, where the decay in a linear fashion from 10 to 0.5
- 4: beta has its first 6 components taking the nonzero values -10,-6, -2,2,6,10
- 5: beta has its first s components equal to 1, and the rest decaying to zero at an exponential rate

Value

A list with the following components: x, y, xval, yval, xtest, ytest, mutest, beta, and sigma.

Author(s)

Trevor Hastie, Rob Tibshirani, Ryan Tibshirani

References

Simulation setup based on "Best subset selection via a modern optimization lens" by Dimitris Bertsimas, Angela King, and Rahul Mazumder, Annals of Statistics, 44(2), 813-852, 2016.

Examples

```
# Simulate some simple regression data with the first 5 coefficients
# being nonzero
set.seed(3)
n = 100
p = 20
ntest = 10000
xy.obj = sim.xy(n,p,nval=0,ntest=ntest,s=5,beta.type=2,snr=1)
x = xy.obj$x
y = xy.obj$y
xtest = xy.obj$xtest
mutest = xy.obj$mutest

# Run forward stepwise regression for 8 steps
fs.obj = fs(x,y,intercept=FALSE,maxsteps=8,verbose=TRUE)
fs.beta = coef(fs.obj)
fs.supp = apply(fs.beta != 0, 2, which)

# Solve best subset selection for 8 sparsity levels
bs.obj = bs(x,y,intercept=FALSE,k=1:8,verbose=TRUE)
bs.beta = coef(bs.obj)
bs.supp = apply(bs.beta != 0, 2, which)

# Compare supports of the solutions with 5 and 8 variables
fs.supp[[5]]; bs.supp[[5]]
fs.supp[[8]]; bs.supp[[8]]

# Predict on test data and record risk
fs.pred = predict(fs.obj,newx=xtest)
bs.pred = predict(fs.obj,newx=xtest)
colMeans((fs.pred - mutest)^2)
colMeans((bs.pred - mutest)^2)
```

Index