Package ‘selectiveInference’

September 7, 2019

Type Package
Title Tools for Post-Selection Inference
Version 1.2.5
Date 2019-09-04
Author Ryan Tibshirani, Rob Tibshirani, Jonathan Taylor, Joshua Loftus, Stephen Reid, Jelena Markovic
Maintainer Rob Tibshirani <tibs@stanford.edu>
Depends glmnet, intervals, survival, adaptMCMC, MASS
Suggests Rmpfr
Description New tools for post-selection inference, for use with forward stepwise regression, least angle regression, the lasso, and the many means problem. The lasso function implements Gaussian, logistic and Cox survival models.
License GPL-2
RoxygenNote 5.0.1
LinkingTo Rcpp
Imports Rcpp
NeedsCompilation yes
Repository CRAN
Date/Publication 2019-09-07 07:00:02 UTC

R topics documented:

debiasingMatrix ......................................................... 2
estimateSigma .......................................................... 4
factorDesign ............................................................. 5
fixedLassoInf ............................................................ 6
forwardStop .............................................................. 12
fs ................................................................. 13
fsInf ............................................................ 15
groupfs ............................................................. 18
debiasingMatrix

Find an approximate inverse of a non-negative definite matrix.

**Description**

Find some rows of an approximate inverse of a non-negative definite symmetric matrix by solving optimization problem described in Javanmard and Montanari (2013). Can be used for approximate Newton step from some consistent estimator (such as the LASSO) to find a debiased solution.

**Usage**

```r
debiasingMatrix(Xinfo,  
is_wide,  
nsample,  
rows,  
verbose=FALSE,  
bound=NULL,  
linesearch=TRUE,  
scaling_factor=1.5,  
max_active=NULL,  
max_try=10,  
warn_kkt=FALSE,  
max_iter=50,  
kkt_stop=TRUE,  
parameter_stop=TRUE,  
objective_stop=TRUE,  
kkt_tol=1.e-4,  
parameter_tol=1.e-4,  
objective_tol=1.e-4)
```
Arguments

- **Xinfo**
  Either a non-negative definite matrix $S=\mathsf{t}(X)$ is\_wide is TRUE, then Xinfo should be X, otherwise it should be $S$.

- **is\_wide**
  Are we solving for rows of the debiasing matrix assuming it is a wide matrix so that Xinfo=X and the non-negative definite matrix of interest is $\mathsf{t}(X)$?

- **nsample**
  Number of samples used in forming the cross-covariance matrix. Used for default value of the bound parameter.

- **rows**
  Which rows of the approximate inverse to compute.

- **verbose**
  Print out progress as rows are being computed.

- **bound**
  Initial bound parameter for each row. Will be changed if linesearch is TRUE.

- **linesearch**
  Run a line search to find as small as possible a bound parameter for each row?

- **scaling\_factor**
  In the linesearch, the bound parameter is either multiplied or divided by this factor at each step.

- **max\_active**
  How large an active set to consider in solving the problem with coordinate descent. Defaults to max(50, 0.3*nsample).

- **max\_try**
  How many tries in the linesearch.

- **warn\_kkt**
  Warn if the problem does not seem to be feasible after running the coordinate descent algorithm.

- **max\_iter**
  How many full iterations to run of the coordinate descent for each value of the bound parameter.

- **kkt\_stop**
  If TRUE, check to stop coordinate descent when KKT conditions are approximately satisfied.

- **parameter\_stop**
  If TRUE, check to stop coordinate descent based on relative convergence of parameter vector, checked at geometrically spaced iterations $2^k$.

- **objective\_stop**
  If TRUE, check to stop coordinate descent based on relative decrease of objective value, checked at geometrically spaced iterations $2^k$.

- **kkt\_tol**
  Tolerance value for assessing whether KKT conditions for solving the dual problem and feasibility of the original problem.

- **parameter\_tol**
  Tolerance value for assessing convergence of the problem using relative convergence of the parameter.

- **objective\_tol**
  Tolerance value for assessing convergence of the problem using relative decrease of the objective.

Details

This function computes an approximate inverse as described in Javanmard and Montanari (2013), specifically display (4). The problem is solved by considering a dual problem which has an objective similar to a LASSO problem and is solvable by coordinate descent. For some values of bound the original problem may not be feasible, in which case the dual problem has no solution. An attempt to detect this is made by stopping when the active set grows quite large, determined by max\_active.

Value

- **M**
  Rows of approximate inverse of $\Sigma$. 
estimateSigma

Estimate the noise standard deviation in regression

Description

Estimates the standard deviation of the noise, for use in the selectiveInference package

Usage

estimateSigma(x, y, intercept=TRUE, standardize=TRUE)

Arguments

- `x`: Matrix of predictors (n by p)
- `y`: Vector of outcomes (length n)
- `intercept`: Should glmnet be run with an intercept? Default is TRUE
- `standardize`: Should glmnet be run with standardized predictors? Default is TRUE

Details

This function estimates the standard deviation of the noise, in a linear regression setting. A lasso regression is fit, using cross-validation to estimate the tuning parameter lambda. With sample size n, yhat equal to the predicted values and df being the number of nonzero coefficients from the lasso fit, the estimate of sigma is \( \sqrt{\text{sum}((y-yhat)^2) / (n-df-1)} \). Important: if you are using glmnet to compute the lasso estimate, be sure to use the settings for the "intercept" and "standardize" arguments in glmnet and estimateSigma. Same applies to fs or lar, where the argument for standardization is called "normalize".
factorDesign

Value

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>sigmahat</td>
<td>The estimate of sigma</td>
</tr>
<tr>
<td>df</td>
<td>The degrees of freedom of lasso fit used</td>
</tr>
</tbody>
</table>

Author(s)

Ryan Tibshirani, Rob Tibshirani, Jonathan Taylor, Joshua Loftus, Stephen Reid

References


Examples

```r
set.seed(33)
n = 50
p = 10
sigma = 1
x = matrix(rnorm(n*p),n,p)
beta = c(3,2,rep(0,p-2))
y = x%*%beta + sigma*rnorm(n)

# run forward stepwise
fsfit = fs(x,y)

# estimate sigma
sigmahat = estimateSigma(x,y)$sigmahat

# run sequential inference with estimated sigma
out = fsInf(fsfit,sigma=sigmahat)
out
```
Value

List containing

- Design matrix, the first columns contain any numeric variables from the original data frame.

- Group membership indicator for expanded matrix.

Examples

```r
## Not run:
fd = factorDesign(warpbreaks)
y = rnorm(nrow(fd$x))
fit = groupfs(fd$x, y, fd$index, maxsteps=2, intercept=F)
pvals = groupfsInf(fit)

## End(Not run)
```

---

**fixedLassoInf**

*Inference for the lasso, with a fixed lambda*

Description

Compute p-values and confidence intervals for the lasso estimate, at a fixed value of the tuning parameter lambda.

Usage

```r
fixedLassoInf(x, y, beta, lambda, family = c("gaussian", "binomial", "cox"), intercept=TRUE, add.targets=NULL, status=NULL, sigma=NULL, alpha=0.1, type=c("partial","full"), tol.beta=1e-5, tol.kkt=0.1, gridrange=c(-100,100), bits=NULL, verbose=FALSE, linesearch.try=10)
```
**fixedLassoInf**

**Arguments**

- **x**: Matrix of predictors (n by p);
- **y**: Vector of outcomes (length n);
- **beta**: Estimated lasso coefficients (e.g., from glmnet). This is of length p (so the intercept is not included as the first component).

Be careful! This function uses the "standard" lasso objective

\[
\frac{1}{2} \| y - x\beta \|_2^2 + \lambda \| \beta \|_1.
\]

In contrast, glmnet multiplies the first term by a factor of 1/n. So after running glmnet, to extract the beta corresponding to a value lambda, you need to use

\[
\text{beta} = \text{coef(obj, s=lambda/n)}[-1]
\]

where obj is the object returned by glmnet (and [-1] removes the intercept, which glmnet always puts in the first component).

- **lambda**: Value of lambda used to compute beta. See the above warning
- **family**: Response type: "gaussian" (default), "binomial", or "cox" (for censored survival data)
- **sigma**: Estimate of error standard deviation. If NULL (default), this is estimated using the mean squared residual of the full least squares fit when n >= 2p, and using the standard deviation of y when n < 2p. In the latter case, the user should use the `estimateSigma` function for a more accurate estimate. Not used for family= "binomial", or "cox"
- **alpha**: Significance level for confidence intervals (target is miscoverage alpha/2 in each tail)
- **intercept**: Was the lasso problem solved (e.g., by glmnet) with an intercept in the model? Default is TRUE. Must be TRUE for "binomial" family. Not used for "cox" family, where no intercept is assumed.
- **add.targets**: Optional vector of predictors to be included as targets of inference, regardless of whether or not they are selected by the lasso. Default is NULL.
- **status**: Censoring status for Cox model; 1=failure 0=censored
- **type**: Contrast type for p-values and confidence intervals: default is "partial"—meaning that the contrasts tested are the partial population regression coefficients, within the active set of predictors; the alternative is "full"—meaning that the full population regression coefficients are tested. The latter does not make sense when p > n.
- **tol.beta**: Tolerance for determining if a coefficient is zero
- **tol.kkt**: Tolerance for determining if an entry of the subgradient is zero
- **grid.range**: Grid range for constructing confidence intervals, on the standardized scale
- **bits**: Number of bits to be used for p-value and confidence interval calculations. Default is NULL, in which case standard floating point calculations are performed. When not NULL, multiple precision floating point calculations are performed with the specified number of bits, using the R package `Rmpfr` (if this package is not installed, then a warning is thrown, and standard floating point calculations are pursued). Note: standard double precision uses 53 bits so, e.g., a
choice of 200 bits uses about 4 times double precision. The confidence interval computation is sometimes numerically challenging, and the extra precision can be helpful (though computationally more costly). In particular, extra precision might be tried if the values in the output columns of tailarea differ noticeably from alpha/2.

verbose
Print out progress along the way? Default is FALSE

linesearch.try
When running type="full" (i.e. debiased LASSO) how many attempts in the line search?

Details
This function computes selective p-values and confidence intervals for the lasso, given a fixed value of the tuning parameter lambda. Three different response types are supported: gaussian, binomial and Cox. The confidence interval construction involves numerical search and can be fragile: if the observed statistic is too close to either end of the truncation interval (vlo and vup, see references), then one or possibly both endpoints of the interval of desired coverage cannot be computed, and default to +/- Inf. The output tailarea gives the achieved Gaussian tail areas for the reported intervals—these should be close to alpha/2, and can be used for error-checking purposes.

Important!: Before running glmnet (or some other lasso-solver) x should be centered, that is x <- scale(X,TRUE,FALSE). In addition, if standardization of the predictors is desired, x should be scaled as well: x <- scale(x,TRUE,TRUE). Then when running glmnet, set standardize=F. See example below.

The penalty.factor facility in glmnet— allowing different penalties lambda for each predictor, is not yet implemented in fixedLassoInf. However you can finesse this— see the example below. One caveat- using this approach, a penalty factor of zero (forcing a predictor in) is not allowed.

Note that the coefficients and standard errors reported are unregularized. Eg for the Gaussian, they are the usual least squares estimates and standard errors for the model fit to the active set from the lasso.

Value

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>Type of coefficients tested (partial or full)</td>
</tr>
<tr>
<td>lambda</td>
<td>Value of tuning parameter lambda used</td>
</tr>
<tr>
<td>pv</td>
<td>One-sided P-values for active variables, uses the fact we have conditioned on the sign.</td>
</tr>
<tr>
<td>ci</td>
<td>Confidence intervals</td>
</tr>
<tr>
<td>tailarea</td>
<td>Realized tail areas (lower and upper) for each confidence interval</td>
</tr>
<tr>
<td>vlo</td>
<td>Lower truncation limits for statistics</td>
</tr>
<tr>
<td>vup</td>
<td>Upper truncation limits for statistics</td>
</tr>
<tr>
<td>vmat</td>
<td>Linear contrasts that define the observed statistics</td>
</tr>
<tr>
<td>y</td>
<td>Vector of outcomes</td>
</tr>
<tr>
<td>vars</td>
<td>Variables in active set</td>
</tr>
<tr>
<td>sign</td>
<td>Signs of active coefficients</td>
</tr>
<tr>
<td>alpha</td>
<td>Desired coverage (alpha/2 in each tail)</td>
</tr>
<tr>
<td>sigma</td>
<td>Value of error standard deviation (sigma) used</td>
</tr>
<tr>
<td>call</td>
<td>The call to fixedLassoInf</td>
</tr>
</tbody>
</table>
Author(s)
Ryan Tibshirani, Rob Tibshirani, Jonathan Taylor, Joshua Loftus, Stephen Reid

References

Examples
```r
set.seed(43)
n = 50
p = 10
sigma = 1

x = matrix(rnorm(n*p),n,p)
x = scale(x,TRUE,TRUE)

beta = c(3,2,rep(0,p-2))
y = x%*%beta + sigma*rnorm(n)

# first run glmnet
gfit = glmnet(x,y,standardize=FALSE)

# extract coef for a given lambda; note the 1/n factor!
# (and we don't save the intercept term)
lambda = .8
beta = coef(gfit, x=x, y=y, s=lambda/n, exact=TRUE)[-1]

# compute fixed lambda p-values and selection intervals
out = fixedLassoInf(x,y,beta,lambda,sigma=sigma)
out

## as above, but use lar function instead to get initial
## lasso fit (should get same results)
lfit = lar(x,y,normalize=FALSE)
beta = coef(lfit, s=lambda, mode="lambda")
out2 = fixedLassoInf(x, y, beta, lambda, sigma=sigma)
out2

## mimic different penalty factors by first scaling x
set.seed(43)
n = 50
p = 10
sigma = 1

x = matrix(rnorm(n*p),n,p)
x=scale(x,TRUE,TRUE)
```
beta = c(3,2,rep(0,p-2))
y = x%*%beta + sigma*rnorm(n)
pf=c(rep(1,7),rep(.1,3)) #define penalty factors
pf=pf/sum(pf) # penalty factors should be rescaled so they sum to p
xs=scale(x,FALSE,pf) #scale cols of x by penalty factors

# first run glmnet
gfit = glmnet(xs, y, standardize=FALSE)

# extract coef for a given lambda; note the 1/n factor!
# (and we don't save the intercept term)
lambda = .8
beta_hat = coef(gfit, x=xs, y=y, s=lambda/n, exact=TRUE)[-1]

# compute fixed lambda p-values and selection intervals
out = fixedLassoInf(xs,y,beta_hat,lambda)

# logistic model
set.seed(43)

n = 50
p = 10
sigma = 1

x = matrix(rnorm(n*p),n,p)
x=scale(x,TRUE,TRUE)

beta = c(3,2,rep(0,p-2))
y = x%*%beta + sigma*rnorm(n)
y=1*(y>mean(y))

# first run glmnet
gfit = glmnet(x,y,standardize=FALSE,family="binomial")

# extract coef for a given lambda; note the 1/n factor!
# (and here we DO include the intercept term)
lambda = .8
beta_hat = coef(gfit, x=x, y=y, s=lambda/n, exact=TRUE)

# compute fixed lambda p-values and selection intervals
out = fixedLassoInf(x,y,beta_hat,lambda)

# Cox model
set.seed(43)

n = 50
p = 10
sigma = 1
\[
x = \text{matrix}(\text{rnorm}(n*p), n, p)
x = \text{scale}(x, \text{TRUE, TRUE})
\]
\[
\beta = (3,2,\text{rep}(0,p-2))
tim = \text{as.vector}(x \times \beta + \text{sigma} \times \text{rnorm}(n))
tim = \text{tim-min(tim)}+1
status = \text{sample}(c(0,1), \text{size}=n, \text{replace=TRUE})
\]
# first run glmnet

\[
y = \text{Surv}(\text{tim}, \text{status})
gfit = \text{glmnet}(x, y, \text{standardize=FALSE, family="cox"})
\]
# extract coef for a given lambda; note the 1/n factor!

\[
\lambda = 1.5
\beta_{\hat{}} = \text{as.numeric}(\text{coef}(gfit, x=x, y=y, s=\lambda/n, \text{exact=TRUE}))
\]
# compute fixed lambda p-values and selection intervals
out = \text{fixedLassoInf}(x, \text{tim, beta}_{\hat{}}, \lambda, \text{status=status, family="cox"})
out

# Debiased lasso or "full"

\[
n = 50
p = 100
\sigma = 1
\]
\[
x = \text{matrix}(\text{rnorm}(n*p), n, p)
x = \text{scale}(x, \text{TRUE, TRUE})
\]
\[
\beta = (3,2,\text{rep}(0,p-2))
y = x \times \beta + \text{sigma} \times \text{rnorm}(n)
\]
# first run glmnet
\[
gfit = \text{glmnet}(x, y, \text{standardize=FALSE, intercept=FALSE})
\]
# extract coef for a given lambda; note the 1/n factor!
# (and we don't save the intercept term)
\[
\lambda = 2.8
\beta = \text{coef}(gfit, x=x, y=y, s=\lambda/n, \text{exact=TRUE})[-1]
\]
# compute fixed lambda p-values and selection intervals
out = \text{fixedLassoInf}(x, y, \beta, \lambda, \sigma=\sigma, \text{type='full', intercept=FALSE})
out

# When n > p and "full" we use the full inverse
# instead of Javanmard and Montanari's approximate inverse

\[
n = 200
p = 50
\sigma = 1
\]
x = matrix(rnorm(n*p),n,p)
x = scale(x,TRUE,TRUE)

beta = c(3,2,rep(0,p-2))
y = x%*%beta + sigma*rnorm(n)

# first run glmnet
gfit = glmnet(x, y, standardize=FALSE, intercept=FALSE)

# extract coef for a given lambda; note the 1/n factor!
# (and we don't save the intercept term)
lambda = 2.8
beta = coef(gfit, x=x, y=y, s=lambda/n, exact=TRUE)[-1]

# compute fixed lambda p-values and selection intervals
out = fixedLassoInf(x, y, beta, lambda, sigma=sigma, type='full', intercept=FALSE)
out

---
forwardStop

ForwardStop rule for sequential p-values

Description
Computes the ForwardStop sequential stopping rule of G’Sell et al (2014)

Usage
forwardStop(pv, alpha=0.1)

Arguments

pv Vector of **sequential** p-values, for example from fsInf or larInf
alpha Desired type FDR level (between 0 and 1)

Details
Computes the ForwardStop sequential stopping rule of G’Sell et al (2014). Guarantees FDR control at the level alpha, for independent p-values.

Value
Step number for sequential stop.

Author(s)
Ryan Tibshirani, Rob Tibshirani, Jonathan Taylor, Joshua Loftus, Stephen Reid
**References**


**Examples**

```r
set.seed(33)
n = 50
p = 10
sigma = 1
x = matrix(rnorm(n*p),n,p)
beta = c(3,2,rep(0,p-2))
y = x%*%beta + sigma*rnorm(n)

# run forward stepwise
fsfit = fs(x,y)

# compute sequential p-values and confidence intervals
# (sigma estimated from full model)
out = fsInf(fsfit)
out

# estimate optimal stopping point
forwardStop(out$pv, alpha=.10)
```

---

**fs**

*Forward stepwise regression*

**Description**

This function implements forward stepwise regression, for use in the selectiveInference package

**Usage**

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

**Arguments**

- **x**: Matrix of predictors (n by p)
- **y**: Vector of outcomes (length n)
- **maxsteps**: Maximum number of steps to take
- **intercept**: Should an intercept be included on the model? Default is TRUE
- **normalize**: Should the predictors be normalized? Default is TRUE. (Note: this argument has no real effect on model selection since forward stepwise is scale invariant already; however, it is included for completeness, and to match the interface for the lar function)
- **verbose**: Print out progress along the way? 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

- `action`: Vector of predictors in order of entry
- `sign`: Signs of coefficients of predictors, upon entry
- `df`: Degrees of freedom of each active model
- `beta`: Matrix of regression coefficients for each model along the path, one column per model
- `completepath`: Was the complete stepwise path computed?
- `bls`: If completepath is TRUE, the full least squares coefficients
- `Gamma`: Matrix that captures the polyhedral selection at each step
- `nk`: Number of polyhedral constraints at each step in path
- `vreg`: Matrix of linear contrasts that gives coefficients of variables to enter along the path
- `x`: Matrix of predictors used
- `y`: Vector of outcomes used
- `bx`: Vector of column means of original x
- `by`: Mean of original y
- `sx`: Norm of each column of original x
- `intercept`: Was an intercept included?
- `normalize`: Were the predictors normalized?
- `call`: The call to fs

Author(s)

Ryan Tibshirani, Rob Tibshirani, Jonathan Taylor, Joshua Loftus, Stephen Reid

See Also

`fsInf`, `predict.fs`, `coef.fs`, `plot.fs`
Examples

```r
codeset.seed(33)
n = 50
p = 10
sigma = 1
x = matrix(rnorm(n*p),n,p)
beta = c(3,2,rep(0,p-2))
y = x%*%beta + sigma*rnorm(n)

# run forward stepwise, plot results
fsfit = fs(x,y)
plot(fsfit)

# compute sequential p-values and confidence intervals
# (sigma estimated from full model)
out = fsInf(fsfit)
out
```

fsInf

Selective inference for forward stepwise regression

Description

Computes p-values and confidence intervals for forward stepwise regression

Usage

```r
fsInf(obj, sigma=NULL, alpha=0.1, k=NULL, type=c("active","all","aic"),
gridrange=c(-100,100), bits=NULL, mult=2, ntimes=2, verbose=FALSE)
```

Arguments

- **obj**: Object returned by `fs` function
- **sigma**: Estimate of error standard deviation. If NULL (default), this is estimated using the mean squared residual of the full least squares fit when n >= 2p, and using the standard deviation of y when n < 2p. In the latter case, the user should use `estimateSigma` function for a more accurate estimate
- **alpha**: Significance level for confidence intervals (target is miscoverage alpha/2 in each tail)
- **k**: See "type" argument below. Default is NULL, in which case k is taken to be the number of steps computed in the forward stepwise path
- **type**: Type of analysis desired: with "active" (default), p-values and confidence intervals are computed for each predictor as it is entered into the active step, all the way through k steps; with "all", p-values and confidence intervals are computed for all variables in the active model after k steps; with "aic", the number of steps k is first estimated using a modified AIC criterion, and then the same type of analysis as in "all" is carried out for this particular value of k.
Note that the AIC scheme is defined to choose a number of steps $k$ after which the AIC criterion increases $n_{\text{times}}$ in a row, where $n_{\text{times}}$ can be specified by the user (see below). Under this definition, the AIC selection event is characterizable as a polyhedral set, and hence the extra conditioning can be taken into account exactly. Also note that an analogous BIC scheme can be specified through the mult argument (see below)

**gridrange**
Grid range for constructing confidence intervals, on the standardized scale

**bits**
Number of bits to be used for p-value and confidence interval calculations. Default is NULL, in which case standard floating point calculations are performed. When not NULL, multiple precision floating point calculations are performed with the specified number of bits, using the R package Rmpfr (if this package is not installed, then a warning is thrown, and standard floating point calculations are pursued). Note: standard double precision uses 53 bits so, e.g., a choice of 200 bits uses about 4 times double precision. The confidence interval computation is sometimes numerically challenging, and the extra precision can be helpful (though computationally more costly). In particular, extra precision might be tried if the values in the output columns of tailarea differ noticeably from alpha/2.

**mult**
Multiplier for the AIC-style penalty. Hence a value of 2 (default) gives AIC, whereas a value of $\log(n)$ would give BIC

**ntimes**
Number of steps for which AIC-style criterion has to increase before minimizing point is declared

**verbose**
Print out progress along the way? Default is FALSE

**Details**
This function computes selective p-values and confidence intervals (selection intervals) for forward stepwise regression. The default is to report the results for each predictor after its entry into the model. See the "type" argument for other options. The confidence interval construction involves numerical search and can be fragile: if the observed statistic is too close to either end of the truncation interval ($vlo$ and $vup$, see references), then one or possibly both endpoints of the interval of desired coverage cannot be computed, and default to +/- Inf. The output tailarea gives the achieved Gaussian tail areas for the reported intervals—these should be close to alpha/2, and can be used for error-checking purposes.

**Value**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>Type of analysis (active, all, or aic)</td>
</tr>
<tr>
<td>k</td>
<td>Value of $k$ specified in call</td>
</tr>
<tr>
<td>khat</td>
<td>When type is &quot;active&quot;, this is an estimated stopping point declared by forwardStop; when type is &quot;aic&quot;, this is the value chosen by the modified AIC scheme</td>
</tr>
<tr>
<td>pv</td>
<td>One sided P-values for active variables, uses the sign that a variable entered the model with.</td>
</tr>
<tr>
<td>ci</td>
<td>Confidence intervals</td>
</tr>
<tr>
<td>tailarea</td>
<td>Realized tail areas (lower and upper) for each confidence interval</td>
</tr>
<tr>
<td>vlo</td>
<td>Lower truncation limits for statistics</td>
</tr>
</tbody>
</table>
fsInf

- vup: Upper truncation limits for statistics
- vmat: Linear contrasts that define the observed statistics
- y: Vector of outcomes
- vars: Variables in active set
- sign: Signs of active coefficients
- alpha: Desired coverage (alpha/2 in each tail)
- sigma: Value of error standard deviation (sigma) used
- call: The call to fsInf

Author(s)
Ryan Tibshirani, Rob Tibshirani, Jonathan Taylor, Joshua Loftus, Stephen Reid

References

See Also
fs

Examples
```
set.seed(33)
n = 50
p = 10
sigma = 1
x = matrix(rnorm(n*p), n, p)
beta = c(3, 2, rep(0, p-2))
y = x %*% beta + sigma * rnorm(n)

# run forward stepwise
fsfit = fs(x, y)

# compute sequential p-values and confidence intervals
# (sigma estimated from full model)
out.seq = fsInf(fsfit)
out.seq

# compute p-values and confidence intervals after AIC stopping
out.aic = fsInf(fsfit, type = "aic")
out.aic

# compute p-values and confidence intervals after 5 fixed steps
out.fix = fsInf(fsfit, type = "all", k = 5)
out.fix
```
**groupfs**

Select a model with forward stepwise.

---

**Description**

This function implements forward selection of linear models almost identically to `step` with direction = "forward". The reason this is a separate function from `fs` is that groups of variables (e.g. dummies encoding levels of a categorical variable) must be handled differently in the selective inference framework.

**Usage**

```r
groupfs(x, y, index, maxsteps, sigma = NULL, k = 2, intercept = TRUE,
        center = TRUE, normalize = TRUE, aicstop = 0, verbose = FALSE)
```

**Arguments**

- `x`: Matrix of predictors (n by p).
- `y`: Vector of outcomes (length n).
- `index`: Group membership indicator of length p. Check that `sort(unique(index)) = 1:G` where G is the number of distinct groups.
- `maxsteps`: Maximum number of steps for forward stepwise.
- `sigma`: Estimate of error standard deviation for use in AIC criterion. This determines the relative scale between RSS and the degrees of freedom penalty. Default is NULL corresponding to unknown sigma. When NULL, `link{groupfsInf}` performs truncated F inference instead of truncated $\chi$. See `extractAIC` for details on the AIC criterion.
- `k`: Multiplier of model size penalty, the default is $k = 2$ for AIC. Use $k = \log(n)$ for BIC, or $k = 2\log(p)$ for RIC (best for high dimensions, when $p > n$). If $G < p$ then RIC may be too restrictive and it would be better to use $\log(G) < k < 2\log(p)$.
- `intercept`: Should an intercept be included in the model? Default is TRUE. Does not count as a step.
- `center`: Should the columns of the design matrix be centered? Default is TRUE.
- `normalize`: Should the design matrix be normalized? Default is TRUE.
- `aicstop`: Early stopping if AIC increases. Default is 0 corresponding to no early stopping. Positive integer values specify the number of times the AIC is allowed to increase in a row, e.g. with `aicstop = 2` the algorithm will stop if the AIC criterion increases for 2 steps in a row. The default of `step` corresponds to `aicstop = 1`.
- `verbose`: Print out progress along the way? Default is FALSE.
Value

An object of class "groupfs" containing information about the sequence of models in the forward stepwise algorithm. Call the function `groupfsInf` on this object to compute selective p-values.

See Also

`groupfsInf`, `factorDesign`.

Examples

```r
x = matrix(rnorm(20*40), nrow=20)
index = sort(rep(1:20, 2))
y = rnorm(20) + 2 * x[,1] - x[,4]
fit = groupfs(x, y, index, maxsteps = 5)
out = groupfsInf(fit)
out
```

```r
# groupfsInf

Compu

t selective p-values for a model fitted by groupfs.

Description

Computes p-values for each group of variables in a model fitted by `groupfs`. These p-values adjust for selection by truncating the usual \( \chi^2 \) statistics to the regions implied by the model selection event. If the \( \sigma \) to `groupfs` was NULL then `groupfsInf` uses truncated F statistics instead of truncated \( \chi \). The \( \sigma \) argument to `groupfsInf` allows users to override and use \( \chi \), but this is not recommended unless \( \sigma \) can be estimated well (i.e. \( n > p \)).

Usage

`groupfsInf(obj, sigma = NULL, verbose = TRUE)`

Arguments

- `obj` Object returned by `groupfs` function
- `sigma` Estimate of error standard deviation. Default is NULL and in this case `groupfsInf` uses the value of `sigma` specified to `groupfs`.
- `verbose` Print out progress along the way? Default is TRUE.

Value

An object of class "groupfsInf" containing selective p-values for the fitted model `obj`. For comparison with `fsInf`, note that the option `type = "active"` is not available.

- `vars` Labels of the active groups in the order they were included.
- `pv` Selective p-values computed from appropriate truncated distributions.
- `sigma` Estimate of error variance used in computing p-values.
TC or TF  Observed value of truncated $\chi$ or $F$.

df  Rank of group of variables when it was added to the model.

support  List of intervals defining the truncation region of the corresponding statistic.

---

**lar**

*Least angle regression*

**Description**

This function implements least angle regression, for use in the selectiveInference package

**Usage**

```r
lar(x, y, maxsteps=2000, minlam=0, intercept=TRUE, normalize=TRUE, verbose=FALSE)
```

**Arguments**

- **x**: Matrix of predictors (n by p)
- **y**: Vector of outcomes (length n)
- **maxsteps**: Maximum number of steps to take
- **minlam**: Minimum value of lambda to consider
- **intercept**: Should an intercept be included on the model? Default is TRUE
- **normalize**: Should the predictors be normalized? Default is TRUE
- **verbose**: Print out progress along the way? Default is FALSE

**Details**

The least angle regression algorithm is described in detail by Efron et al. (2002). This function should match (in terms of its output) that from the `lars` package, but returns additional information (namely, the polyhedral constraints) needed for the selective inference calculations.

**Value**

- **lambda**: Values of lambda (knots) visited along the path
- **action**: Vector of predictors in order of entry
- **sign**: Signs of coefficients of predictors, upon entry
- **df**: Degrees of freedom of each active model
- **beta**: Matrix of regression coefficients for each model along the path, one model per column
- **completepath**: Was the complete stepwise path computed?
- **bls**: If completepath is TRUE, the full least squares coefficients
- **Gamma**: Matrix that captures the polyhedral selection at each step
nk  Number of polyhedral constraints at each step in path
vreg  Matrix of linear contrasts that gives coefficients of variables to enter along the path
mp  Value of M+ (for internal use with the spacing test)
x  Matrix of predictors used
y  Vector of outcomes used
bx  Vector of column means of original x
by  Mean of original y
sx  Norm of each column of original x
intercept  Was an intercept included?
normalize  Were the predictors normalized?
call  The call to lar

Author(s)
Ryan Tibshirani, Rob Tibshirani, Jonathan Taylor, Max G’Sell, Joshua Loftus, Stephen Reid

References
See also the descriptions in Trevor Hastie, Rob Tibshirani, and Jerome Friedman (2002, 2009). Elements of Statistical Learning.

See Also
larInf, predict.lar, coef.lar, plot.lar

Examples
set.seed(43)
n = 50
p = 10
sigma = 1
x = matrix(rnorm(n*p),n,p)
beta = c(3,2,rep(0,p-2))
y = x%*%beta + sigma*rnorm(n)

# run LAR, plot results
larfit = lar(x,y)
plot(larfit)

# compute sequential p-values and confidence intervals
# (sigma estimated from full model)
out = larInf(larfit)
out
larInf

Selective inference for least angle regression

Description

Computes p-values and confidence intervals for least angle regression

Usage

larInf(obj, sigma=NULL, alpha=0.1, k=NULL, type=c("active","all","aic"),
gridrange=c(-100,100), bits=NULL, mult=2, ntimes=2, verbose=FALSE)

Arguments

obj Object returned by lar function (not the lars function!)
sigma Estimate of error standard deviation. If NULL (default), this is estimated using
the mean squared residual of the full least squares fit when n >= 2p, and using
the standard deviation of y when n < 2p. In the latter case, the user should use
estimateSigma function for a more accurate estimate
alpha Significance level for confidence intervals (target is miscoverage alpha/2 in each
tail)
k See "type" argument below. Default is NULL, in which case k is taken to be the
the number of steps computed in the least angle regression path
type Type of analysis desired: with "active" (default), p-values and confidence inter-
vals are computed for each predictor as it is entered into the active step, all the
way through k steps; with "all", p-values and confidence intervals are computed
for all variables in the active model after k steps; with "aic", the number of steps
k is first estimated using a modified AIC criterion, and then the same type of
analysis as in "all" is carried out for this particular value of k.

Note that the AIC scheme is defined to choose a number of steps k after which
the AIC criterion increases ntimes in a row, where ntimes can be specified
by the user (see below). Under this definition, the AIC selection event is char-
acterizable as a polyhedral set, and hence the extra conditioning can be taken
into account exactly. Also note that an analogous BIC scheme can be specified
through the mult argument (see below)

gridrange Grid range for constructing confidence intervals, on the standardized scale
bits Number of bits to be used for p-value and confidence interval calculations. De-
fault is NULL, in which case standard floating point calculations are performed. When
not NULL, multiple precision floating point calculations are performed with
the specified number of bits, using the R package Rmpfr (if this package
is not installed, then a warning is thrown, and standard floating point calcu-
lations are pursued). Note: standard double precision uses 53 bits so, e.g., a
choice of 200 bits uses about 4 times double precision. The confidence interval
computation is sometimes numerically challenging, and the extra precision can
be helpful (though computationally more costly). In particular, extra precision
might be tried if the values in the output columns of `tailarea` differ noticeably from `alpha/2`.

- `mult`: Multiplier for the AIC-style penalty. Hence a value of 2 (default) gives AIC, whereas a value of \log(n) would give BIC.
- `ntimes`: Number of steps for which AIC-style criterion has to increase before minimizing point is declared.
- `verbose`: Print out progress along the way? Default is `FALSE`.

### Details

This function computes selective p-values and confidence intervals (selection intervals) for least angle regression. The default is to report the results for each predictor after its entry into the model. See the "type" argument for other options. The confidence interval construction involves numerical search and can be fragile: if the observed statistic is too close to either end of the truncation interval (`vlo` and `vup`, see references), then one or possibly both endpoints of the interval of desired coverage cannot be computed, and default to \( \pm \infty \). The output `tailarea` gives the achieved Gaussian tail areas for the reported intervals—these should be close to `alpha/2`, and can be used for error-checking purposes.

### Value

- `type`: Type of analysis (active, all, or aic)
- `k`: Value of k specified in call
- `khat`: When `type` is "active", this is an estimated stopping point declared by `forwardStop`; when `type` is "aic", this is the value chosen by the modified AIC scheme.
- `pv`: P-values for active variables
- `ci`: Confidence intervals
- `tailarea`: Realized tail areas (lower and upper) for each confidence interval
- `vlo`: Lower truncation limits for statistics
- `vup`: Upper truncation limits for statistics
- `vmat`: Linear contrasts that define the observed statistics
- `y`: Vector of outcomes
- `pv.spacing`: P-values from the spacing test (here M+ is used)
- `pv.modspac`: P-values from the modified form of the spacing test (here M+ is replaced by the next knot)
- `pv.covtest`: P-values from covariance test
- `vars`: Variables in active set
- `sign`: Signs of active coefficients
- `alpha`: Desired coverage (\( \alpha/2 \) in each tail)
- `sigma`: Value of error standard deviation (\( \sigma \)) used
- `call`: The call to `larInf`
Author(s)
Ryan Tibshirani, Rob Tibshirani, Jonathan Taylor, Joshua Loftus, Stephen Reid

References

See Also
lar

Examples
set.seed(43)
 n = 50
 p = 10
 sigma = 1
 x = matrix(rnorm(n*p),n,p)
 beta = c(3,2,rep(0,p-2))
 y = x%*%beta + sigma*rnorm(n)

 # run LAR
 larfit = lar(x,y)

 # compute sequential p-values and confidence intervals
 # (sigma estimated from full model)
 out.seq = larInf(larfit)
 out.seq

 # compute p-values and confidence intervals after AIC stopping
 out.aic = larInf(larfit,type="aic")
 out.aic

 # compute p-values and confidence intervals after 5 fixed steps
 out.fix = larInf(larfit,type="all",k=5)
 out.fix

manyMeans
Selective inference for many normal means

Description
Computes p-values and confidence intervals for the largest k among many normal means

Usage
manyMeans(y, alpha=0.1, bh.q=NULL, k=NULL, sigma=1, verbose=FALSE)
Arguments

- **y**: Vector of outcomes (length n)
- **alpha**: Significance level for confidence intervals (target is miscoverage alpha/2 in each tail)
- **bh.q**: q parameter for BH(q) procedure
- **k**: Number of means to consider
- **sigma**: Estimate of error standard deviation
- **verbose**: Print out progress along the way? Default is FALSE

Details

This function compute p-values and confidence intervals for the largest k among many normal means. One can specify a fixed number of means k to consider, or choose the number to consider via the BH rule.

Value

- **mu.hat**: Vector of length n containing the estimated signal sizes. If a sample element is not selected, then its signal size estimate is 0
- **selected.set**: Indices of the vector y of the sample elements that were selected by the procedure (either BH(q) or top-K). Labelled "Selind" in output table.
- **pv**: P-values for selected signals
- **ci**: Confidence intervals
- **method**: Method used to choose number of means
- **sigma**: Value of error standard deviation (sigma) used
- **bh.q**: BH q-value used
- **k**: Desired number of means
- **threshold**: Computed cutoff
- **call**: The call to manyMeans

Author(s)

Ryan Tibshirani, Rob Tibshirani, Jonathan Taylor, Joshua Loftus, Stephen Reid

References


Examples

```r
set.seed(12345)
n = 100
mu = c(rep(3, floor(n/5)), rep(0, n-floor(n/5)))
y = mu + rnorm(n)
out = manyMeans(y, bh.q=0.1)
out
```
plot.fs

Plot function for forward stepwise regression

Description

Plot coefficient profiles along the forward stepwise path

Usage

## S3 method for class 'fs'
plot(x, breaks=TRUE, omit.zeros=TRUE, var.labels=TRUE, ...)

Arguments

x
Object returned by a call to fs function

breaks
Should vertical lines be drawn at each break point in the piecewise linear coefficient paths? Default is TRUE

omit.zeros
Should segments of the coefficients paths that are equal to zero be omitted (to avoid clutter in the figure)? Default is TRUE

var.labels
Should paths be labelled with corresponding variable numbers? Default is TRUE

...
Additional arguments for plotting

Author(s)

Ryan Tibshirani, Rob Tibshirani, Jonathan Taylor, Joshua Loftus, Stephen Reid

Examples

set.seed(33)
n = 50
p = 10
sigma = 1
x = matrix(rnorm(n=p),n,p)
beta = c(3,2,rep(0,p-2))
y = x%*%beta + sigma*rnorm(n)

# run forward stepwise, plot results
fsfit = fs(x,y)
plot(fsfit)
plot.lar | Plot function for least angle regression

Description

Plot coefficient profiles along the LAR path

Usage

## S3 method for class 'lar'
plot(x, xvar=c("norm", "step", "lambda"), breaks=TRUE,
     omit.zeros=TRUE, var.labels=TRUE, ...)

Arguments

x Object returned by a call to lar function (not the lars function!)
xvar Either "norm" or "step" or "lambda", determining what is plotted on the x-axis
breaks Should vertical lines be drawn at each break point in the piecewise linear coefficient paths? Default is TRUE
omit.zeros Should segments of the coefficients paths that are equal to zero be omitted (to avoid clutter in the figure)? Default is TRUE
var.labels Should paths be labelled with corresponding variable numbers? Default is TRUE
... Additional arguments for plotting

Author(s)

Ryan Tibshirani, Rob Tibshirani, Jonathan Taylor, Joshua Loftus, Stephen Reid

Examples

set.seed(43)
n = 50
p = 10
sigma = 1
x = matrix(rnorm(n*p),n,p)
beta = c(3,2,rep(0,p-2))
y = x%*%beta + sigma*rnorm(n)

# run LAR, plot results
larfit = lar(x,y)
plot(larfit)
**predict.fs**  
*Prediction and coefficient functions for forward stepwise regression*

**Description**

Make predictions or extract coefficients from a forward stepwise object

**Usage**

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

**Arguments**

- `object`: Object returned by a call to `fs` function
- `newx`: Matrix of x values at which the predictions are desired. If NULL, the x values from forward stepwise fitting are used
- `s`: Step number(s) at which predictions or coefficients are desired
- `...`: Additional arguments

**Value**

Either a vector/matrix of predictions, or a vector/matrix of coefficients.

**Author(s)**

Ryan Tibshirani, Rob Tibshirani, Jonathan Taylor, Joshua Loftus, Stephen Reid

**Examples**

```r
set.seed(33)
n = 200
p = 20
sigma = 1
x = matrix(rnorm(n*p),n,p)
beta = c(rep(3,10),rep(0,p-10))
y = x%*%beta + sigma*rnorm(n)

# run forward stepwise and predict functions
obj = fs(x,y)
fit = predict(obj,x,s=3)
```
predict.groupfs  

**Prediction and coefficient functions for groupfs.**

Description

Make predictions or extract coefficients from a groupfs forward stepwise object.

Usage

```r
## S3 method for class 'groupfs'
predict(object, newx)
```

Arguments

- **object**: Object returned by a call to `groupfs`.
- **newx**: Matrix of x values at which the predictions are desired. If NULL, the x values from groupfs fitting are used.

Value

A vector of predictions or a vector of coefficients.

predict.lar  

**Prediction and coefficient functions for least angle regression**

Description

Make predictions or extract coefficients from a least angle regression object.

Usage

```r
## S3 method for class 'lar'
predict(object, newx, s, mode=c("step","lambda"), ...)
## S3 method for class 'lar'
coef(object, s, mode=c("step","lambda"), ...)
```

Arguments

- **object**: Object returned by a call to `lar` function (not the `lars` function!)
- **newx**: Matrix of x values at which the predictions are desired. If NULL, the x values from least angle regression fitting are used.
- **s**: Step number(s) or lambda value(s) at which predictions or coefficients are desired.
- **mode**: Either "step" or "lambda", determining the role of s (above)
- **...**: Additional arguments
randomizedLasso

Inference for the randomized lasso, with a fixed lambda

Description
Solve a randomly perturbed LASSO problem.

Usage
randomizedLasso(X,
y,
lam,
family=c("gaussian", "binomial"),
noise_scale=NULL,
ridge_term=NULL,
max_iter=100,
kkt_tol=1.e-4,
parameter_tol=1.e-8,
objective_tol=1.e-8,
objective_stop=FALSE,
kkt_stop=TRUE,
parameter_stop=TRUE)
Arguments

- **x**: Matrix of predictors (n by p);
- **y**: Vector of outcomes (length n);
- **lam**: Value of lambda used to compute beta. See the above warning Be careful! This function uses the "standard" lasso objective

\[ \frac{1}{2}||y - x\beta||^2_2 + \lambda||\beta||_1. \]

In contrast, glmnet multiplies the first term by a factor of 1/n. So after running glmnet, to extract the beta corresponding to a value lambda, you need to use \( \beta = \text{coef(obj, s=\lambda/n)}[-1] \), where obj is the object returned by glmnet (and [-1] removes the intercept, which glmnet always puts in the first component).

- **family**: Response type: "gaussian" (default), "binomial".
- **noise_scale**: Scale of Gaussian noise added to objective. Default is \( 0.5 \times \text{sd}(y) \times \sqrt{\text{trace}(X^TX)} \).
- **ridge_term**: A small "elastic net" or ridge penalty is added to ensure the randomized problem has a solution. \( 0.5 \times \text{sd}(y) \times \sqrt{\text{trace}(X^TX)} / \sqrt{n} \).
- **max_iter**: How many rounds of updates used of coordinate descent in solving randomized LASSO.
- **kkt_tol**: Tolerance for checking convergence based on KKT conditions.
- **parameter_tol**: Tolerance for checking convergence based on convergence of parameters.
- **objective_tol**: Tolerance for checking convergence based on convergence of objective value.
- **kkt_stop**: Should we use KKT check to determine when to stop?
- **parameter_stop**: Should we use convergence of parameters to determine when to stop?
- **objective_stop**: Should we use convergence of objective value to determine when to stop?

Details

For family="gaussian" this function uses the "standard" lasso objective

\[ \frac{1}{2}||y - x\beta||^2_2 + \lambda||\beta||_1 \]

and adds a term

\[ -\omega^T\beta + \frac{\epsilon}{2}||\beta||^2_2 \]

where omega is drawn from IID normals with standard deviation noise_scale and epsilon given by ridge_term. See below for default values of noise_scale and ridge_term.

For family="binomial", the squared error loss is replaced by the negative of the logistic log-likelihood.
**Value**

- **X**
  Design matrix.

- **y**
  Response vector.

- **lam**
  Vector of penalty parameters.

- **family**
  Family: "gaussian" or "binomial".

- **active_set**
  Set of non-zero coefficients in randomized solution that were penalized. Integers from 1:p.

- **inactive_set**
  Set of zero coefficients in randomized solution. Integers from 1:p.

- **unpenalized_set**
  Set of non-zero coefficients in randomized solution that were not penalized. Integers from 1:p.

- **sign_soln**
  The sign pattern of the randomized solution.

- **full_law**
  List describing sampling parameters for conditional law of all optimization variables given the data in the LASSO problem.

- **conditional_law**
  List describing sampling parameters for conditional law of only the scaling variables given the data and the observed subgradient in the LASSO problem.

- **internal_transform**
  Affine transformation describing relationship between internal representation of the data and the data component of score of the likelihood at the unregularized MLE based on the sign_vector (a.k.a. relaxed LASSO).

- **observed_raw**
  Data component of the score at the unregularized MLE.

- **noise_scale**
  SD of Gaussian noise used to draw the perturbed objective.

- **soln**
  The randomized solution. Inference is made conditional on its sign vector (so no more snooping of this value is formally permitted.) If condition_subgrad == TRUE when sampling, then we may snoop on the observed subgradient.

- **perturb**
  The random vector in the linear term added to the objective.

**Author(s)**

Jelena Markovic, Jonathan Taylor

**References**


**Examples**

```r
set.seed(43)
n = 50
p = 10
sigma = 0.2
```
lam = 0.5

X = matrix(rnorm(n*p), n, p)
X = scale(X, TRUE, TRUE) / sqrt(n-1)

beta = c(3,2,rep(0,p-2))
y = X%*%beta + sigma*rnorm(n)

result = randomizedLasso(X, y, lam)

randomizedLassoInf  Inference for the randomized lasso, with a fixed lambda

Description

Compute p-values and confidence intervals based on selecting an active set with the randomized lasso, at a fixed value of the tuning parameter lambda and using Gaussian randomization.

Usage

randomizedLassoInf(rand_lasso_soln,
  targets=NULL,
  level=0.9,
  sampler=c("norejection", "adaptMCMC"),
  nsample=10000,
  burnin=2000,
  opt_samples=NULL)

Arguments

rand_lasso_soln
A randomized lasso solution as returned by randomizedLasso.

targets
If not NULL, should be a list with entries observed_target, cov_target, crosscov_target_internal.
The observed_target should be (pre-selection) asymptotically Gaussian around targeted parameters. The quantity cov_target should be an estimate of the (pre-selection) covariance of observed_target. Finally, crosscov_target_internal should be an estimate of the (pre-selection) covariance of observed_target and the internal representation of the data of the LASSO. For both "gaussian" and "binomial", this is the vector

\[ \hat{\beta}_{E,MLE} X_E^T (y - \mu(E \hat{\beta}_{E,MLE})) \]

For example, this cross-covariance could be estimated by jointly bootstrapping the target of interest and the above vector.

level
Level for confidence intervals.

target
Which sampler to use – default is a no-rejection sampler. Otherwise use MCMC from the adaptMCMC package.
nsample: Number of samples of optimization variables to sample.
burnin: How many samples of optimization variable to discard (should be less than nsample).

Details

This function computes selective p-values and confidence intervals for a randomized version of the lasso, given a fixed value of the tuning parameter lambda.

Value

targets: A list with entries observed_target, cov_target, crosscov_target_internal. See argument description above.
pvalues: P-values testing hypotheses that each specific target is 0.

Author(s)

Jelena Markovic, Jonathan Taylor

References


Examples

set.seed(43)
n = 50
p = 10
sigma = 0.2
lam = 0.5

X = matrix(rnorm(n*p), n, p)
X = scale(X, TRUE, TRUE) / sqrt(n-1)

beta = c(3,2,rep(0,p-2))
y = X%*%beta + sigma*runif(n)

result = randomizedLasso(X, y, lam)
inf_result = randomizedLassoInf(result)
**Description**

Compute p-values and confidence intervals for the lasso estimate, at a fixed value of the tuning parameter lambda using the "relevant" conditioning event of arxiv.org/1801.09037.

**Usage**

```r
ROSI(X,
y, soln, lambda,
penalty_factor=NULL, dispersion=1, family=c('gaussian', 'binomial'), solver=c('QP', 'glmnet'), construct_ci=TRUE, debiasing_method=c("JM", "BN"), verbose=FALSE, level=0.9, use_debiased=TRUE)
```

**Arguments**

- **x** Matrix of predictors (n by p);
- **y** Vector of outcomes (length n)
- **soln** Estimated lasso coefficients (e.g., from glmnet). This is of length p (so the intercept is not included as the first component).
  
  Be careful! This function uses the "standard" lasso objective
  
  $$\frac{1}{2} \| y - X \beta \|^2_2 + \lambda \| \beta \|_1.$$
  
  In contrast, glmnet multiplies the first term by a factor of 1/n. So after running glmnet, to extract the beta corresponding to a value lambda, you need to use
  
  ```r
  beta = coef(obj, s=lambda/n)[-1],
  ```
  
  where obj is the object returned by glmnet (and [-1] removes the intercept, which glmnet always puts in the first component)

- **lambda** Value of lambda used to compute beta. See the above warning

- **penalty_factor** Penalty factor as used by glmnet. Actual penalty used in solving the problem is
  
  $$\lambda \cdot \sum_{i=1}^{p} f_i | \beta_i |$$

  with f being the penalty_factor. Defaults to vector of 1s.
dispersion  Estimate of dispersion in the GLM. Can be taken to be 1 for logistic and should be an estimate of the error variance for the Gaussian.

family  Family used for likelihood.

solver  Solver used to solve restricted problems needed to find truncation set. Each active variable requires solving a new LASSO problem obtained by zeroing out one coordinate of original problem. The "QP" choice uses coordinate descent for a specific value of lambda, rather than glmnet which would solve for a new path each time.

construct_ci  Report confidence intervals or just p-values?

debiasing_method  Which method should be used for debiasing? Choices are "JM" (Javanmard, Montanari) or "BN" (method described in arxiv.org/1703.03282).

verbose  Print out progress along the way? Default is FALSE.

level  Confidence level for intervals.

use_debiased  Use the debiased estimate of the parameter or not. When FALSE, this is the method described in arxiv.org/1801.09037. The default TRUE often produces noticeably shorter intervals and more powerful tests when p is comparable to n. Ignored when n<p and set to TRUE. Also note that with "BN" as debiasing method and n > p, this agrees with method in arxiv.org/1801.09037.

Details

???

Value

active_set  Active set of LASSO.

pvalues  Two-sided P-values for active variables.

intervals  Confidence intervals

estimate  Relaxed (i.e. unshrunk) selected estimates.

std_err  Standard error of relaxed estimates (pre-selection).

dispersion  Dispersion parameter.

lower_trunc  Lower truncation point. The estimates should be outside the interval formed by the lower and upper truncation points.

upper_trunc  Lower truncation point. The estimates should be outside the interval formed by the lower and upper truncation points.

lambda  Value of tuning parameter lambda used.

penalty_factor  Penalty factor used for solving problem.

level  Confidence level.

call  The call to fixedLassoInf.

Author(s)

Jelena Markovic, Jonathan Taylor
References


Tom Boot, Didier Nibbering. Inference in high-dimensional linear regression models. arXiv:1703.03282

Examples

```r
library(selectiveInference)
library(glmnet)
set.seed(43)

n = 100
p = 200
s = 2
sigma = 1

x = matrix(rnorm(n*p),n,p)
x = scale(x,TRUE,TRUE)

beta = c(rep(10, s), rep(0,p-s)) / sqrt(n)
y = x %*% beta + sigma*rnorm(n)

# first run glmnet
gfit = glmnet(x,y,standardize=FALSE)

# extract coef for a given lambda; note the 1/n factor!
# (and we don't save the intercept term)
lambda = 4 * sqrt(n)
lambda_glmnet = 4 / sqrt(n)
beta = selectiveInference:::solve_problem_glmnet(x,
y,
lambda_glmnet,
penalty_factor=rep(1, p),
family="gaussian")

# compute fixed lambda p-values and selection intervals
out = ROSI(x,
y,
beta,
lambda,
dispersion=sigma^2)

out

# an alternate approximate inverse from Boot and Nibbering

out = ROSI(x,
y,
beta,
lambda,
dispersion=sigma^2,
debiasing_method="BN")
```
scaleGroups  Center and scale design matrix by groups

Description
For internal use by groupfs.

Usage
scaleGroups(x, index, center = TRUE, normalize = TRUE)

Arguments
x  Design matrix.
index  Group membership indicator of length p.
center  Center groups, default is TRUE.
normalize  Scale groups by Frobenius norm, default is TRUE.

Value
x  Optionally centered/scaled design matrix.
    xm  Means of groups in original design matrix.
    xs  Frobenius norms of groups in original design matrix.

selectiveInference  Tools for selective inference

Description
Functions to perform post-selection inference for forward stepwise regression, least angle regression, the lasso and the many normal means problem. The lasso function also supports logistic regression and the Cox model.

Details
Package: selectiveInference
Type: Package
License: GPL-2
This package provides tools for inference after selection, in forward stepwise regression, least angle regression, the lasso, and the many normal means problem. The functions compute p-values and selection intervals that properly account for the inherent selection carried out by the procedure. These have exact finite sample type I error and coverage under Gaussian errors. For the logistic and Cox families (fixedLassoInf), the coverage is asymptotically valid.

This R package was developed as part of the selective inference software project in Python and R: https://github.com/selective-inference

Some of the R code in this work is a modification of Python code from this repository. Here is the current selective inference software team:

Yuval Benjamini, Leonard Blier, Will Fithian, Jason Lee, Joshua Loftus, Joshua Loftus, Stephen Reid, Dennis Sun, Yuekai Sun, Jonathan Taylor, Xiaoying Tian, Ryan Tibshirani, Rob Tibshirani

The main functions included in the package are: fs, fsInf, lar, larInf, fixedLassoInf, manyMeans

Author(s)

Ryan Tibshirani, Rob Tibshirani, Jonathan Taylor, Joshua Loftus, Stephen Reid
Maintainer: Rob Tibshirani <tibs@stanford.edu>

References


Examples

```r
set.seed(33)
n = 50
p = 10
sigma = 1
x = matrix(rnorm(n*p),n,p)
beta = c(3,2,rep(0,p-2))
y = x%*%beta + sigma*rnorm(n)

# run forward stepwise
fsfit = fs(x,y)

# compute sequential p-values and confidence intervals
# (sigma estimated from full model)
out.seq = fsInf(fsfit)
out.seq

# compute p-values and confidence intervals after AIC stopping
```
out.aic = fsInf(fsfit,type="aic")
out.aic

# compute p-values and confidence intervals after 5 fixed steps
out.fix = fsInf(fsfit,type="all",k=5)
out.fix

## NOT RUN---lasso at fixed lambda- Gaussian family
## first run glmnet
# gfit = glmnet(x,y)

## extract coef for a given lambda; note the 1/n factor!
## (and we don’t save the intercept term)
# lambda = .1
# beta = coef(gfit, s=lambda/n, exact=TRUE)[[-1]]

## compute fixed lambda p-values and selection intervals
# out = fixedLassoInf(x,y,beta,lambda,sigma=sigma)
# out

# lasso at fixed lambda- logistic family
# set.seed(43)
  # n = 50
  # p = 10
  # sigma = 1

  # x = matrix(rnorm(n*p),n,p)
x=scale(x,TRUE,TRUE)
  #
  # beta = c(3,2,rep(0,p-2))
  # y = x%*%beta + sigma*rvnorm(n)
  # y=1*(y>mean(y))
  # first run glmnet
  # gfit = glmnet(x,y,standardize=FALSE,family="binomial")

  # extract coef for a given lambda; note the 1/n factor!
  # (and here we DO include the intercept term)
  # lambda = .8
  # beta = coef(gfit, s=lambda/n, exact=TRUE)

  # compute fixed lambda p-values and selection intervals
  # out = fixedLassoInf(x,y,beta,lambda,family="binomial")
  # out

# lasso at fixed lambda- Cox family
# set.seed(43)
  # n = 50
  # p = 10
  # sigma = 1

  # x = matrix(rnorm(n*p),n,p)
  # x=scale(x,TRUE,TRUE)
```r
# beta = c(3,2,rep(0,p-2))
# tim = as.vector(x%*%beta + sigma*rnorm(n))
# tim = tim-min(tim)+1
# status=sample(c(0,1),size=n,replace=T)
# first run glmnet
# gfit = glmnet(x,Surv(tim,status),standardize=FALSE,family="cox")
# extract coef for a given lambda; note the 1/n factor!
# lambda = 1.5
# beta = as.numeric(coef(gfit, s=lambda/n, exact=TRUE))

# compute fixed lambda p-values and selection intervals
# out = fixedLassoInf(x,tim,beta,lambda,status=status,family="cox")
# out

## NOT RUN---many normal means
# set.seed(12345)
# n = 100
# mu = c(rep(3,floor(n/5)), rep(0,n-floor(n/5)))
# y = mu + rnorm(n)
# out = manyMeans(y, bh.q=0.1)

## NOT RUN---forward stepwise with groups
# set.seed(1)
# n = 20
# p = 40
# x = matrix(rnorm(n*p), nrow=n)
# index = sort(rep(1:(p/2), 2))
# y = rnorm(n) + 2 * x[,1] - x[,4]
# fit = groupfs(x, y, index, maxsteps = 5)
# out = groupfsInf(fit)

## NOT RUN---estimation of sigma for use in fsInf
## (or larInf or fixedLassoInf)
# set.seed(33)
# n = 50
# p = 10
# sigma = 1
# x = matrix(rnorm(n*p),n,p)
# beta = c(3,2,rep(0,p-2))
# y = x%*%beta + sigma*rnorm(n)

## run forward stepwise
# fsfit = fs(x,y)

## estimate sigma
# sigmahat = estimateSigma(x,y)$sigmahat

## run sequential inference with estimated sigma
# out = fsInf(fit,sigma=sigmahat)
# out
```
TG.interval

**Truncated Gaussian confidence interval.**

**Description**

Compute truncated Gaussian interval of Lee et al. (2016) with arbitrary affine selection and covariance. $Z$ should satisfy $A$

**Usage**

```r
TG.interval(Z, A, b, eta, Sigma=NULL, alpha=0.1,
gridrange=c(-100,100),
gridpts=100,
griddepth=2,
flip=FALSE,
bits=NULL)
```

**Arguments**

- **Z**
  - Observed data (assumed to follow $N(\mu, \Sigma)$ with $\sum(\eta*\mu)=$null_value)
- **A**
  - Matrix specifying affine inequalities $AZ \leq b$
- **b**
  - Offsets in the affine inequalities $AZ \leq b$
- **eta**
  - Determines the target $\sum(\eta*\mu)$ and estimate $\sum(\eta*Z)$.
- **Sigma**
  - Covariance matrix of $Z$. Defaults to identity.
- **alpha**
  - Significance level for confidence intervals (target is miscoverage $\alpha/2$ in each tail)
- **gridrange**
  - Grid range for constructing confidence intervals, on the standardized scale.
- **gridpts**
  - Grid points
- **griddepth**
  - Grid depth
- **flip**
  - ????
- **bits**
  - Number of bits to be used for p-value and confidence interval calculations. Default is NULL, in which case standard floating point calculations are performed. When not NULL, multiple precision floating point calculations are performed with the specified number of bits, using the R package Rmpfr (if this package is not installed, then a warning is thrown, and standard floating point calculations are pursued). Note: standard double precision uses 53 bits so, e.g., a choice of 200 bits uses about 4 times double precision. The confidence interval computation is sometimes numerically challenging, and the extra precision can be helpful (though computationally more costly). In particular, extra precision might be tried if the values in the output columns of tailarea differ noticeably from $\alpha/2$. 


**TG.limits**

**Details**

This function computes selective confidence intervals based on the polyhedral lemma of Lee et al. (2016).

**Value**

- **int** Selective confidence interval.
- **tailarea** Realized tail areas (lower and upper) for each confidence interval.

**Author(s)**

Ryan Tibshirani, Rob Tibshirani, Jonathan Taylor, Joshua Loftus, Stephen Reid

**References**


**Examples**

```r
A = diag(5)
b = rep(1, 5)
Z = rep(0, 5)
Sigma = diag(5)
eta = as.numeric(c(1, 1, 0, 0, 0))
TG.interval(Z, A, b, eta, Sigma)
```

---

**TG.limits**

*Truncation limits and standard deviation.*

**Description**

Compute truncated limits and SD for use in computing p-values or confidence intervals of Lee et al. (2016). Z should satisfy A

**Usage**

```r
TG.limits(Z, A, b, eta, Sigma)
```

**Arguments**

- **Z** Observed data (assumed to follow N(mu, Sigma) with sum(eta*mu)=null_value)
- **A** Matrix specifying affine inequalities AZ <= b
- **b** Offsets in the affine inequalities AZ <= b.
- **eta** Determines the target sum(eta*mu) and estimate sum(eta*Z).
- **Sigma** Covariance matrix of Z. Defaults to identity.
Details

This function computes the limits of truncation and the implied standard deviation in the polyhedral lemma of Lee et al. (2016).

Value

- vlo: Lower truncation limits for statistic
- vup: Upper truncation limits for statistic
- sd: Standard error of sum(eta*Z)

Author(s)

Ryan Tibshirani, Rob Tibshirani, Jonathan Taylor, Joshua Loftus, Stephen Reid

References


Examples

```r
A = diag(5)
b = rep(1, 5)Z = rep(0, 5)Sigma = diag(5)eta = as.numeric(c(1, 1, 0, 0, 0))TG.limits(Z, A, b, eta, Sigma)
```

Description

Compute truncated Gaussian p-value of Lee et al. (2016) with arbitrary affine selection and covariance. Z should satisfy A

Usage

```r
TG.pvalue(Z, A, b, eta, Sigma, null_value=0, bits=NULL)
```
Arguments

- **Z**: Observed data (assumed to follow $N(\mu, \Sigma)$ with $\sum(\eta*\mu)=\text{null_value}$)
- **A**: Matrix specifying affine inequalities $AZ \leq b$
- **b**: Offsets in the affine inequalities $AZ \leq b$
- **eta**: Determines the target $\sum(\eta*\mu)$ and estimate $\sum(\eta*Z)$
- **Sigma**: Covariance matrix of $Z$. Defaults to identity.
- **null_value**: Hypothesized value of $\sum(\eta*\mu)$ under the null.
- **bits**: Number of bits to be used for p-value and confidence interval calculations. Default is NULL, in which case standard floating point calculations are performed. When not NULL, multiple precision floating point calculations are performed with the specified number of bits, using the R package `Rmpfr` (if this package is not installed, then a warning is thrown, and standard floating point calculations are pursued). Note: standard double precision uses 53 bits so, e.g., a choice of 200 bits uses about 4 times double precision. The confidence interval computation is sometimes numerically challenging, and the extra precision can be helpful (though computationally more costly). In particular, extra precision might be tried if the values in the output columns of `tailarea` differ noticeably from alpha/2.

Details

This function computes selective p-values based on the polyhedral lemma of Lee et al. (2016).

Value

- **pv**: One-sided P-values for active variables, uses the fact we have conditioned on the sign.
- **vlo**: Lower truncation limits for statistic
- **vup**: Upper truncation limits for statistic
- **sd**: Standard error of $\sum(\eta*Z)$

Author(s)

Ryan Tibshirani, Rob Tibshirani, Jonathan Taylor, Joshua Loftus, Stephen Reid

References


Examples

\[
\begin{align*}
A &= \text{diag}(5) \\
b &= \text{rep}(1, 5) \\
Z &= \text{rep}(0, 5) \\
\Sigma &= \text{diag}(5) \\
\eta &= \text{as.numeric}(c(1, 1, 0, 0, 0)) \\
\text{TG.pvalue}(Z, A, b, \eta, \Sigma) \\
\text{TG.pvalue}(Z, A, b, \eta, \Sigma, \text{null.value}=1)
\end{align*}
\]
Index

*Topic package
  selectiveInference, 38

coef.fs, 14
coef.fs(predict.fs), 28
coef.lar, 21
coef.lar(predict.lar), 29
debiasingMatrix, 2
estimateSigma, 4, 7, 15, 22
extractAIC, 18
factorDesign, 5, 19
fixedLassoInf, 6, 39
forwardStop, 12, 16, 23
fs, 13, 15, 17, 18, 39
fsInf, 14, 15, 19, 39
groupfs, 5, 18, 19, 29, 38
groupfsInf, 19, 19
lar, 20, 24, 39
larInf, 21, 22, 39
manyMeans, 24, 39
plot.fs, 14, 26
plot.lar, 21, 27
predict.fs, 14, 28
predict.groupfs, 29
predict.lar, 21, 29
randomizedLasso, 30
randomizedLassoInf, 33
ROSI, 35
scaleGroups, 38
selectiveInference, 38
step, 18
TG.interval, 42
TG.limits, 43
TG.pvalue, 44