

Package: ptLasso (via r-universe)

October 27, 2026

Type Package

Title Pretraining the Lasso

Description Implements lasso pretraining for generalized linear models for data with grouped observations; data without grouped observations, but with multinomial responses; data with multiple Gaussian responses; and data with repeated measurements over time. The method fits a lasso model to an initial dataset and then uses its support and coefficient values to guide model fitting for a related task, which can improve performance by leveraging larger datasets and preserve interpretability by identifying features which are generally shared, and which are task specific. See Craig et al. (2026) <[doi:10.1093/jrsssb/qkaf050](https://doi.org/10.1093/jrsssb/qkaf050)> for details. Model fitting in this package is done with `cv.glmnet()` from the 'glmnet' package, and syntax closely follows that of 'glmnet' (Friedman, Hastie and Tibshirani (2010) <[doi:10.18637/jss.v033.i01](https://doi.org/10.18637/jss.v033.i01)>).

Version 1.0.2

Maintainer Erin Craig <ercr@umich.edu>

License GPL-2

Encoding UTF-8

URL <https://erincr.github.io/ptLasso/>,
<https://doi.org/10.1093/jrsssb/qkaf050>

BugReports <https://github.com/erincr/ptLasso/issues>

NeedsCompilation yes

RoxygenNote 7.3.3

Depends ggplot2, glmnet, gridExtra, Matrix

Suggests knitr, rmarkdown, rpart, survival, xgboost, sparsepca, MASS, testthat

VignetteBuilder knitr

Language en

Author Erin Craig [aut, cre, cph] (ORCID:
<<https://orcid.org/0000-0002-4893-6671>>), Rob Tibshirani [aut,
cph]
Repository <https://cran.r-universe.dev>
Date/Publication 2026-10-27 18:49:47 UTC
RemoteUrl <https://github.com/cran/ptLasso>
RemoteRef HEAD
RemoteSha 9261ccbc1b9aa0e87a5d65d45e98ef5ebffe974d

Contents

binomial.example.data	2
coef.cv.ptLasso	4
coef.ptLasso	5
cv.ptLasso	6
gaussian.example.data	12
get.individual.support	14
get.overall.support	15
get.pretrain.support	16
makedata	18
makedata.targetgroups	20
plot.cv.ptLasso	21
plot.ptLasso	22
predict.cv.ptLasso	23
predict.ptLasso	25
print.cv.ptLasso	27
print.predict.cv.ptLasso	28
print.predict.ptLasso	29
print.ptLasso	30
ptLasso	31
Index	39

binomial.example.data	<i>Simulate input grouped data (binomial outcome) for testing with pt-Lasso.</i>
-----------------------	--

Description

No required arguments; used primarily for documentation. Simply calls makedata with a reasonable set of features.

Usage

```

binomial.example.data(
  k = 3,
  class.sizes = rep(100, k),
  n = sum(class.sizes),
  scommon = 5,
  sindiv = rep(5, k),
  p = 2 * (sum(sindiv) + scommon),
  beta.common = list(c(-0.5, 0.5, 0.3, -0.9, 0.1), c(-0.3, 0.9, 0.1, -0.1, 0.2), c(0.1,
    0.2, -0.1, 0.2, 0.3)),
  beta.indiv = lapply(1:k, function(i) 0.9 * beta.common[[i]]),
  intercepts = rep(0, k),
  sigma = NULL
)

```

Arguments

k	Default: 3.
class.sizes	Default: rep(100, k).
n	Default: sum(class.sizes).
scommon	Default: 5.
sindiv	Default: rep(5, k).
p	Default: 2*(sum(sindiv) + scommon).
beta.common	Default: list(c(-.5, .5, .3, -.9, .1), c(-.3, .9, .1, -.1, .2), c(0.1, .2, -.1, .2, .3)).
beta.indiv	Default: lapply(1:k, function(i) 0.9 * beta.common[[i]]).
intercepts	Default: rep(0,k).
sigma	Default: NULL.

Value

A list for data with 5 groups and a binomial outcome, n=300 and p=40:

x	Simulated features, size n x p.
y	Outcomes y, length n.
groups	Vector of length n, indicating which observations belong to which group.
snr	Gaussian outcome only: signal to noise ratio.
mu	Gaussian outcome only: the value of y before noise is added.

Author(s)

Erin Craig and Rob Tibshirani
 Maintainer: Erin Craig <ercr@umich.edu>

See Also

cv.ptLasso, ptLasso.

Examples

```
out = binomial.example.data()
x = out$x; y=out$y; groups = out$group
```

code: `coef.cv.ptLasso`
Get the coefficients from a fitted cv.ptLasso model.

Description

Get the coefficients from a fitted cv.ptLasso model.

Usage

```
## S3 method for class 'cv.ptLasso'
coef(
  object,
  model = c("all", "individual", "overall", "pretrain"),
  alpha = NULL,
  ...
)
```

Arguments

<code>object</code>	fitted "cv.ptLasso" object.
<code>model</code>	string indicating which coefficients to retrieve. Must be one of "all", "individual", "overall" or "pretrain".
<code>alpha</code>	value between 0 and 1, indicating which alpha to use. If NULL, return coefficients from all models. Only impacts the results for <code>model = "all"</code> or <code>model = "pretrain"</code> .
<code>...</code>	other arguments to be passed to the "coef" function. May be e.g. <code>s = "lambda.min"</code> .

Value

Model coefficients. If `model = "overall"`, this returns the output of `coef` for the overall model. If `model` is "individual", this returns a list containing the results of `coef` for each group-specific (or response-specific) model. If `model = "pretrain"`, this returns the pretrained model coefficients: a list with one entry per value of `alpha` if `alpha = NULL`, otherwise the coefficients for the chosen `alpha`. If `model = "all"`, this returns a list containing all (overall, individual and pretrain) coefficients. (For time series data, there is no overall model, so only individual and pretrain coefficients are returned.) Note that the pretrained coefficients are from the second stage of pretraining only: they do not include the offset from the overall model. The full pretrained linear predictor for a group (or response) is $(1 - \alpha)$ times the overall model's linear predictor (at `fitoverall.lambda`), plus the linear predictor from these coefficients. For time series data, the offset instead comes from the pretrained model for the previous time point.

Author(s)

Erin Craig and Rob Tibshirani
 Maintainer: Erin Craig <ercr@umich.edu>

See Also

cv.ptLasso, ptLasso.

Examples

```
set.seed(1234)
out = gaussian.example.data(k=2, class.sizes = c(50, 50))
x = out$x; y=out$y; groups = out$group;

cvfit = cv.ptLasso(x, y, groups = groups, family = "gaussian", type.measure = "mse")
# Get all model coefficients.
names(coef(cvfit))

coef(cvfit, model = "overall") # Overall model only
length(coef(cvfit, model = "individual")) # List of coefficients for each group model
length(coef(cvfit, model = "pretrain", alpha = .5)) # List of coefficients for each group model
```

coef.ptLasso

Get the coefficients from a fitted ptLasso model.

Description

Get the coefficients from a fitted ptLasso model.

Usage

```
## S3 method for class 'ptLasso'
coef(object, model = c("all", "individual", "overall", "pretrain"), ...)
```

Arguments

object	fitted "ptLasso" object.
model	string indicating which coefficients to retrieve. Must be one of "all", "individual", "overall" or "pretrain".
...	other arguments to be passed to the "coef" function. May be e.g. s = "lambda.min".

Value

Model coefficients. If `model = "overall"`, this function returns the output of `coef`. If `model` is "individual" or "pretrain", this function returns a list containing the results of `coef` for each group-specific model. If `model = "all"`, this returns a list containing all (overall, individual and pretrain) coefficients. Note that the pretrained coefficients are from the second stage of pretraining only: they do not include the offset from the overall model. The full pretrained linear predictor for a group (or response) is $(1 - \alpha)$ times the overall model's linear predictor (at `fitoverall.lambda`), plus the linear predictor from these coefficients. For time series data, the offset instead comes from the pretrained model for the previous time point.

Author(s)

Erin Craig and Rob Tibshirani
 Maintainer: Erin Craig <ercr@umich.edu>

See Also

ptLasso.

Examples

```
# Train data
out = gaussian.example.data()
x = out$x; y=out$y; groups = out$group;

fit = ptLasso(x, y, groups = groups, family = "gaussian", type.measure = "mse")
# Get all model coefficients.
names(coef(fit))

coef(fit, model = "overall") # Overall model only
length(coef(fit, model = "individual")) # List of coefficients for each group model
length(coef(fit, model = "pretrain")) # List of coefficients for each group model
```

cv.ptLasso

Cross-validation for ptLasso

Description

Cross-validation for ptLasso.

Usage

```
cv.ptLasso(
  x,
  y,
  groups = NULL,
  alphalist = seq(0, 1, length = 11),
```

```

family = c("default", "gaussian", "multinomial", "binomial", "cox"),
use.case = c("inputGroups", "targetGroups", "multiresponse", "timeSeries"),
type.measure = c("default", "mse", "mae", "auc", "deviance", "class", "C"),
nfolds = 10,
foldid = NULL,
verbose = FALSE,
fitoverall = NULL,
fitind = NULL,
s = "lambda.min",
gamma = "gamma.min",
alphahat.choice = "overall",
group.intercepts = TRUE,
...
)

```

Arguments

<code>x</code>	<code>x</code> matrix as in <code>ptLasso</code> .
<code>y</code>	<code>y</code> vector or matrix as in <code>ptLasso</code> .
<code>groups</code>	A vector of length <code>nobs</code> indicating to which group each observation belongs. For data with <code>k</code> groups, <code>groups</code> should be coded as integers 1 through <code>k</code> . Only for <code>'use.case = "inputGroups"</code> .
<code>alphalist</code>	A vector of values of the pretraining hyperparameter <code>alpha</code> . Defaults to <code>seq(0, 1, length.out=11)</code> . This function will do pretraining for each choice of <code>alpha</code> in <code>alphalist</code> and return the CV performance for each <code>alpha</code> .
<code>family</code>	Response type as in <code>ptLasso</code> .
<code>use.case</code>	The type of grouping observed in the data. Can be one of <code>"inputGroups"</code> , <code>"targetGroups"</code> , <code>"multiresponse"</code> or <code>"timeSeries"</code> .
<code>type.measure</code>	Measure computed in <code>cv.glmnet</code> , as in <code>ptLasso</code> .
<code>nfolds</code>	Number of folds for CV (default is 10). Although <code>nfolds</code> can be as large as the sample size (leave-one-out CV), it is not recommended for large datasets. Smallest value allowable is <code>nfolds = 3</code> .
<code>foldid</code>	An optional vector of values between 1 and <code>nfolds</code> identifying what fold each observation is in. If supplied, <code>nfolds</code> can be missing.
<code>verbose</code>	If <code>verbose=1</code> , print a statement showing which model is currently being fit.
<code>fitoverall</code>	An optional <code>cv.glmnet</code> object specifying the overall model. This should have been trained on the full training data, with the argument <code>keep = TRUE</code> .
<code>fitind</code>	An optional list of <code>cv.glmnet</code> objects specifying the individual models. These should have been trained on the training data, with the argument <code>keep = TRUE</code> .
<code>s</code>	The choice of <code>lambda</code> to be used by all models when estimating the CV performance for each choice of <code>alpha</code> . Defaults to <code>"lambda.min"</code> . May be <code>"lambda.1se"</code> , or a numeric value. (Use caution when supplying a numeric value: the same <code>lambda</code> will be used for all models.)
<code>gamma</code>	For use only when <code>relax = TRUE</code> . The choice of <code>gamma</code> to be used by all models when estimating the CV performance for each choice of <code>alpha</code> . Defaults to <code>"gamma.min"</code> . May also be <code>"gamma.1se"</code> .

<code>alphahat.choice</code>	When choosing <code>alphahat</code> , we may prefer the best performance using all data (<code>alphahat.choice = "overall"</code>) or the best average performance across groups (<code>alphahat.choice = "mean"</code>). This is particularly useful when <code>type.measure</code> is "auc" or "C", because the average performance across groups is different than the performance with the full dataset. The default is "overall".
<code>group.intercepts</code>	For <code>'use.case = "inputGroups"'</code> only. If <code>'TRUE'</code> , fit the overall model with a separate intercept for each group. If <code>'FALSE'</code> , ignore the grouping and fit one overall intercept. Default is <code>'TRUE'</code> .
<code>...</code>	Additional arguments to be passed to the <code>'cv.glmnet'</code> function. Notable choices include <code>"trace.it"</code> and <code>"parallel"</code> . If <code>trace.it = TRUE</code> , then a progress bar is displayed for each call to <code>cv.glmnet</code> ; useful for big models that take a long time to fit. If <code>parallel = TRUE</code> , use parallel foreach to fit each fold. Must register parallel before hand, such as <code>doMC</code> or others. Importantly, <code>"cv.ptLasso"</code> does not support the arguments <code>"intercept"</code> , <code>"offset"</code> , <code>"fit"</code> and <code>"check.args"</code> .

Details

This function runs `ptLasso` once for each requested choice of `alpha`, and returns the cross validated performance.

Value

An object of class `"cv.ptLasso"`, which is a list with the ingredients of the cross-validation fit.

<code>call</code>	The call that produced this object.
<code>alphahat</code>	Value of <code>alpha</code> that optimizes CV performance on all data.
<code>varying.alphahat</code>	Vector of values of <code>alpha</code> , the <code>k</code> th of which optimizes performance for group <code>k</code> .
<code>alphalist</code>	Vector of all <code>alphas</code> that were compared.
<code>errall</code>	CV performance for the overall model.
<code>errpre</code>	CV performance for the pretrained models (one for each <code>alpha</code> tried).
<code>errind</code>	CV performance for the individual model.
<code>fit</code>	List of <code>ptLasso</code> objects, one for each <code>alpha</code> tried.
<code>fitoverall</code>	The fitted overall model used for the first stage of pretraining.
<code>fitoverall.lambda</code>	The value of <code>lambda</code> used for the first stage of pretraining.
<code>fitind</code>	A list containing one individual model for each group.
<code>use.case</code>	The use case: <code>"inputGroups"</code> or <code>"targetGroups"</code> .
<code>family</code>	The family used.
<code>type.measure</code>	The <code>type.measure</code> used.

See Also

[ptLasso](#) and [plot.cv.ptLasso](#).

Examples

```

# Getting started. First, we simulate data: we need covariates x, response y and group IDs.
set.seed(1234)
n = 80
p = 20
x = matrix(rnorm(n*p), n, p)
y = rnorm(n)
groups = sort(rep(1:5, n/5))

xtest = matrix(rnorm(n*p), n, p)
ytest = rnorm(n)
groupstest = sort(rep(1:5, n/5))

# Model fitting
cvfit = cv.ptLasso(x, y, groups = groups, family = "gaussian", nfolds=3,
                  type.measure = "mse")
cvfit
plot(cvfit) # to see CV performance as a function of alpha
predict(cvfit, xtest, groupstest, s="lambda.min") # to predict with held out data
predict(cvfit, xtest, groupstest, s="lambda.min", ytest=ytest) # to also measure performance

# By default, we used s = "lambda.min" to compute CV performance.
# We could instead use s = "lambda.1se":
cvfit = cv.ptLasso(x, y, groups = groups, family = "gaussian", nfolds=3,
                  type.measure = "mse", s = "lambda.1se")

# We could have used the glmnet option relax = TRUE:
cvfit = cv.ptLasso(x, y, groups = groups, family = "gaussian", nfolds=3,
                  type.measure = "mse", relax = TRUE)
# And, as we did with lambda, we may want to specify the choice of gamma to compute CV performance:
cvfit = cv.ptLasso(x, y, groups = groups, family = "gaussian", nfolds=3,
                  type.measure = "mse", relax = TRUE, gamma = "gamma.1se")

# Note that the first stage of pretraining uses "lambda.1se" and "gamma.1se" by default.
# This behavior can be modified by specifying overall.lambda and overall.gamma;
# see the documentation for ptLasso for more information.

# Now, we are ready to simulate slightly more realistic data.
# This continuous outcome example has k = 5 groups, where each group has 200 observations.
# There are scommon = 10 features shared across all groups, and
# sindiv = 10 features unique to each group.
# n = 1000 and p = 120 (60 informative features and 60 noise features).
# The coefficients of the common features differ across groups (beta.common).
# In group 1, these coefficients are rep(1, 10); in group 2 they are rep(2, 10), etc.
# Each group has 10 unique features, the coefficients of which are all 3 (beta.indiv).
# The intercept in all groups is 0.
# The variable sigma = 20 indicates that we add noise to y according to 20 * rnorm(n).
set.seed(1234)
k=5
class.sizes=rep(200, k)

```

```

scommon=10; sindiv=rep(10, k)
n=sum(class.sizes); p=2*(sum(sindiv) + scommon)
beta.common=3*(1:k); beta.indiv=rep(3, k)
intercepts=rep(0, k)
sigma=20
out = gaussian.example.data(k=k, class.sizes=class.sizes,
                           scommon=scommon, sindiv=sindiv,
                           n=n, p=p,
                           beta.common=beta.common, beta.indiv=beta.indiv,
                           intercepts=intercepts, sigma=20)
x = out$x; y=out$y; groups = out$group

outtest = gaussian.example.data(k=k, class.sizes=class.sizes,
                              scommon=scommon, sindiv=sindiv,
                              n=n, p=p,
                              beta.common=beta.common, beta.indiv=beta.indiv,
                              intercepts=intercepts, sigma=20)
xtest=outtest$x; ytest=outtest$y; groupstest=outtest$groups

cvfit = cv.ptLasso(x, y, groups = groups, family = "gaussian", type.measure = "mse")
cvfit
# plot(cvfit) # to see CV performance as a function of alpha
predict(cvfit, xtest, groupstest, ytest=ytest, s="lambda.min")

# Now, we repeat with a binomial outcome.
# This example has k = 3 groups, where each group has 100 observations.
# There are scommon = 5 features shared across all groups, and
# sindiv = 5 features unique to each group.
# n = 300 and p = 40 (20 informative features and 20 noise features).
# The coefficients of the common features differ across groups (beta.common),
# as do the coefficients specific to each group (beta.indiv).
set.seed(1234)
k=3
class.sizes=rep(100, k)
scommon=5; sindiv=rep(5, k)
n=sum(class.sizes); p=2*(sum(sindiv) + scommon)
beta.common=list(c(-.5, .5, .3, -.9, .1), c(-.3, .9, .1, -.1, .2), c(0.1, .2, -.1, .2, .3))
beta.indiv = lapply(1:k, function(i) 0.9 * beta.common[[i]])

out = binomial.example.data(k=k, class.sizes=class.sizes,
                           scommon=scommon, sindiv=sindiv,
                           n=n, p=p,
                           beta.common=beta.common, beta.indiv=beta.indiv)
x = out$x; y=out$y; groups = out$group

outtest = binomial.example.data(k=k, class.sizes=class.sizes,
                              scommon=scommon, sindiv=sindiv,
                              n=n, p=p,
                              beta.common=beta.common, beta.indiv=beta.indiv)
xtest=outtest$x; ytest=outtest$y; groupstest=outtest$groups

```

```

cvfit = cv.ptLasso(x, y, groups = groups, family = "binomial",
                  type.measure = "auc", nfolds=3, verbose=TRUE,
                  alpha = c(0, .5, 1),
                  alphahat.choice="mean")

cvfit
# plot(cvfit) # to see CV performance as a function of alpha
predict(cvfit, xtest, groupstest, ytest=ytest, s="lambda.1se")

## Not run:
### Model fitting with parallel = TRUE
require(doMC)
registerDoMC(cores = 4)
cvfit = cv.ptLasso(x, y, groups = groups, family = "binomial",
                  type.measure = "auc", parallel=TRUE)

## End(Not run)

# Multiresponse pretraining
# Now let's consider the case of a multiresponse outcome. We'll start by simulating data:
set.seed(1234)
n = 1000; ntrain = 500;
p = 500
sigma = 2

x = matrix(rnorm(n*p), n, p)
beta1 = c(rep(1, 5), rep(0.5, 5), rep(0, p - 10))
beta2 = c(rep(1, 5), rep(0, 5), rep(0.5, 5), rep(0, p - 15))

mu = cbind(x %*% beta1, x %*% beta2)
y = cbind(mu[, 1] + sigma * rnorm(n),
          mu[, 2] + sigma * rnorm(n))
cat("SNR for the two tasks:", round(diag(var(mu)/var(y-mu)), 2), fill=TRUE)
cat("Correlation between two tasks:", cor(y[, 1], y[, 2]), fill=TRUE)

xtest = x[-(1:ntrain), ]
ytest = y[-(1:ntrain), ]

x = x[1:ntrain, ]
y = y[1:ntrain, ]

# Now, we can fit a ptLasso model:
fit = cv.ptLasso(x, y, type.measure = "mse", use.case = "multiresponse")
plot(fit) # to see the cv curve.
predict(fit, xtest) # to predict with new data
predict(fit, xtest, ytest=ytest) # if ytest is included, we also measure performance
# By default, we used s = "lambda.min" to compute CV performance.
# We could instead use s = "lambda.1se":
cvfit = cv.ptLasso(x, y, type.measure = "mse", s = "lambda.1se", use.case = "multiresponse")

# We could also use the glmnet option relax = TRUE:
cvfit = cv.ptLasso(x, y, type.measure = "mse", relax = TRUE, use.case = "multiresponse")
# And, as we did with lambda, we may want to specify the choice of gamma to compute CV performance:

```

```

cvfit = cv.ptLasso(x, y, type.measure = "mse", relax = TRUE, gamma = "gamma.1se",
                  use.case = "multiresponse")

# Time series pretraining
# Now suppose we have time series data with a binomial outcome measured at 3 different time points.
set.seed(1234)
n = 600; ntrain = 300; p = 50
x = matrix(rnorm(n*p), n, p)

beta1 = c(rep(0.25, 10), rep(0, p-10))
beta2 = beta1 + c(rep(0.1, 10), runif(5, min = -0.25, max = 0), rep(0, p-15))
beta3 = beta1 + c(rep(0.2, 10), runif(5, min = -0.25, max = 0),
                  runif(5, min = 0, max = 0.1), rep(0, p-20))

y1 = rbinom(n, 1, prob = 1/(1 + exp(-x %*% beta1)))
y2 = rbinom(n, 1, prob = 1/(1 + exp(-x %*% beta2)))
y3 = rbinom(n, 1, prob = 1/(1 + exp(-x %*% beta3)))
y = cbind(y1, y2, y3)

xtest = x[-(1:ntrain), ]
ytest = y[-(1:ntrain), ]

x = x[1:ntrain, ]
y = y[1:ntrain, ]

cvfit = cv.ptLasso(x, y, use.case="timeSeries", family="binomial",
                  type.measure="auc")
plot(cvfit, plot.alphahat = TRUE)
predict(cvfit, xtest, ytest=ytest)

# The glmnet option relax = TRUE:
cvfit = cv.ptLasso(x, y, type.measure = "auc", family = "binomial", relax = TRUE,
                  use.case = "timeSeries")

```

gaussian.example.data *Simulate input grouped data (gaussian outcome) for testing with pt-Lasso.*

Description

No required arguments; used primarily for documentation. Simply calls `makedata` with a reasonable set of features.

Usage

```

gaussian.example.data(
  k = 5,

```

```

    class.sizes = rep(200, k),
    n = sum(class.sizes),
    scommon = 10,
    sindiv = rep(10, k),
    p = 2 * (sum(sindiv) + scommon),
    beta.common = 3 * (1:k),
    beta.indiv = rep(3, k),
    intercepts = rep(0, k),
    sigma = 20
  )

```

Arguments

k	Default: 5.
class.sizes	Default: rep(200, k).
n	Default: sum(class.sizes).
scommon	Default: 10.
sindiv	Default: rep(10, k).
p	Default: 2*(sum(sindiv) + scommon).
beta.common	Default: 3*(1:k).
beta.indiv	Default: rep(3, k).
intercepts	Default: rep(0, k).
sigma	Default: 20.

Value

A list for data with 5 groups and a gaussian outcome, n=1000 and p=120:

x	Simulated features, size n x p.
y	Outcomes y, length n.
groups	Vector of length n, indicating which observations belong to which group.
snr	Gaussian outcome only: signal to noise ratio.
mu	Gaussian outcome only: the value of y before noise is added.

Author(s)

Erin Craig and Rob Tibshirani
 Maintainer: Erin Craig <ercr@umich.edu>

See Also

cv.ptLasso, ptLasso.

Examples

```

out = gaussian.example.data()
x = out$x; y=out$y; groups = out$group

```

```
get.individual.support
```

Get the support for individual models

Description

Get the indices of nonzero coefficients from the individual models in a fitted ptLasso or cv.ptLasso object, excluding the intercept.

Usage

```
get.individual.support(
  fit,
  s = "lambda.min",
  gamma = "gamma.min",
  commonOnly = FALSE,
  groups = 1:length(fit$fitind)
)
```

Arguments

fit	fitted "ptLasso" or "cv.ptLasso" object.
s	the choice of lambda to use. May be "lambda.min", "lambda.1se" or a numeric value. Default is "lambda.min".
gamma	for use only when 'relax = TRUE' was specified during training. The choice of 'gamma' to use. May be "gamma.min" or "gamma.1se". Default is "gamma.min".
commonOnly	whether to return the features that are chosen by more than half of the group- or response-specific models (TRUE) or the features that are chosen by any of the group-specific models (FALSE). Default is FALSE.
groups	which groups or responses to include when computing the support. Default is to include all groups/responses.

Value

This returns a vector containing the indices of nonzero coefficients (excluding the intercept).

Author(s)

Erin Craig and Rob Tibshirani
 Maintainer: Erin Craig <ercr@umich.edu>

See Also

ptLasso, cv.ptLasso.

Examples

```
# Train data
set.seed(1234)
out = gaussian.example.data()
x = out$x; y=out$y; groups = out$group;

fit = ptLasso(x, y, groups = groups, family = "gaussian", type.measure = "mse")

get.individual.support(fit)

# only return features common to all groups
get.individual.support(fit, commonOnly = TRUE)

# group 1 only
get.individual.support(fit, groups = 1)

cvfit = cv.ptLasso(x, y, groups = groups, alphalist = c(0, .5, 1),
                  family = "gaussian", type.measure = "mse")

get.individual.support(cvfit)

# group 1 only
get.individual.support(cvfit, groups = 1)
```

get.overall.support	<i>Get the support for the overall model</i>
---------------------	--

Description

Get the indices of nonzero coefficients from the overall model in a fitted ptLasso or cv.ptLasso object, excluding the intercept.

Usage

```
get.overall.support(fit, s = "lambda.min", gamma = "gamma.min")
```

Arguments

fit	fitted "ptLasso" or "cv.ptLasso" object.
s	the choice of lambda to use. May be "lambda.min", "lambda.1se" or a numeric value. Default is "lambda.min".
gamma	for use only when 'relax = TRUE' was specified during training. The choice of 'gamma' to use. May be "gamma.min" or "gamma.1se". Default is "gamma.min".

Value

This returns a vector containing the indices of nonzero coefficients (excluding the intercept).

Author(s)

Erin Craig and Rob Tibshirani
Maintainer: Erin Craig <ercr@umich.edu>

See Also

ptLasso, cv.ptLasso.

Examples

```
# Train data
set.seed(1234)
out = gaussian.example.data(k=2, class.sizes = c(50, 50))
x = out$x; y=out$y; groups = out$group;

fit = ptLasso(x, y, groups = groups, family = "gaussian", type.measure = "mse")

get.overall.support(fit, s="lambda.min")
get.overall.support(fit, s="lambda.1se")

cvfit = cv.ptLasso(x, y, groups = groups, family = "gaussian", type.measure = "mse")

get.overall.support(cvfit, s="lambda.min")
get.overall.support(cvfit, s="lambda.1se")
```

get.pretrain.support *Get the support for pretrained models*

Description

Get the indices of nonzero coefficients from the pretrained models in a fitted ptLasso or cv.ptLasso object, excluding the intercept.

Usage

```
get.pretrain.support(
  fit,
  s = "lambda.min",
  gamma = "gamma.min",
  commonOnly = FALSE,
  includeOverall = TRUE,
  groups = 1:length(fit$fitind)
)
```

Arguments

<code>fit</code>	fitted "ptLasso" or "cv.ptLasso" object.
<code>s</code>	the choice of lambda to use. May be "lambda.min", "lambda.1se" or a numeric value. Default is "lambda.min".
<code>gamma</code>	for use only when 'relax = TRUE' was specified during training. The choice of 'gamma' to use. May be "gamma.min" or "gamma.1se". Default is "gamma.min".
<code>commonOnly</code>	whether to return the features that are chosen by more than half of the group- or response-specific models (TRUE) or the features that are chosen by any of the group-specific models (FALSE). Default is FALSE.
<code>includeOverall</code>	whether to return the features that are chosen by the overall model and not the group-specific models (TRUE) or the features that are chosen by the overall model or the group-specific models (FALSE). Default is TRUE. Not used when 'use.case = "timeSeries"'.
<code>groups</code>	which groups or responses to include when computing the support. Default is to include all groups/responses.

Value

If a ptLasso object is supplied, this returns a vector containing the indices of nonzero coefficients (excluding the intercept). If a cv.ptLasso object is supplied, this returns a list of results - one for each value of alpha.

See Also

ptLasso, cv.ptLasso.

Examples

```
# Train data
set.seed(1234)
out = gaussian.example.data(k=2, class.sizes = c(50, 50))
x = out$x; y=out$y; groups = out$group;

fit = ptLasso(x, y, groups = groups, family = "gaussian", type.measure = "mse")

get.pretrain.support(fit)

# only return features common to all groups
get.pretrain.support(fit, commonOnly = TRUE)

# group 1 only, don't include the overall model support
get.pretrain.support(fit, groups = 1, includeOverall = FALSE)

# group 1 only, include the overall model support
get.pretrain.support(fit, groups = 1, includeOverall = TRUE)

cvfit = cv.ptLasso(x, y, groups = groups, family = "gaussian", type.measure = "mse")

get.pretrain.support(cvfit)
```

```
get.pretrain.support(cvfit, groups = 1)
```

makedata

Simulate input grouped data for testing with ptLasso.

Description

Simulate input grouped data for testing with ptLasso.

Usage

```
makedata(
  n,
  p,
  k,
  scommon,
  sindiv,
  class.sizes,
  beta.common,
  beta.indiv,
  intercepts = rep(0, k),
  sigma = 0,
  outcome = c("gaussian", "binomial", "multinomial"),
  mult.classes = 3
)
```

Arguments

n	Total number of observations to simulate.
p	Total number of features to simulate.
k	Number of groups.
scommon	Number of features shared by all groups.
sindiv	Vector of length k. The i^{th} entry indicates the number of features specific to group i .
class.sizes	Vector of length k. The i^{th} entry indicates the number of observations in group i .
beta.common	The coefficients for the common features. This can be a vector of length k, in which case, the i^{th} entry is the coefficient for all scommon features for group i . This can alternatively be a list of length k (one for each group). Each entry of this list should be a vector of length scommon, containing the coefficients for the scommon features.
beta.indiv	The coefficients for the individual features, in the same form as beta.common.
intercepts	A vector of length k, indicating the intercept for each group. Default is 0.

<code>sigma</code>	Only used for the Gaussian outcome. Should be a number greater than or equal to 0, used to modify the amount of noise added. Default is 0.
<code>outcome</code>	May be <code>"gaussian"</code> , <code>"binomial"</code> or <code>"multinomial"</code> .
<code>mult.classes</code>	Number of classes to simulate for the multinomial setting.

Value

A list:

<code>x</code>	Simulated features, size $n \times p$.
<code>y</code>	Outcomes y , length n .
<code>groups</code>	Vector of length n , indicating which observations belong to which group.
<code>snr</code>	Gaussian outcome only: signal to noise ratio.
<code>mu</code>	Gaussian outcome only: the value of y before noise is added.

Author(s)

Erin Craig and Rob Tibshirani
 Maintainer: Erin Craig <ercr@umich.edu>

See Also

`cv.ptLasso`, `ptLasso`.

Examples

```
# Data with a binary outcome:
k = 3
class.sizes = rep(100, k)
n = sum(class.sizes)
scommon = 5
sindiv = rep(5, k)
p = 2*(sum(sindiv) + scommon)
beta.common = lapply(1:k, function(i) c(-.5, .5, .3, -.9, .1))
beta.indiv = lapply(1:k, function(i) 0.9 * beta.common[[i]])

out = makedata(n=n, p=p, k=k, scommon=scommon, sindiv=sindiv,
               beta.common=beta.common, beta.indiv=beta.indiv,
               class.sizes=class.sizes, outcome="binomial")
x = out$x; y=out$y; groups = out$group
```

makedata.targetgroups *Simulate target grouped data for testing with ptLasso.*

Description

Simulate target grouped data for testing with ptLasso.

Usage

```
makedata.targetgroups(
  n,
  p,
  scommon,
  sindiv,
  class.sizes,
  shift.common,
  shift.indiv
)
```

Arguments

n	Total number of observations to simulate.
p	Total number of features to simulate.
scommon	Number of features shared by all groups.
sindiv	Vector of length k. The i^{th} entry indicates the number of features specific to group i .
class.sizes	Vector of length k. The i^{th} entry indicates the number of observations in group i .
shift.common	A list of length k (one for each group). Each entry of this list should be a vector of length scommon, containing the shifts for the scommon features. The i^{th} entry of this list will be added to the first scommon columns of x for observations in group i .
shift.indiv	The shifts for the individual features, in the same form as shift.common.

Value

A list:

x	Simulated features, size $n \times p$.
y	Outcomes y, length n.

Author(s)

Erin Craig and Rob Tibshirani
 Maintainer: Erin Craig <ercr@umich.edu>

See Also

cv.ptLasso, ptLasso.

Examples

```
k = 5
class.sizes = rep(50, k)
n = sum(class.sizes)
scommon = 3
sindiv = rep(3, k)
p = 3*(sum(sindiv) + scommon)
shift.common = lapply(seq(-.1, .1, length.out = k), function(i) rep(i, scommon))
shift.indiv = lapply(1:k, function(i) -shift.common[[i]])

out = makedata.targetgroups(n=n, p=p, scommon=scommon,
                           sindiv=sindiv, class.sizes=class.sizes,
                           shift.common=shift.common, shift.indiv=shift.indiv)

x = out$x; y=out$y
```

plot.cv.ptLasso	<i>Plot the cross-validation curve produced by cv.ptLasso, as a function of the alpha values used.</i>
-----------------	--

Description

A plot is produced, and the plot object is returned invisibly.

Usage

```
## S3 method for class 'cv.ptLasso'
plot(x, plot.alphahat = FALSE, ...)
```

Arguments

x	Fitted "cv.ptLasso" object.
plot.alphahat	If TRUE, show a dashed vertical line indicating the single value of alpha that maximized overall cross-validated performance.
...	Other graphical parameters to plot.

Value

Invisibly returns a gtable object (from `gridExtra::grid.arrange`) containing the combined plot. Called mainly for its side effect of drawing the plot.

See Also

ptLasso, cv.ptLasso and predict.cv.ptLasso.

Examples

```

set.seed(1234)
out = gaussian.example.data(k=2, class.sizes = c(50, 50))
x = out$x; y=out$y; groups = out$group

cvfit = cv.ptLasso(x, y, groups = groups, family = "gaussian", type.measure = "mse")
plot(cvfit)

```

plot.ptLasso

Plot the models trained by a ptLasso object

Description

A plot is produced, and nothing is returned.

Usage

```

## S3 method for class 'ptLasso'
plot(x, ...)

```

Arguments

x	Fitted "ptLasso" object.
...	Other parameters to pass to the plot function.

Value

No return value (returns NULL invisibly); called for its side effect of drawing the plot.

Author(s)

Erin Craig and Rob Tibshirani
Maintainer: Erin Craig <ercr@umich.edu>

See Also

ptLasso, cv.ptLasso and predict.cv.ptLasso.

Examples

```

set.seed(1234)
out = gaussian.example.data()
x = out$x; y=out$y; groups = out$group

fit = ptLasso(x, y, groups = groups, alpha = 0.5, family = "gaussian", type.measure = "mse")
plot(fit)

```

predict.cv.ptLasso	<i>Predict using a cv.ptLasso object.</i>
--------------------	---

Description

Return predictions and performance measures for a test set.

Usage

```
## S3 method for class 'cv.ptLasso'
predict(
  object,
  xtest,
  groupstest = NULL,
  ytest = NULL,
  alpha = NULL,
  alphatype = c("fixed", "varying"),
  type = c("link", "response", "class"),
  s = "lambda.min",
  gamma = "gamma.min",
  return.link = FALSE,
  ...
)
```

Arguments

object	Fitted "cv.ptLasso" object.
xtest	Input matrix, matching the form used by "cv.ptLasso" for model training.
groupstest	A vector indicating to which group each observation belongs. Coding should match that used for model training. Will be NULL for target grouped data.
ytest	Response variable. Optional. If included, "predict" will compute performance measures for xtest using "type.measure" from the cvfit object.
alpha	The chosen alpha to use for prediction. May be a vector containing one value of alpha for each group. If NULL, this will rely on the choice of "alphatype".
alphatype	Choice of "fixed" or "varying". If "fixed", use the alpha that achieved best cross-validated performance. If "varying", each group uses the alpha that optimized the group-specific cross-validated performance.
type	Type of prediction required. Type "link" gives the linear predictors for "binomial", "multinomial" or "cox" models; for "gaussian" models it gives the fitted values. Type "response" gives the fitted probabilities for "binomial" or "multinomial", and the fitted relative-risk for "cox"; for "gaussian" type "response" is equivalent to type "link". Note that for "binomial" models, results are returned only for the class corresponding to the second level of the factor response. Type "class" applies only to "binomial" or "multinomial" models, and produces the class label corresponding to the maximum probability.

s	Value of the penalty parameter 'lambda' at which predictions are required. Will use the same lambda for all models; can be a numeric value, "lambda.min" or "lambda.1se". Default is "lambda.min".
gamma	For use only when 'relax = TRUE' was specified during training. Value of the penalty parameter 'gamma' at which predictions are required. Will use the same gamma for all models; can be a numeric value, "gamma.min" or "gamma.1se". Default is "gamma.min".
return.link	If TRUE, will additionally return the linear link for the overall, pretrained and individual models: linkoverall, linkpre and linkind.
...	other arguments to be passed to the "predict" function.

Value

A list containing the requested predictions. If ytest is included, will also return error measures.

call	The call that produced this object.
alpha	The value(s) of alpha used to generate predictions.
yhatoverall	Predictions from the overall model.
yhatind	Predictions from the individual models.
yhatpre	Predictions from the pretrained models.
supoverall	Indices of the features selected by the overall model.
supind	Union of the indices of the features selected by the individual models.
suppre.common	Features selected in the first stage of pretraining.
suppre.individual	Union of the indices of the features selected by the pretrained models, without the features selected in the first stage.
type.measure	If ytest is supplied, the performance measure computed.
erroverall	If ytest is supplied, performance for the overall model. This is a named vector containing performance for (1) the entire dataset, (2) the average performance across groups, (3) the average performance across groups weighted by group size and (4) group-specific performance.
errind	If ytest is supplied, performance for the overall model. As described in erroverall.
errpre	If ytest is supplied, performance for the overall model. As described in erroverall.
linkoverall	If return.link is TRUE, return the linear link from the overall model.
linkind	If return.link is TRUE, return the linear link from the individual models.
linkpre	If return.link is TRUE, return the linear link from the pretrained models.

Author(s)

Erin Craig and Rob Tibshirani
 Maintainer: Erin Craig <ercr@umich.edu>

See Also

ptLasso, cv.ptLasso and predict.cv.ptLasso.

Examples

```
#### Gaussian example
set.seed(1234)
out = gaussian.example.data(k=2, class.sizes = c(50, 50))
x = out$x; y=out$y; groups = out$group;
outtest = gaussian.example.data(k=2, class.sizes = c(50, 50))
xtest=outtest$x; ytest=outtest$y; groupstest=outtest$groups;

# Model fitting
# By default, use the single value of alpha that had the best CV performance on the entire dataset:
cvfit = cv.ptLasso(x, y, groups = groups, family = "gaussian", type.measure = "mse")
pred = predict(cvfit, xtest, groupstest, ytest=ytest, s="lambda.min")
pred

# For each group, use the value of alpha that had the best CV performance for that group:
pred = predict(cvfit, xtest, groupstest, ytest=ytest, s="lambda.min", alphas = "varying")
pred

# Specify a single value of alpha and use lambda.1se.
pred = predict(cvfit, xtest, groupstest, ytest=ytest, s="lambda.1se",
               alphas = "varying", alpha = .3)
pred

# Specify a vector of choices for alpha:
pred = predict(cvfit, xtest, groupstest, ytest=ytest, s="lambda.min",
               alphas = "varying", alpha = c(.1, .5))
pred
```

predict.ptLasso

Predict using a ptLasso object.

Description

Return predictions and performance measures for a test set.

Usage

```
## S3 method for class 'ptLasso'
predict(
  object,
  xtest,
  groupstest = NULL,
  ytest = NULL,
  type = c("link", "response", "class"),
  s = "lambda.min",
  gamma = "gamma.min",
  return.link = FALSE,
  ...
)
```

Arguments

<code>object</code>	Fitted "ptLasso" object.
<code>xtest</code>	Input matrix, matching the form used by "ptLasso" for model training.
<code>groupstest</code>	A vector indicating to which group each observation belongs. Coding should match that used for model training. Will be NULL for target grouped data.
<code>ytest</code>	Response variable. Optional. If included, "predict" will compute performance measures for xtest using "type.measure" from the cvfit object.
<code>type</code>	Type of prediction required. Type "link" gives the linear predictors for "binomial", "multinomial" or "cox" models; for "gaussian" models it gives the fitted values. Type "response" gives the fitted probabilities for "binomial" or "multinomial", and the fitted relative-risk for "cox"; for "gaussian" type "response" is equivalent to type "link". Note that for "binomial" models, results are returned only for the class corresponding to the second level of the factor response. Type "class" applies only to "binomial" or "multinomial" models, and produces the class label corresponding to the maximum probability.
<code>s</code>	Value of the penalty parameter 'lambda' at which predictions are required. Will use the same lambda for all models; can be a numeric value, "lambda.min" or "lambda.1se". Default is "lambda.min".
<code>gamma</code>	For use only when 'relax = TRUE' was specified during training. Value of the penalty parameter 'gamma' at which predictions are required. Will use the same gamma for all models; can be a numeric value, "gamma.min" or "gamma.1se". Default is "gamma.min".
<code>return.link</code>	If TRUE, will additionally return the linear link for the overall, pretrained and individual models: linkoverall, linkpre and linkind.
<code>...</code>	other arguments to be passed to the "predict" function.

Value

A list containing the requested predictions. If ytest is included, will also return error measures.

<code>call</code>	The call that produced this object.
<code>alpha</code>	The value(s) of alpha used to generate predictions. Will be the same alpha used to in model training.
<code>yhatoverall</code>	Predictions from the overall model.
<code>yhatind</code>	Predictions from the individual models.
<code>yhatpre</code>	Predictions from the pretrained models.
<code>supoverall</code>	Indices of the features selected by the overall model.
<code>supind</code>	Union of the indices of the features selected by the individual models.
<code>suppre.common</code>	Features selected in the first stage of pretraining.
<code>suppre.individual</code>	Union of the indices of the features selected by the pretrained models, without the features selected in the first stage.
<code>type.measure</code>	If ytest is supplied, the string name of the computed performance measure.

erroverall	If ytest is supplied, performance for the overall model. This is a named vector containing performance for (1) the entire dataset, (2) the average performance across groups, (3) the average performance across groups weighted by group size and (4) group-specific performance.
errind	If ytest is supplied, performance for the overall model. As described in erroverall.
errpre	If ytest is supplied, performance for the overall model. As described in erroverall.
linkoverall	If return.link is TRUE, return the linear link from the overall model.
linkind	If return.link is TRUE, return the linear link from the individual models.
linkpre	If return.link is TRUE, return the linear link from the pretrained models.

Examples

```
# Gaussian example
set.seed(1234)
out = gaussian.example.data()
x = out$x; y=out$y; groups = out$group

outtest = gaussian.example.data()
xtest=outtest$x; ytest=outtest$y; groupstest=outtest$groups

fit = ptLasso(x, y, groups = groups, alpha = 0.5, family = "gaussian", type.measure = "mse")
pred = predict(fit, xtest, groupstest, ytest=ytest)
pred
```

print.cv.ptLasso	<i>Print the cv.ptLasso object.</i>
------------------	-------------------------------------

Description

Print the cv.ptLasso object.

Usage

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

Arguments

x	fitted "cv.ptLasso" object.
...	other arguments to pass to the print function.

Value

Invisibly returns x. Called for its side effect of printing a summary to the console.

Author(s)

Erin Craig and Rob Tibshirani
Maintainer: Erin Craig <ercr@umich.edu>

See Also

ptLasso, cv.ptLasso and predict.cv.ptLasso.

Examples

```
out = gaussian.example.data(k=2, class.sizes = c(50, 50))
x = out$x; y=out$y; groups = out$group;

cvfit = cv.ptLasso(x, y, groups = groups, family = "gaussian", type.measure = "mse")
print(cvfit)
```

```
print.predict.cv.ptLasso
```

Print the predict.cv.ptLasso object.

Description

Print the predict.cv.ptLasso object.

Usage

```
## S3 method for class 'predict.cv.ptLasso'
print(x, ...)
```

Arguments

x	output of predict called with a ptLasso object.
...	other arguments to pass to the print function.

Value

Invisibly returns x. Called for its side effect of printing a summary to the console.

Author(s)

Erin Craig and Rob Tibshirani
Maintainer: Erin Craig <ercr@umich.edu>

See Also

cv.ptLasso and predict.cv.ptLasso.

Examples

```
# Train data
out = gaussian.example.data(k=2, class.sizes = c(50, 50))
x = out$x; y=out$y; groups = out$group;

# Test data
outtest = gaussian.example.data(k=2, class.sizes = c(50, 50))
xtest=outtest$x; ytest=outtest$y; groupstest=outtest$groups

cvfit = cv.ptLasso(x, y, groups = groups, nfolds = 3, family = "gaussian", type.measure = "mse")
pred = predict(cvfit, xtest, groupstest, ytest=ytest, s="lambda.min")
print(pred)

# If ytest is not supplied, just prints the pretrained predictions.
pred = predict(cvfit, xtest, groupstest, s="lambda.min")
print(pred)
```

```
print.predict.ptLasso Print the predict.ptLasso object.
```

Description

Print the predict.ptLasso object.

Usage

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

Arguments

x	output of predict called with a ptLasso object.
...	other arguments to pass to the print function.

Value

Invisibly returns x. Called for its side effect of printing a summary to the console.

Author(s)

Erin Craig and Rob Tibshirani
 Maintainer: Erin Craig <ererc@umich.edu>

See Also

ptLasso and predict.ptLasso.

Examples

```
# Train data
out = gaussian.example.data(k=2, class.sizes = c(50, 50))
x = out$x; y=out$y; groups = out$group;

# Test data
outtest = gaussian.example.data(k=2, class.sizes = c(50, 50))
xtest=outtest$x; ytest=outtest$y; groupstest=outtest$groups

fit = ptLasso(x, y, groups = groups, nfolds = 3, family = "gaussian", type.measure = "mse")
pred = predict(fit, xtest, groupstest, ytest=ytest, s="lambda.min")
print(pred)

# If ytest is not supplied, just prints the pretrained predictions.
pred = predict(fit, xtest, groupstest, s="lambda.min")
print(pred)
```

print.ptLasso	<i>Print the ptLasso object.</i>
---------------	----------------------------------

Description

Print the ptLasso object.

Usage

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

Arguments

x	fitted "ptLasso" object.
...	other arguments to pass to the print function.

Value

Invisibly returns x. Called for its side effect of printing a summary to the console.

Author(s)

Erin Craig and Rob Tibshirani
 Maintainer: Erin Craig <ererc@umich.edu>

See Also

ptLasso, cv.ptLasso and predict.cv.ptLasso.

Examples

```
out = gaussian.example.data()
x = out$x; y=out$y; groups = out$group;

fit = ptLasso(x, y, groups = groups, family = "gaussian", type.measure = "mse")
print(fit)
```

ptLasso

Fit a pretrained lasso model using glmnet.

Description

Fits a pretrained lasso model using the glmnet package, for a fixed choice of the pretraining hyperparameter alpha. Additionally fits an "overall" model (using all data) and "individual" models (use each individual group). Can fit input-grouped data with Gaussian, multinomial, binomial or Cox outcomes, and target-grouped data, which necessarily has a multinomial outcome. Many ptLasso arguments are passed directly to glmnet, and therefore the glmnet documentation is another good reference for ptLasso.

Usage

```
ptLasso(
  x,
  y,
  groups = NULL,
  alpha = 0.5,
  family = c("default", "gaussian", "multinomial", "binomial", "cox"),
  type.measure = c("default", "mse", "mae", "auc", "deviance", "class", "C"),
  use.case = c("inputGroups", "targetGroups", "multiresponse", "timeSeries"),
  overall.lambda = c("lambda.1se", "lambda.min"),
  overall.gamma = "gamma.1se",
  foldid = NULL,
  nfolds = 10,
  standardize = TRUE,
  verbose = FALSE,
  weights = NULL,
  penalty.factor = rep(1, p),
  fitoverall = NULL,
  fitind = NULL,
  en.alpha = 1,
  group.intercepts = TRUE,
  ...
)
```

Arguments

<code>x</code>	input matrix, of dimension <code>nobs x nvars</code> ; each row is an observation vector. Can be in sparse matrix format (inherit from class <code>"sparseMatrix"</code> as in package <code>'Matrix'</code>). Requirement: <code>'nvars > 1'</code> ; in other words, <code>'x'</code> should have 2 or more columns. If <code>'use.case = "timeSeries"</code> , then <code>x</code> may be a list of matrices with identical dimensions, one for each point in time.
<code>y</code>	response variable. Quantitative for <code>'family="gaussian"</code> . For <code>'family="binomial"</code> should be either a factor with two levels, or a two-column matrix of counts or proportions (the second column is treated as the target class; for a factor, the last level in alphabetical order is the target class). For <code>'family="multinomial"</code> , can be a <code>'nc >= 2'</code> level factor, or a matrix with <code>'nc'</code> columns of counts or proportions. For either <code>"binomial"</code> or <code>"multinomial"</code> , if <code>'y'</code> is presented as a vector, it will be coerced into a factor. For <code>'family="cox"</code> , preferably a <code>'Surv'</code> object from the survival package: see Detail section for more information. For <code>'use.case = "multiresponse"</code> or <code>'use.case = "timeSeries"</code> , <code>'y'</code> is a matrix of responses.
<code>groups</code>	A vector of length <code>nobs</code> indicating to which group each observation belongs. Only for <code>'use.case = "inputGroups"</code> .
<code>alpha</code>	The pretrained lasso hyperparameter, with $0 \leq \alpha \leq 1$. The range of <code>alpha</code> is from 0 (which fits the overall model with fine tuning) to 1 (the individual models). The default value is 0.5, chosen mostly at random. To choose the appropriate value for your data, please either run <code>ptLasso</code> with a few choices of <code>alpha</code> and evaluate with a validation set, or use <code>cv.ptLasso</code> , which recommends a value of <code>alpha</code> using cross validation.
<code>family</code>	Either a character string representing one of the built-in families, or else a <code>'glm()'</code> family object. For more information, see Details section below or the documentation for response type (see above).
<code>type.measure</code>	loss to use for cross-validation within each individual, overall, or pretrained lasso model. Currently five options, not all available for all models. The default is <code>'type.measure="deviance"</code> , which uses squared-error for gaussian models (a.k.a <code>'type.measure="mse"</code> there), deviance for logistic and poisson regression, and partial-likelihood for the Cox model. <code>'type.measure="class"</code> applies to binomial and multinomial logistic regression only, and gives misclassification error. <code>'type.measure="auc"</code> is for two-class logistic regression only, and gives area under the ROC curve. <code>'type.measure="mse"</code> or <code>'type.measure="mae"</code> (mean absolute error) can be used by all models except the <code>"cox"</code> ; they measure the deviation from the fitted mean to the response. <code>'type.measure="C"</code> is Harrel's concordance measure, only available for <code>'cox'</code> models.
<code>use.case</code>	The type of grouping observed in the data. Can be one of <code>"inputGroups"</code> , <code>"targetGroups"</code> , <code>"multiresponse"</code> or <code>"timeSeries"</code> .
<code>overall.lambda</code>	The choice of <code>lambda</code> to be used by the overall model to define the offset and penalty factor for pretrained lasso. Defaults to <code>"lambda.lse"</code> , could alternatively be <code>"lambda.min"</code> . This choice of <code>lambda</code> will be used to compute the offset and penalty factor (1) during model training and (2) during prediction. In the <code>predict</code> function, another <code>lambda</code> must be specified for the individual models, the second stage of pretraining and the overall model.

<code>overall.gamma</code>	For use only when the option <code>relax = TRUE</code> is specified. The choice of gamma to be used by the overall model to define the offset and penalty factor for pretrained lasso. Defaults to "gamma.lse", but "gamma.min" is also a good option. This choice of gamma will be used to compute the offset and penalty factor (1) during model training and (2) during prediction. In the predict function, another gamma must be specified for the individual models, the second stage of pretraining and the overall model.
<code>foldid</code>	An optional vector of values between 1 and <code>nfolds</code> identifying what fold each observation is in. If supplied, <code>nfold</code> can be missing.
<code>nfolds</code>	Number of folds for CV (default is 10). Although <code>nfolds</code> can be as large as the sample size (leave-one-out CV), it is not recommended for large datasets. Smallest value allowable is <code>nfolds = 3</code> .
<code>standardize</code>	Should the predictors be standardized before fitting (default is <code>TRUE</code>).
<code>verbose</code>	If <code>verbose=TRUE</code> , print a statement showing which model is currently being fit with <code>cv.glmnet</code> .
<code>weights</code>	observation weights. Default is 1 for each observation.
<code>penalty.factor</code>	Separate penalty factors can be applied to each coefficient. This is a number that multiplies 'lambda' to allow differential shrinkage. Can be 0 for some variables, which implies no shrinkage, and that variable is always included in the model. Default is 1 for all variables (and implicitly infinity for variables listed in 'exclude'). For more information, see <code>?glmnet</code> . For pretraining, the user-supplied <code>penalty.factor</code> will be multiplied by that computed by the overall model.
<code>fitoverall</code>	An optional <code>cv.glmnet</code> object specifying the overall model. This should have been trained on the full training data, with the argument 'keep = <code>TRUE</code> '.
<code>fitind</code>	An optional list of <code>cv.glmnet</code> objects specifying the individual models. These should have been trained on the original training data, with the argument 'keep = <code>TRUE</code> '.
<code>en.alpha</code>	The elasticnet mixing parameter, with $0 \leq \text{en.alpha} \leq 1$. The penalty is defined as $(1-\alpha)/2\ \beta\ _2^2 + \alpha\ \beta\ _1$. 'alpha=1' is the lasso penalty, and 'alpha=0' the ridge penalty. Default is 'en.alpha = 1' (lasso).
<code>group.intercepts</code>	For 'use.case = "inputGroups"' only. If 'TRUE', fit the overall model with a separate intercept for each group. If 'FALSE', ignore the grouping and fit one overall intercept. Default is 'TRUE'.
<code>...</code>	Additional arguments to be passed to the <code>cv.glmnet</code> functions. Notable choices include "trace.it" and "parallel". If <code>trace.it = TRUE</code> , then a progress bar is displayed for each call to "cv.glmnet"; useful for big models that take a long time to fit. If <code>parallel = TRUE</code> , use parallel foreach to fit each fold. Must register parallel before hand, such as <code>doMC</code> or others. ptLasso does not support the arguments <code>intercept</code> , <code>offset</code> , <code>fit</code> and <code>check.args</code> .

Value

An object of class "ptLasso", which is a list with the ingredients of the fitted models.

`call` The call that produced this object.

k	The number of groups (for 'use.case = "inputGroups")).
nresps	The number of responses (for 'use.case = "multiresponse" and 'use.case = "timeseries").
alpha	The value of alpha used for pretraining.
group.levels	IDs for all of the groups used in training (for 'use.case = "inputGroups").
group.legend	Mapping from user-supplied group ids to numeric group ids. For internal use (e.g. with predict). (For 'use.case = "inputGroups")
fitoverall	A fitted cv.glmnet object trained using the full data. Not available for 'use.case = "timeseries".
fitpre	A list of fitted (pretrained) cv.glmnet objects, one trained with each data group or response.
fitind	A list of fitted cv.glmnet objects, one trained with each group or response.
fitoverall.lambda	Lambda used with fitoverall, to compute the offset for pretraining.
fitoverall.gamma	Gamma used with fitoverall when 'relax = TRUE', to compute the offset for pretraining.

References

Friedman, J., Hastie, T., & Tibshirani, R. (2010). Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, 33(1), 1-22.

See Also

[glmnet](#)

Examples

```
# Getting started. First, we simulate data: we need covariates x, response y and group IDs.
set.seed(1234)
n=100
p=10
n.groups=2
x = matrix(rnorm(n*p), n, p)
y = rnorm(n)
groups = sort(rep(1:n.groups, n/n.groups))

xtest = matrix(rnorm(n*p), n, p)
ytest = rnorm(n)
groupstest = sort(rep(1:n.groups, n/n.groups))

# Now, we can fit a ptLasso model:
fit = ptLasso(x, y, groups = groups, alpha = 0.5, family = "gaussian",
             nfolds = 3, type.measure = "mse")
plot(fit) # to see all of the cv.glmnet models trained
predict(fit, xtest, groupstest) # to predict on new data
predict(fit, xtest, groupstest, ytest=ytest) # if ytest is included, we also measure performance
```

[illegible]

```

n=n, p=p,
beta.common=beta.common, beta.indiv=beta.indiv,
intercepts=intercepts, sigma=20)
xtest=outtest$x; ytest=outtest$y; groupstest=outtest$groups

fit = ptLasso(x, y, groups = groups, alpha = 0.5, family = "gaussian", type.measure = "mse")
plot(fit) # to see all of the cv.glmnet models trained
predict(fit, xtest, groupstest, ytest=ytest)

# Now, we repeat with a binomial outcome.
# This example has k = 3 groups, where each group has 100 observations.
# There are scommon = 5 features shared across all groups, and
# sindiv = 5 features unique to each group.
# n = 300 and p = 40 (20 informative features and 20 noise features).
# The coefficients of the common features differ across groups (beta.common),
# as do the coefficients specific to each group (beta.indiv).
set.seed(1234)
k=3
class.sizes=rep(100, k)
scommon=5; sindiv=rep(5, k)
n=sum(class.sizes); p=2*(sum(sindiv) + scommon)
beta.common=list(c(-.5, .5, .3, -.9, .1), c(-.3, .9, .1, -.1, .2), c(0.1, .2, -.1, .2, .3))
beta.indiv = lapply(1:k, function(i) 0.9 * beta.common[[i]])

out = binomial.example.data(k=k, class.sizes=class.sizes,
                             scommon=scommon, sindiv=sindiv,
                             n=n, p=p,
                             beta.common=beta.common, beta.indiv=beta.indiv)
x = out$x; y=out$y; groups = out$group

outtest = binomial.example.data(k=k, class.sizes=class.sizes,
                                 scommon=scommon, sindiv=sindiv,
                                 n=n, p=p,
                                 beta.common=beta.common, beta.indiv=beta.indiv)
xtest=outtest$x; ytest=outtest$y; groupstest=outtest$groups

fit = ptLasso(x, y, groups = groups, alpha = 0.5, family = "binomial", type.measure = "auc")
plot(fit) # to see all of the cv.glmnet models trained
predict(fit, xtest, groupstest, ytest=ytest)

## Not run:
### Model fitting with parallel = TRUE
require(doMC)
registerDoMC(cores = 4)
fit = ptLasso(x, y, groups = groups, family = "gaussian", type.measure = "mse", parallel=TRUE)

## End(Not run)

# Multiresponse pretraining:

```

```

# Now let's consider the case of a multiresponse outcome. We'll start by simulating data:
set.seed(1234)
n = 1000; ntrain = 500;
p = 500
sigma = 2

x = matrix(rnorm(n*p), n, p)
beta1 = c(rep(1, 5), rep(0.5, 5), rep(0, p - 10))
beta2 = c(rep(1, 5), rep(0, 5), rep(0.5, 5), rep(0, p - 15))

mu = cbind(x %*% beta1, x %*% beta2)
y = cbind(mu[, 1] + sigma * rnorm(n),
          mu[, 2] + sigma * rnorm(n))
cat("SNR for the two tasks:", round(diag(var(mu)/var(y-mu)), 2), fill=TRUE)
cat("Correlation between two tasks:", cor(y[, 1], y[, 2]), fill=TRUE)

xtest = x[-(1:ntrain), ]
ytest = y[-(1:ntrain), ]

x = x[1:ntrain, ]
y = y[1:ntrain, ]

fit = ptLasso(x, y, type.measure = "mse", use.case = "multiresponse")
plot(fit) # to see all of the cv.glmnet models trained
predict(fit, xtest) # to predict with new data
predict(fit, xtest, ytest=ytest) # if ytest is included, we also measure performance
# By default, we used lambda = "lambda.min" to measure performance.
# We could instead use lambda = "lambda.1se":
predict(fit, xtest, ytest=ytest, s="lambda.1se")

# We could also use the glmnet option relax = TRUE:
fit = ptLasso(x, y, type.measure = "mse", relax = TRUE, use.case = "multiresponse")

# Time series pretraining
# Now suppose we have time series data with a binomial outcome measured at 3 different time points.
set.seed(1234)
n = 600; ntrain = 300; p = 50
x = matrix(rnorm(n*p), n, p)

beta1 = c(rep(0.5, 10), rep(0, p-10))
beta2 = beta1 + c(rep(0, 10), runif(5, min = 0, max = 0.5), rep(0, p-15))
beta3 = beta1 + c(rep(0, 10), runif(5, min = 0, max = 0.5), rep(.5, 5), rep(0, p-20))

y1 = rbinom(n, 1, prob = 1/(1 + exp(-x %*% beta1)))
y2 = rbinom(n, 1, prob = 1/(1 + exp(-x %*% beta2)))
y3 = rbinom(n, 1, prob = 1/(1 + exp(-x %*% beta3)))
y = cbind(y1, y2, y3)

xtest = x[-(1:ntrain), ]
ytest = y[-(1:ntrain), ]

```

```
x = x[1:ntrain, ]
y = y[1:ntrain, ]

fit = ptLasso(x, y, use.case="timeSeries", family="binomial", type.measure = "auc")
plot(fit)
predict(fit, xtest, ytest=ytest)

# The glmnet option relax = TRUE:
fit = ptLasso(x, y, type.measure = "auc", family = "binomial", relax = TRUE,
              use.case = "timeSeries")
plot(fit)
predict(fit, xtest, ytest=ytest)
```

Index

* **classification**

- binomial.example.data, [2](#)
- coef.cv.ptLasso, [4](#)
- coef.ptLasso, [5](#)
- gaussian.example.data, [12](#)
- get.individual.support, [14](#)
- get.overall.support, [15](#)
- get.pretrain.support, [16](#)
- makedata, [18](#)
- makedata.targetgroups, [20](#)
- plot.cv.ptLasso, [21](#)
- plot.ptLasso, [22](#)
- predict.cv.ptLasso, [23](#)
- print.cv.ptLasso, [27](#)
- print.predict.cv.ptLasso, [28](#)
- print.predict.ptLasso, [29](#)
- print.ptLasso, [30](#)

* **models**

- binomial.example.data, [2](#)
- coef.cv.ptLasso, [4](#)
- coef.ptLasso, [5](#)
- gaussian.example.data, [12](#)
- get.individual.support, [14](#)
- get.overall.support, [15](#)
- get.pretrain.support, [16](#)
- makedata, [18](#)
- makedata.targetgroups, [20](#)
- plot.cv.ptLasso, [21](#)
- plot.ptLasso, [22](#)
- predict.cv.ptLasso, [23](#)
- print.cv.ptLasso, [27](#)
- print.predict.cv.ptLasso, [28](#)
- print.predict.ptLasso, [29](#)
- print.ptLasso, [30](#)

* **regression**

- binomial.example.data, [2](#)
- coef.cv.ptLasso, [4](#)
- coef.ptLasso, [5](#)
- gaussian.example.data, [12](#)

- get.individual.support, [14](#)
- get.overall.support, [15](#)
- get.pretrain.support, [16](#)
- makedata, [18](#)
- makedata.targetgroups, [20](#)
- plot.cv.ptLasso, [21](#)
- plot.ptLasso, [22](#)
- predict.cv.ptLasso, [23](#)
- print.cv.ptLasso, [27](#)
- print.predict.cv.ptLasso, [28](#)
- print.predict.ptLasso, [29](#)
- print.ptLasso, [30](#)

- binomial.example.data, [2](#)

- coef.cv.ptLasso, [4](#)
- coef.ptLasso, [5](#)
- cv.ptLasso, [6](#)

- gaussian.example.data, [12](#)
- get.individual.support, [14](#)
- get.overall.support, [15](#)
- get.pretrain.support, [16](#)
- glmnet, [34](#)

- makedata, [18](#)
- makedata.targetgroups, [20](#)

- plot.cv.ptLasso, [8](#), [21](#)
- plot.ptLasso, [22](#)
- predict.cv.ptLasso, [23](#)
- predict.ptLasso, [25](#)
- print.cv.ptLasso, [27](#)
- print.predict.cv.ptLasso, [28](#)
- print.predict.ptLasso, [29](#)
- print.ptLasso, [30](#)
- ptLasso, [8](#), [31](#)