Package 'uniLasso'

May 8, 2026

Type Package

Title Univariate-Guided Sparse Regression

Version 2.11

Date 2026-01-13

Depends glmnet, stats, R (>= 3.6.0)

Imports methods, utils, MASS

Suggests testthat

Description Fit a univariate-guided sparse regression (lasso), by a two-stage procedure. The first stage fits p separate univariate models to the response. The second stage gives more weight to the more important univariate features, and preserves their signs. Conveniently, it returns an objects that inherits from class 'glmnet', so that all of the methods for 'glmnet' are available. See Chatterjee, Hastie and Tibshirani (2025) <doi:10.1162/99608f92.c79ff6db> for details.

Encoding UTF-8

License GPL-2

NeedsCompilation no

RoxygenNote 7.3.2

Author Trevor Hastie [aut, cre],
Rob Tibshirani [aut],
Sourav Chatterjee [aut]

Maintainer Trevor Hastie <hastie@stanford.edu>

Repository CRAN

Date/Publication 2026-01-26 17:00:02 UTC

Contents

`ci.uniReg` *Compute bootstrap confidence intervals for a univariate guided regression model*

Description

Fit a univariate-guided sparse regression (lasso), by a two-stage procedure. The first stage fits p separate univariate models to the response. The second stage gives more weight to the more important univariate features, and preserves their signs. Conveniently, it returns an objects that inherits from `glmnet`, so that all of the methods for `glmnet` can be applied, such as `predict`, `plot`, `coef` and`print`.

Fit a univariate-guided regression, by a two-stage procedure. The first stage fits p separate univariate models to the response. The second stage fits a regression model, preserving the univariate signs.

Usage

```
ci.uniReg(
  x,
  y,
  family = c("gaussian", "binomial", "cox"),
  weights = NULL,
  B = 500,
  alpha = 0.05,
  ...
)

uniLasso(
  x,
  y,
  family = c("gaussian", "binomial", "cox"),
  weights = NULL,
  loo = TRUE,
  lower.limits = 0,
  standardize = FALSE,
  info = NULL,
  loob.nit = 2,
  loob.ridge = 0,
  loob.eps = 1e-06,
```

```
  ...
)

uniReg(
  x,
  y,
  family = c("gaussian", "binomial", "cox"),
  weights = NULL,
  loo = TRUE,
  lower.limits = 0,
  standardize = FALSE,
  info = NULL,
  loob.nit = 2,
  loob.ridge = 0,
  loob.eps = 1e-04,
  hard.zero = TRUE,
  ...
)
```

Arguments

x	Input matrix, of dimension nobs x nvars; each row is an observation vector.
y	Response variable. Quantitative for family = "gaussian" or family = "poisson" (non-negative counts). For family="binomial", should be a numeric vector consisting of 0s and 1s. For family="cox", y should be a two-column matrix with columns named 'time' and 'status'. The latter is a binary variable, with '1' indicating death, and '0' indicating right-censored.
family	one of "gaussian","binomial" or "cox". Currently only these families are implemented. In the future others will be added.
weights	optional vector of non-negative weights, default is NULL which results in all weights = 1.
B	Number of bootstrap samples. Default is 500.
alpha	size of confidence interval.
loo	TRUE (the default) means that uniLasso uses the prevalidated loo fits (approximate loo or 'alo' for "binomial" and "cox") for each univariate model as features to avoid overfitting. loo=FALSE means it uses the univariate fitted predictor.
lower.limits	= 0 (default) means that uniLasso constrains the sign of the coefs produced in the second round to be the same as those in the univariate fits. (Since uniLasso uses the univariate *fits* as features, a positivity constraint at the second stage is equivalent.)
standardize	input argument to glmnet for final non-negative lasso fit. Strongly recommend standardize=FALSE (default) since the univariate fit determines the correct scale for each variable.
info	Users can supply results of uniInfo on external datasets rather than compute them on the same data used to fit the model.

loob.nit	Number of Newton iterations for GLM or Cox in computing univariate linear predictors. Default is 2.
loob.ridge	A nonnegative number to apply ridge penalization to the slope parameters. This is helpful if some of the variables are near constant or have very small standard deviations. Default is 0.0.
loob.eps	A small number used in regularizing the Hessian for the Cox model. Default is 1e-6.
hard.zero	if TRUE (default), the model fits the unpenalized regression. This is potentially unstable when p > n. In this case hard.zero = FALSE might be preferable, and the model is then fit using the smallest value of lambda in the path.
...	additional arguments passed to glmnet.

Details

Fits a two stage lasso model. First stage replaces each feature by the univariate fit for that feature. Second stage fits a (positive) lasso using the first stage features (which preserves the signs of the first stage model). Hence the second stage selects and modifies the coefficients of the first stage model, similar to the adaptive lasso. Leads to sparser and more interpretable models.

For "binomial" family y is a binary response. For "cox" family, y should be a Surv object for right censored data, or a matrix with columns labeled 'time' and 'status' Although glmnet has more flexible options say for binary responses, and for cox responses, these are not yet implemented (but are possible and will appear in future versions). Likewise, other glm families are possible as well, but not yet implemented.

loo = TRUE means it uses the prevalidated loo fits (approximate loo or 'alo' for binomial and cox) for each univariate model as features to avoid overfitting in the second stage. The coefficients are then multiplied into the original univariate coefficients to get the final model.

loo = FALSE means it uses the univariate fitted predictor, and hence it is a form of adaptive lasso, but tends to overfit. lower.limits = 0 means uniLasso constrains the sign of the coefs in the second round to be that of the univariate fits.

Value

An object that inherits from "glmnet". There is one additional parameter returned, which is info and has two components. They are beta0 and beta, the intercepts and slopes for the usual (non-LOO) univariate fits from stage 1.

Examples

```
# ci.uniReg usage

sigma =3
set.seed(1)
n <- 100; p <- 20
x <- matrix(rnorm(n * p), n, p)
beta <- matrix(c(rep(2, 5), rep(0, 15)), ncol = 1)
y <- x %*% beta + rnorm(n)*sigma
ci <- ci.uniReg(x, y, B=100)
print(ci)
```

```
# uniLasso usage
# Default uniLasso usage for Gaussian data

sigma =3
set.seed(1)
n <- 100; p <- 20
x <- matrix(rnorm(n * p), n, p)
beta <- matrix(c(rep(2, 5), rep(0, 15)), ncol = 1)
y <- x %*% beta + rnorm(n)*sigma
xtest=matrix(rnorm(n * p), n, p)
ytest <- xtest %*% beta + rnorm(n)*sigma

fit <- uniLasso(x, y)
plot(fit)
predict(fit,xtest[1:10,],s=1) #predict on test data

# Two-stage variation where we carve off a small dataset for computing the univariate coefs.

cset=1:20
info = uniInfo(x[cset,],y[cset])
fit_two_stage <- uniLasso(x[-cset,], y[-cset], info = info)
plot(fit_two_stage)

# Binomial response uniLasso

yb =as.numeric(y>0)
fitb = uniLasso(x, y)
predict(fitb, xtest[1:10,], s=1, type="response")

# uniLasso with same positivity constraints, but starting `beta`
# from univariate fits on the same data. With loo=FALSE, does not tend to do as well,
# probably due to overfitting.

 fit_pos_adapt <- uniLasso(x, y, loo = FALSE)
 plot(fit_pos_adapt)

# uniLasso with no constraints, but starting `beta` from univariate fits.
# This is a version of the adaptive lasso, which tends to overfit, and loses interpretability.

 fit_adapt <- uniLasso(x, y, loo = FALSE, lower.limits = -Inf)
 plot(fit_adapt)

# Cox response uniLasso

set.seed(10101)
N = 1000
p = 30
nzc = p/3
x = matrix(rnorm(N * p), N, p)
beta = rnorm(nzc)
fx = x[, seq(nzc)] %*% beta/3
hx = exp(fx)
```

```
ty = rexp(N, hx)
tcens = rbinom(n = N, prob = 0.3, size = 1)  # censoring indicator
y = cbind(time = ty, status = 1 - tcens)  # y=Surv(ty,1-tcens) with library(survival)
fitc = uniLasso(x, y, family = "cox")
plot(fitc)

# uniReg usage

sigma =3
set.seed(1)
n <- 100; p <- 20
x <- matrix(rnorm(n * p), n, p)
beta <- matrix(c(rep(2, 5), rep(0, 15)), ncol = 1)
y <- x %*% beta + rnorm(n)*sigma
xtest=matrix(rnorm(n * p), n, p)

fit <- uniReg(x, y)
predict(fit,xtest[1:10,]) #predict on test data
coef(fit)
print(fit)

fita <- uniReg(x, y, hard.zero = FALSE)
print(fita)

fitb <- uniReg(x, y>0, family = "binomial")
coef(fitb)
print(fitb)
```

`cv.uniLasso` *Fit a cross-validated univariate guided lasso model.*

Description

Fit a univariate-guided sparse regression `uniLasso` model using cross-validation to select the second stage lasso penalty parameter. Conveniently, it returns an object that inherits from `cv.glmnet`, so that all of the methods for `cv.glmnet` can be applied, such as `predict`, `plot`, `coef`, `print`, and `assess.glmnet`.

Fit a cross-validated univariate-guided sparse regression `uniLasso` model, with a focus on the end of the path which corresponds to the `uniReg` fit. Conveniently, it returns an object that inherits from `cv.glmnet`,and methods such as `predict`, `plot`, `coef`, `print` all gives sensible results.

Usage

```
cv.uniLasso(
  x,
  y,
  family = c("gaussian", "binomial", "cox"),
  weights = NULL,
```

```
  loo = TRUE,
  lower.limits = 0,
  standardize = FALSE,
  info = NULL,
  loob.nit = 2,
  loob.ridge = 0,
  loob.eps = 1e-06,
  ...
)

cv.uniReg(
  x,
  y,
  family = c("gaussian", "binomial", "cox"),
  weights = NULL,
  loo = TRUE,
  lower.limits = 0,
  standardize = FALSE,
  info = NULL,
  loob.nit = 2,
  loob.ridge = 0,
  loob.eps = 1e-06,
  hard.zero = TRUE,
  ...
)
```

Arguments

x	Input matrix, of dimension nobs x nvars; each row is an observation vector.
y	Response variable. Quantitative for family = "gaussian" or family = "poisson" (non-negative counts). For family="binomial", should be a numeric vector consisting of 0s and 1s. For family="cox", y should be a two-column matrix with columns named 'time' and 'status'. The latter is a binary variable, with '1' indicating death, and '0' indicating right-censored.
family	one of "gaussian","binomial" or "cox". Currently only these families are implemented. In the future others will be added.
weights	optional vector of non-negative weights, default is NULL which results in all weights = 1.
loo	TRUE (the default) means that uniLasso uses the prevalidated loo fits (approximate loo or 'alo' for "binomial" and "cox") for each univariate model as features to avoid overfitting. loo=FALSE means it uses the univariate fitted predictor.
lower.limits	= 0 (default) means that uniLasso constrains the sign of the coefs in the second round to be that of the univariate fits.
standardize	input argument to glmnet for final non-negative lasso fit. Strongly recommend standardize=FALSE (default) since the univariate fit determines the right scale for each variable.

info	Users can supply results of uniInfo on external datasets rather than compute them on the same data used to fit the model. If this is supplied, its $betas are used. Default is NULL.
loob.nit	Number of Newton iterations for GLM or Cox in computing univariate linear predictors. Default is 2.
loob.ridge	A nonnegative number to apply ridge penalization to the slope parameters. This is helpful if some of the variables are near constant or have very small standard deviations. Default is 0.0.
loob.eps	A small number used in regularizing the Hessian for the Cox model. Default is 0.0001.
hard.zero	if TRUE (default), the model fits the unpenalized regression. This is potentially unstable when p > n. In this case hard.zero = FALSE might be preferable, and the model is then fit using the smallest value of lambda in the path.
...	additional arguments passed to cv.glmnet.

Details

Fits a two stage lasso model and selects the penalty parameter by cross validation. First stage replaces each feature by the univariate fit for that feature. Second stage fits a (positive) lasso using the first stage features. Hence the second stage selects and modifies the coefficients of the first stage model, similar to the adaptive lasso. Leads to potentially sparser models.

For "binomial" family y is a binary response. For "cox" family, y should be a Surv object for right censored data, or a matrix with columns labeled 'time' and 'status' Although glmnet has more flexible options say for binary responses, and for cox responses, these are not yet implemented (but are possible and will appear in future versions). Likewise, other glm families are possible as well, but not yet implemented.

This is a one-visit function that returns a 'cv.glmnet' object. You can make predictions from the whole path, at 'lambda.min' etc just like you can for a 'cv.glmnet object'.

loo = TRUE means it uses the prevalidated loo fits (approximate loo or 'alo' for binomial and cox) for each univariate model as features to avoid overfitting in the second stage. The coefficients are then multiplied into the original univariate coefficients to get the final model.

loo = FALSE means it uses the univariate fitted predictor, and hence it is a form of adaptive lasso, but tends to overfit. lower.limits = 0 means uniLasso constrains the sign of the coefs in the second round to be that of the univariate fits.

Value

An object that inherits from class "cv.glmnet". There is one additional parameter returned, which is info and has two components. They are beta0 and beta, the intercepts and slopes for the usual (non-LOO) univariate fits from stage 1.

Examples

```
# cv.uniLasso examples
# Default usage with Gaussian data

sigma =3
```

```
set.seed(1)
n <- 100; p <- 20
x <- matrix(rnorm(n * p), n, p)
beta <- matrix(c(rep(2, 5), rep(0, 15)), ncol = 1)
y <- x %*% beta + rnorm(n)*sigma
xtest=matrix(rnorm(n * p), n, p)
ytest <- xtest %*% beta + rnorm(n)*sigma

cvfit <- cv.uniLasso(x, y)
plot(cvfit)
predict(cvfit,xtest[1:10,], s="lambda.min") # predict at some test data points

# Two-stage variation where we carve off a small dataset for computing the univariate coefs.

cset=1:20
info = uniInfo(x[cset,],y[cset])
cvfit_two_stage <- cv.uniLasso(x[-cset,], y[-cset], info = info)
plot(cvfit_two_stage)

# Binomial response cv.uniLasso

yb =as.numeric(y>0)
cvfitb = cv.uniLasso(x, yb, family="binomial")
predict(cvfitb, xtest[1:10,], type="response") # predict at default s = "lambda.1se"


# cv.uniLasso with same positivity constraints, but starting `beta`
# from univariate fits on the same data. With loo=FALSE, does not tend to do as well,
# probably due to overfitting.

 cvfit_pos_adapt <- cv.uniLasso(x, y, loo = FALSE)
 plot(cvfit_pos_adapt)

# cv.uniLasso with no constraints, but starting `beta` from univariate fits.
# This is a version of the adaptive lasso, which tends to overfit, and loses interpretability.

 cvfit_adapt <- cv.uniLasso(x, y, loo = FALSE, lower.limits = -Inf)
 plot(cvfit_adapt)

# Cox response cv.uniLasso

set.seed(10101)
N = 1000
p = 30
nzc = p/3
x = matrix(rnorm(N * p), N, p)
beta = rnorm(nzc)
fx = x[, seq(nzc)] %*% beta/3
hx = exp(fx)
ty = rexp(N, hx)
tcens = rbinom(n = N, prob = 0.3, size = 1)  # censoring indicator
y = cbind(time = ty, status = 1 - tcens)  # y=Surv(ty,1-tcens) with library(survival)
cvfitc = cv.uniLasso(x, y, family = "cox")
```

```
plot(cvfitc)

# cv.uniReg usage

sigma =3
set.seed(1)
n <- 100; p <- 20
x <- matrix(rnorm(n * p), n, p)
beta <- matrix(c(rep(2, 5), rep(0, 15)), ncol = 1)
y <- x %*% beta + rnorm(n)*sigma
xtest=matrix(rnorm(n * p), n, p)


fit <- cv.uniReg(x, y)
plot(fit)
coef(fit)
print(fit)
predict(fit,xtest[1:10,]) #predict on test data

fita <- cv.uniReg(x, y, hard.zero = FALSE)
plot(fita)
print(fita)

fitb <- cv.uniReg(x, y>0, family = "binomial")
plot(fitb)
print(fitb)
```

`plot.cv.uniReg` *plot the cross-validation curve produced by cv.uniReg*

Description

Plots the cross-validation `cv.uniLasso` curve (which is a `cv.glmnet` curve), and upper and lower standard deviation curves, as a function of the `lambda` values used. It highlights the value at the end of the path, which is either `lambda = 0`, or else the smallest `lambda` if `hard.zero = FALSE`.

Usage

```
## S3 method for class 'cv.uniReg'
plot(x, ...)
```

Arguments

`x`	fitted `"cv.uniReg"` object, which inherits from `"cv.uniLasso"`. negative if `sign.lambda=-1`.
`...`	Other graphical parameters to plot

Details

A plot is produced, and nothing is returned.

Value

A plot is produced, and nothing is returned.

Author(s)

Trevor Hastie and Rob Tibshirani
Maintainer: Trevor Hastie hastie@stanford.edu

See Also

`cv.uniLasso` and `glmnet:::cv.glmnet`.

Examples

```
set.seed(1010)
n = 1000
p = 100
nzc = trunc(p/10)
x = matrix(rnorm(n * p), n, p)
beta = rnorm(nzc)
fx = (x[, seq(nzc)] %*% beta)
eps = rnorm(n) * 5
y = drop(fx + eps)
cvob0 = cv.uniReg(x, y)
plot(cvob0)
cvob = cv.uniReg(x, y, hard.zero = FALSE)
plot(cvob)
```

`polish.uniLasso` *Fit a cross-validated univariate guided lasso model, followed by a lasso polish.*

Description

This function has two stages. In the first we fit a univariate-guided sparse regression `uniLasso` model using cross-validation to select the lasso penalty parameter (using `s = "lambda.min"`). In the second stage, we use the predictions from this chosen model as an offset, and fit a cross-validated unrestricted lasso model. For squared error loss, this means we post-fit a lasso model to the residuals. Conveniently, it returns an object that inherits from `cv.glmnet`, in which the two models are *stitched* together. What this means is that the chosen coefficients from the first model are added to the coefficients from the second, and other related components are updated as well. This means at predict time we do not have to fiddle with offsets. All of the methods for `cv.glmnet` can be applied, such as `predict`, `plot`, `coef`, `print`, and `assess.glmnet`.

Usage

```
polish.uniLasso(
  x,
  y,
  family = c("gaussian", "binomial", "cox"),
  weights = NULL,
  ...
)
```

Arguments

x	Input matrix, of dimension nobs x nvars; each row is an observation vector.
y	Response variable. Quantitative for family = "gaussian" or family = "poisson" (non-negative counts). For family="binomial", should be a numeric vector consisting of 0s and 1s. For family="cox", y should be a two-column matrix with columns named 'time' and 'status'. The latter is a binary variable, with '1' indicating death, and '0' indicating right-censored.
family	one of "gaussian","binomial" or "cox". Currently only these families are implemented. In the future others will be added.
weights	optional vector of non-negative weights, default is NULL which results in all weights = 1.
...	additional arguments passed to cv.uniLasso and cv.glmnet. Note: by defaults cv.uniLasso() uses standardize=FALSE, and cv.glmnet() uses standardize=TRUE. These are both the sensible defaults for this function. Users can supply standardize=FALSE (via the ... argument) which will overide the cv.glmnet() default. Users should avoid using standardize=TRUE, since this will affect the first stage model as well, where this is not a suitable choice.

Value

An object of class "polish.unilasso" that inherits from class "cv.glmnet". The "glmnet.fit" is the *stitched* second-stage model, from which predictions are made. An additional component named "cv.uniLasso" is the first stage model.

Examples

```
# Gaussian data, p=1000, n=300, SNR=1  "medium SNR"
# use the built-in simulate function to create Gaussian data
set.seed(101)
data <- simulate_uniLasso("medium-SNR")
attach(data) # has components "x","y","xtest","ytest","mutest","sigma"
pfit <- polish.uniLasso(x,y)
plot(pfit)
pred <- predict(pfit, newx = xtest,  s = "lambda.min") # ie predict from a "cv.glmnet" object.
mean((ytest-pred)^2) # test error
print(pfit)
print(pfit$glmnet.fit)
plot(pfit$glmnet.fit) # coefficient plot of the second stage
```

```
plot(pfit$cv.uniLasso) # cv.glmnet plot of the first stage
plot(pfit$cv.uniLasso$glmnet.fit) # coefficient plot of the first stage


# Binomial response

yb =as.numeric(y>0)
pfitb = polish.uniLasso(x, yb, family="binomial")
predict(pfitb, xtest[1:10,], type="response") # predict at default s = "lambda.1se"
plot(pfitb)
plot(pfitb$glmnet.fit) # plot second stage lasso coefficient path
plot(pfitb$cv.uniLasso) # plot first stage cv.uniLasso results
# Cox response

set.seed(10101)
N = 1000
p = 30
nzc = p/3
x = matrix(rnorm(N * p), N, p)
beta = rnorm(nzc)
fx = x[, seq(nzc)] %*% beta/3
hx = exp(fx)
ty = rexp(N, hx)
tcens = rbinom(n = N, prob = 0.3, size = 1)  # censoring indicator
y = cbind(time = ty, status = 1 - tcens)  # y=Surv(ty,1-tcens) with library(survival)
pfitc = polish.uniLasso(x, y, family = "cox")
plot(pfitc)
plot(pfitc$cv.uniLasso)
```

`predict.cv.uniReg` *make predictions from a "cv.uniReg" object.*

Description

This function makes predictions from a cross-validated `uniReg` model, using the stored `"glmnet.fit"` object, and by default the smallest value of `lambda` used.

Usage

```
## S3 method for class 'cv.uniReg'
predict(object, newx, s = c("zero", "lambda.1se", "lambda.min"), ...)
```

Arguments

`object`	Fitted `"cv.uniReg"`.
`newx`	Matrix of new values for `x` at which predictions are to be made. Must be a matrix; can be sparse as in `Matrix` package. See documentation for `predict.glmnet`.

s	Value(s) of the penalty parameter lambda at which predictions are required. Default is the value s="zero" which corresponds to the smallest value of lambda used. Alternatively s="lambda.1se" or s="lambda.min" can be used. If s is numeric, it is taken as the value(s) of lambda to be used.
...	Not used. Other arguments to predict.

Details

This function makes it easier to use the results of cross-validation to make a prediction.

Value

The object returned depends on the ... argument which is passed on to the predict method for glmnet objects.

Author(s)

Trevor Hastie and Rob Tibshirani
Maintainer: Trevor Hastie hastie@stanford.edu

See Also

print, and coef methods, and cv.uniReg.

Examples

```
x = matrix(rnorm(100 * 20), 100, 20)
y = rnorm(100)
cv.fit = cv.uniReg(x, y)
predict(cv.fit, newx = x[1:5, ])
coef(cv.fit)
```

---

print.cv.uniReg *print a cross-validated uniReg object*

Description

Print a summary of the results of cross-validation for a uniReg model.

Usage

```
## S3 method for class 'cv.uniReg'
print(x, digits = max(3, getOption("digits") - 3), ...)
```

Arguments

x	fitted 'cv.uniReg' object
digits	significant digits in printout
...	additional print arguments

Details

A summary of the cross-validated uniReg fit is produced. This is an augmented summary of a cv.glmnet object, with an extra row corresponding to the smallest lambda in the path

Value

A summary is printed, and nothing is returned.

Author(s)

Trevor Hastie and Rob Tibshirani
Maintainer: Trevor Hastie hastie@stanford.edu

Examples

```
x = matrix(rnorm(100 * 20), 100, 20)
y = rnorm(100)
fit1 = cv.uniReg(x, y)
print(fit1)
```

---

simulate_counterexample

simulate counterexample data

Description

A particular counterexample where the first two features are strongly positively correlated, yet they have coefficients of opposite sign in a multiple regression.

Usage

```
simulate_counterexample(ntrain, ntest)
```

Arguments

ntrain	number of training examples.
ntest	number of test examples.

Value

a list with components "x", "y", "xtest", "ytest", "mutest", and "sigma", where "mutest" is the true test mean, and "ytest <- mutest + rnorm(ntest)*sigma."

Examples

```
dat = simulate_counterexample(300,3000)
fit = cv.uniLasso(dat$x, dat$y)
err = mean( (predict(fit, dat$xtest,s="lambda.min")- dat$mutest)^2)
```

`simulate_Gaussian` *simulate Gaussian data*

Description

A simulator that builds a training and test set with particular characteristics, as used in our "uniLasso" paper.

Usage

```
simulate_Gaussian(
  ntrain = 300,
  ntest = 3000,
  p = 1000,
  snr = 1,
  rho = 0.8,
  sparsity = 0.1,
  homecourt = FALSE
)
```

Arguments

`ntrain`	number of training examples.
`ntest`	number of test examples.
`p`	number of features.
`snr`	desired SNR (signal-to-noise ratio).
`rho`	for `homecourt=TRUE` 'rho' controls the autocorrelation between variables. Variables k units apart have correlation `rho^k`.
`sparsity`	fraction of variables with nonzero coefficients.
`homecourt`	logical; if `TRUE` then correlated features, with a special boost for large coefficients, mimicking the uniLasso two-stage algorithm.

Value

a list with components "x", "y", "xtest", "ytest", "mutest", and "sigma", where "mutest" is the true test mean, and "ytest <- mutest + rnorm(ntest)*sigma."

Examples

```
dat = simulate_Gaussian(300,3000,p=500,snr=1.2)
fit = cv.uniLasso(dat$x, dat$y)
mse = mean( (predict(fit, dat$xtest)- dat$mutest)^2)
```

simulate_twoclass *simulate two class data*

Description

simulate two class data

Usage

```
simulate_twoclass(ntrain, ntest, wide = TRUE)
```

Arguments

ntrain	number of training examples.
ntest	number of test examples.
wide	logical. If TRUE p=500, else p=100.

Value

a list with components "x", "y", "xtest", "ytest", "mutest", and "sigma", where "mutest" is the true test mean, and "ytest <- mutest + rnorm(ntest)*sigma."

Examples

```
dat = simulate_twoclass(300,3000)
fit = cv.uniLasso(dat$x, dat$y, family="binomial")
misclass = mean( sign(predict(fit, dat$xtest,s="lambda.min"))== sign(dat$ytest-0.5))
```

simulate_uniLasso *Simulate data for use in uniLasso and uniReg*

Description

We use some standard examples in our uniLasso paper, and for convenience we provide generators for these datasets.

Usage

```
simulate_uniLasso(
  example = c("low-SNR", "medium-SNR", "high-SNR", "home-court", "two-class",
    "counter-example"),
  wide = TRUE
)
```

Arguments

example	which of the prepackaged examples to use. Choices are "low-SNR","medium-SNR","high-SNR","home-court","two-class","counter-example", as described in the uniLasso paper. The three SNRs used are 0.5 (low), 1.0 (medium) and 2.0 (high) (also used for home-court). The training sizes for the first four are 300, and test sizes 3000.
wide	logical variable which determines if p>n (default, 1000) or not (100). This function calls worker functions simulate_gaussian(), simulate_two-class(), and simulate_counterexample(), which are currently not documented.

Value

a list with components "x", "y", "xtest", "ytest", "mutest", and "sigma", where "mutest" is the true test mean, and "ytest <- mutest + rnorm(nrow(xtest))*sigma."

Examples

```
dat = simulate_uniLasso("high-SNR")
fit = cv.uniLasso(dat$x, dat$y)
mse = mean( (predict(fit, dat$xtest)- dat$mutest)^2)
```

---

uniCoef *Compare the nonzero coefficients and univariate counterparts*

Description

For a `cv.uniLasso` object, compare the CV-selected nonzero coefficients to their univariate counterparts. Also works for a `cv.glmnet` object.

Usage

```
uniCoef(cv.object, info = NULL, s = c("lambda.min", "lambda.1se"), ...)
```

Arguments

cv.object	a `cv.uniLasso` or a `cv.glmnet` object.
info	the result of a call to `uniInfo()`. If `cv.object` inherits from `cv.uniLasso`, the `$info` component will be used.
s	the value of lambda to be used, with default `s="lambda.min"`. Alternatively, can be 's="lambda.1se".
...	other arguments to `coef`.

Value

a three-columns data frame with the second column being the non-zero coefficients from the `cv.object`, the first the corresponding univariate coefficients, and the third an indication if there was a sign change.

@examples

sigma <- 3 set.seed(1) n <- 100 p <- 20 x <- matrix(rnorm(n * p), n, p) beta <- matrix(c(rep(2, 5), rep(0, 15)), ncol = 1) y <- x %*% beta + rnorm(n)*sigma

cvfit <- cv.uniLasso(x, y) uniCoef(cvfit) cvfit2 <- cv.glmnet(x,y) uniCoef(cvfit2, info=cvfit$info)

`uniInfo` *Create the univariate info for use in uniLasso*

Description

Fit p separate univariate fits, and if requested computes the loo fit matrix F. It is called internally by `uniLasso`, or can be called externally on separate data and passed as input to `uniLasso`. Currently this function can accommodate "gaussian", "binomial", and "Cox" families.

Usage

```
uniInfo(
  X,
  y,
  family = c("gaussian", "binomial", "cox"),
  weights = NULL,
  nit = 2,
  loo = FALSE,
  ridge = 0,
  eps = 1e-06
)
```

Arguments

`X`	An n x p feature matrix
`y`	A response object, depending on the family. For "gaussian" it is just a response vector. For "binomial" either a binary vector, a two level factor, or a two column non-negative matrix with rows summing to 1. For "cox" it is a Surv object (currently for right censored data).
`family`	one of "gaussian","binomial" or "cox". Currently only these families are implemented. In the future others will be added.
`weights`	Vector of non-negative weights. Default is NULL, which results in all weights equal to 1.
`nit`	Number of iterations if Newton steps are required (in "binomial" and "cox"). Default is 2. In principal more is better, but in some cases can run into convergence issues.

loo	A logical, default=FALSE. If TRUE it computes the matrix of loo fits F.
ridge	A positive number that penalizes the square of the slope parameters. This is useful if some of the variables are nearly constant, or have very small variances. Default is 0.0.
eps	A small number to regularize the hessian for "cox"; default is 1e-6.

Value

a list with components $beta and $beta0, and if loo=TRUE, a n x p matrix F with the loo fits.

Examples

```
# Gaussian model
set.seed(1)
sigma=3
n <- 100; p <- 20
x <- matrix(rnorm(n * p), n, p)
beta <- matrix(c(rep(2, 5), rep(0, 15)), ncol = 1)
y <- x %*% beta + rnorm(n)*sigma


info = uniInfo(x,y)
names(info)

yb = as.numeric(y>0)
info = uniInfo(x,yb, family = "binomial", loo = TRUE)
names(info)
```

Index