Package 'grpnet'

January 24, 2025

Type Package

Title Group Elastic Net Regularized GLMs and GAMs

Version 0.7

Date 2025-01-23

Description Efficient algorithms for fitting generalized linear and additive models with group elastic net penalties as described in Helwig (2024) <doi:10.1080/10618600.2024.2362232>. Implements group LASSO, group MCP, and group SCAD with an optional group ridge penalty. Computes the regularization path for linear regression (gaussian), multivariate regression (multigaussian), logistic regression (binomial), multinomial logistic regression (multinomial), log-linear count regression (poisson and negative.binomial), and log-linear continuous regression (gamma and inverse gaussian). Supports default and formula methods for model specification, k-fold cross-validation for tuning the regularization parameters, and nonparametric regression via tensor product reproducing kernel (smoothing spline) basis function expansion.

License GPL (>= 2)

Encoding UTF-8

Depends R (>= 3.5.0)

NeedsCompilation yes

Author Nathaniel E. Helwig [aut, cre]

Maintainer Nathaniel E. Helwig <helwig@umn.edu>

Repository CRAN

Date/Publication 2025-01-24 10:20:09 UTC

Contents

`auto` *Auto MPG Data Set*

Description

Miles per gallon and other characteristics of vehicles from the 1970s-1980s. A version of this dataset was used as the 1983 American Statistical Association Exposition dataset.

Usage

```
data("auto")
```

Format

A data frame with 392 observations on the following 9 variables.

`mpg` miles per gallon (numeric vector)

`cylinders` number of cylinders: 3,4,5,6,8 (ordered factor)

`displacement` engine displacement in cubic inches (numeric vector)

`horsepower` engine horsepower (integer vector)

`weight` vehicle weight in of lbs. (integer vector)

`acceleration` 0-60 mph time in sec. (numeric vector)

`model.year` ranging from 1970 to 1982 (integer vector)

`origin` region of origin: American, European, Japanese (factor vector)

Details

This is a modified version of the "Auto MPG Data Set" on the UCI Machine Learning Repository, which is a modified version of the "cars" dataset on StatLib.

Compared to the version of the dataset in UCI's MLR, this version of the dataset has removed (i) the 6 rows with missing `horsepower` scores, and (ii) the last column giving the name of each vehicle (`car.name`).

Source

The dataset was originally collected by Ernesto Ramos and David Donoho.

StatLib—Datasets Archive at Carnegie Mellon University http://lib.stat.cmu.edu/datasets/cars.data

Machine Learning Repository at University of California Irvine https://archive.ics.uci.edu/ml/datasets/Auto+MPG

Examples

```
# load data
data(auto)

# display structure
str(auto)

# display header
head(auto)

# see 'cv.grpnet' for cross-validation examples
?cv.grpnet

# see 'grpnet' for fitting examples
?grpnet
```

---

coef *Extract Coefficients for cv.grpnet and grpnet Fits*

Description

Obtain coefficients from a cross-validated group elastic net regularized GLM (cv.grpnet) or a group elastic net regularized GLM (grpnet) object.

Usage

```
## S3 method for class 'cv.grpnet'
coef(object,
     s = c("lambda.1se", "lambda.min"),
     ...)

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

Arguments

`object`	Object of class "cv.grpnet" or "grpnet"
`s`	Lambda value(s) at which predictions should be obtained. For "cv.grpnet" objects, default uses the 1se solution. For "grpnet" objects, default uses `s = object$lambda`. Interpolation is used for `s` values that are not included in `object$lambda`.
`...`	Additional arguments (ignored)

Details

coef.cv.grpnet:
Returns the coefficients that are used by the `predict.cv.grpnet` function to form predictions from a fit `cv.grpnet` object.

coef.grpnet:
Returns the coefficients that are used by the `predict.grpnet` function to form predictions from a fit `grpnet` object.

Value

For multigaussian and multinomial response variables, returns a list of length `length(object$ylev)`, where the `j`-th element is a matrix of dimension `c(ncoef, length(s))` giving the coefficients for `object$ylev[j]`.

For other response variables, returns a matrix of dimension `c(ncoef, length(s))`, where the `i`-th column gives the coefficients for `s[i]`.

Note

The syntax of these functions closely mimics that of the `coef.cv.glmnet` and `coef.glmnet` functions in the **glmnet** package (Friedman, Hastie, & Tibshirani, 2010).

Author(s)

Nathaniel E. Helwig <helwig@umn.edu>

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. doi:10.18637/jss.v033.i01

Helwig, N. E. (2024). Versatile descent algorithms for group regularization and variable selection in generalized linear models. *Journal of Computational and Graphical Statistics*. doi:10.1080/10618600.2024.2362232

See Also

`print.coef.grpnet` for printing `coef.grpnet` objects

`predict.cv.grpnet` for predicting from `cv.grpnet` objects

`predict.grpnet` for predicting from `grpnet` objects

Examples

```
################   grpnet   ################

# load data
data(auto)

# fit model (formula method, response = mpg)
mod <- grpnet(mpg ~ ., data = auto)
```

```
# extract coefs for regularization path (output = 12 x 100 matrix)
coef(mod)

# extract coefs at 3 particular points (output = 12 x 3 matrix)
coef(mod, s = c(1.5, 1, 0.5))


##############   cv.grpnet   ##############

# load data
data(auto)

# 5-fold cv (formula method, response = mpg)
set.seed(1)
mod <- cv.grpnet(mpg ~ ., data = auto, nfolds = 5, alpha = 1)

# extract coefs for "min" solution (output = 12 x 1 matrix)
coef(mod)

# extract coefs for "1se" solution (output = 12 x 1 matrix)
coef(mod, s = "lambda.1se")

# extract coefs at 3 particular points (output = 12 x 3 matrix)
coef(mod, s = c(1.5, 1, 0.5))
```

`cv.compare` *Compare Multiple cv.grpnet Solutions*

Description

Creates a plot (default) or returns a data frame (otherwise) that compares the cross-validation error for multiple `cv.grpnet` fits.

Usage

```
cv.compare(x,
           s = c("lambda.1se", "lambda.min"),
           plot = TRUE,
           at = 1:length(x),
           nse = 1,
           point.col = "red",
           line.col = "gray",
           lwd = 2,
           bwd = 0.02,
           labels = NULL,
           xlim = NULL,
           ylim = NULL,
           xlab = NULL,
```

```
ylab = NULL,
...)
```

Arguments

x	a single cv.grpnet object or a list of cv.grpnet objects.
s	the tuning parameter value at which to plot results (if x is a list).
plot	switch controlling whether a plot is produced (default) versus data frame.
at	x-axis coordinates for plotting the cv error for each solution.
nse	number of standard errors to use for error bars in plot.
point.col	color for point used to plot the average of the cv error.
line.col	color for lines used to plot the standard error for the cv error.
lwd	width of lines used to plot the standard error for the cv error.
bwd	width of standard error bars in terms of proportion of range(x).
labels	labels for x-axis tick marks. Defaults to names(x).
xlim	axis limits for abscissa (x-axis)
ylim	axis limits for ordinate (y-axis)
xlab	axis label for abscissa (x-axis)
ylab	axis label for ordinate (y-axis)
...	additional arguments passed to plotting functions.

Details

Default behavior creates a plot that displays the mean cv error +/- 1 se for each of the requested solutions.

If the input x is a single cv.grpnet object, then the function plots the lambda.min and lambda.1se solutions.

If the input x is a list of cv.grpnet objects, then the function plots either the lambda.min or the lambda.1se solution (controlled by s argument) for all of the input models.

Value

When plot = TRUE, there is no return value (it produces a plot)

When plot = FALSE, a data.frame is returned with the mean cv error (and se) for each solution

Author(s)

Nathaniel E. Helwig <helwig@umn.edu>

References

Helwig, N. E. (2024). Versatile descent algorithms for group regularization and variable selection in generalized linear models. *Journal of Computational and Graphical Statistics*. doi:10.1080/10618600.2024.2362232

See Also

plot.cv.grpnet for plotting cv error path (for all lambdas)

plot.grpnet for plotting regularization path (for single lambda)

Examples

```
# load data
data(auto)

# LASSO penalty
set.seed(1)
mod1 <- cv.grpnet(mpg ~ ., data = auto, nfolds = 5, alpha = 1)

# MCP penalty
set.seed(1)
mod2 <- cv.grpnet(mpg ~ ., data = auto, nfolds = 5, alpha = 1, penaly = "MCP")

# SCAD penalty
set.seed(1)
mod3 <- cv.grpnet(mpg ~ ., data = auto, nfolds = 5, alpha = 1, penaly = "SCAD")

# compare lambda.min and lambda.1se for mod1
cv.compare(mod1)

# compare lambda.1se for mod1, mod2, mod3
cv.compare(x = list(mod1, mod2, mod3), labels = c("LASSO", "MCP", "SCAD"))
```

---

cv.grpnet *Cross-Validation for grpnet*

Description

Implements k-fold cross-validation for grpnet to find the regularization parameters that minimize the prediction error (deviance, mean squared error, mean absolute error, or misclassification rate).

Usage

```
cv.grpnet(x, ...)

## Default S3 method:
cv.grpnet(x,
          y,
          group,
          weights = NULL,
          offset = NULL,
          alpha = c(0.01, 0.25, 0.5, 0.75, 1),
          gamma = c(3, 4, 5),
          type.measure = NULL,
```

```
            nfolds = 10,
            foldid = NULL,
            same.lambda = FALSE,
            parallel = FALSE,
            cluster = NULL,
            verbose = interactive(),
            adaptive = FALSE,
            power = 1,
            ...)

## S3 method for class 'formula'
cv.grpnet(formula,
            data,
            use.rk = TRUE,
            weights = NULL,
            offset = NULL,
            alpha = c(0.01, 0.25, 0.5, 0.75, 1),
            gamma = c(3, 4, 5),
            type.measure = NULL,
            nfolds = 10,
            foldid = NULL,
            same.lambda = FALSE,
            parallel = FALSE,
            cluster = NULL,
            verbose = interactive(),
            adaptive = FALSE,
            power = 1,
            ...)
```

Arguments

x	Model (design) matrix of dimension nobs by nvars ($n \times p$).
y	Response vector of length n. Matrix inputs are allowed for binomial and multinomial families (see "Binomial and multinomial" section in grpnet).
group	Group label vector (factor, character, or integer) of length p. Predictors with the same label are grouped together for regularization.
formula	Model formula: a symbolic description of the model to be fitted. Uses the same syntax as lm and glm.
data	Optional data frame containing the variables referenced in formula.
use.rk	If TRUE (default), the rk.model.matrix function is used to build the model matrix. Otherwise, the model.matrix function is used to build the model matrix. Additional arguments to the rk.model.matrix function can be passed via the ... argument.
weights	Optional vector of length n with non-negative weights to use for weighted (penalized) likelihood estimation. Defaults to a vector of ones.
offset	Optional vector of length n with an a priori known term to be included in the model's linear predictor. Defaults to a vector of zeros.

`alpha`	Scalar or vector specifying the elastic net tuning parameter α. If `alpha` is a vector (default), then (a) the same `foldid` is used to compute the cross-validation error for each α, and (b) the solution for the optimal α is returned.
`gamma`	Scalar or vector specifying the penalty hyperparameter γ for MCP or SCAD. If `gamma` is a vector (default), then (a) the same `foldid` is used to compute the cross-validation error for each γ, and (b) the solution for the optimal γ is returned.
`type.measure`	Loss function for cross-validation. Options include: `"deviance"` for model deviance, `"mse"` for mean squared error, `"mae"` for mean absolute error, or `"class"` for classification error. Note that `"class"` is only available for binomial and multinomial families. The default is classification error (for binomial and multinomial) or deviance (others).
`nfolds`	Number of folds for cross-validation.
`foldid`	Optional vector of length n giving the fold identification for each observation. Must be coercible into a factor. After coersion, the `nfolds` argument is defined as `nfolds = nlevels(foldid)`.
`same.lambda`	Logical specfying if the same λ sequence should be used for fitting the model to each fold's data. If `FALSE` (default), the λ sequence is determined separately holding out each fold, and the λ sequence from the full model is used to align the predictions. If `TRUE`, the λ sequence from the full model is used to fit the model for each fold. The default often provides better (i.e., more stable) computational performance.
`parallel`	Logical specifying if sequential computing (default) or parallel computing should be used. If `TRUE`, the fitting for each fold is parallelized.
`cluster`	Optional cluster to use for parallel computing. If `parallel = TRUE` and `cluster = NULL`, then the cluster is defined `cluster = makeCluster(2L)`, which uses two cores. Recommended usage: `cluster = makeCluster(detectCores())`
`verbose`	Logical indicating if the fitting progress should be printed. Defaults to `TRUE` in interactive sessions and `FALSE` otherwise.
`adaptive`	Logical indicating if the adaptive group elastic net should be used (see Note).
`power`	If `adaptive = TRUE`, then the adaptive penalty weights are defined by dividing the original penalty weights by `tapply(coef, group, norm, type = "F")^power`.
`...`	Optional additional arguments for `grpnet` (e.g., `standardize`, `penalty.factor`, etc.)

Details

This function calls the `grpnet` function `nfolds+1` times: once on the full dataset to obtain the `lambda` sequence, and once holding out each fold's data to evaluate the prediction error. The syntax of (the default S3 method for) this function closely mimics that of the `cv.glmnet` function in the **glmnet** package (Friedman, Hastie, & Tibshirani, 2010).

Let $\mathbf{D}_u = \{\mathbf{y}_u, \mathbf{X}_u\}$ denote the u-th fold's data, let $\mathbf{D}_{[u]} = \{\mathbf{y}_{[u]}, \mathbf{X}_{[u]}\}$ denote the full dataset excluding the u-th fold's data, and let $\boldsymbol{\beta}_{\lambda[u]}$ denote the coefficient estimates obtained from fitting the model to $\mathbf{D}_{[u]}$ using the regularization parameter λ.

The cross-validation error for the u-th fold is defined as

$$E_u(\lambda) = C(\boldsymbol{\beta}_{\lambda[u]}, \mathbf{D}_u)$$

where $C(\cdot, \cdot)$ denotes the cross-validation loss function that is specified by `type.measure`. For example, the `"mse"` loss function is defined as

$$C(\boldsymbol{\beta}_{\lambda[u]}, \mathbf{D}_u) = \|\mathbf{y}_u - \mathbf{X}_u \boldsymbol{\beta}_{\lambda[u]}\|^2$$

where $\|\cdot\|$ denotes the L2 norm.

The mean cross-validation error `cvm` is defined as

$$\bar{E}(\lambda) = \frac{1}{v} \sum_{u=1}^{v} E_u(\lambda)$$

where v is the total number of folds. The standard error `cvsd` is defined as

$$S(\lambda) = \sqrt{\frac{1}{v(v-1)} \sum_{u=1}^{v} (E_u(\lambda) - \bar{E}(\lambda))^2}$$

which is the classic definition of the standard error of the mean.

Value

`lambda`	regularization parameter sequence for the full data
`cvm`	mean cross-validation error for each `lambda`
`cvsd`	estimated standard error of `cvm`
`cvup`	upper curve: `cvm` + `cvsd`
`cvlo`	lower curve: `cvm` - `cvsd`
`nzero`	number of non-zero groups for each `lambda`
`grpnet.fit`	fitted grpnet object for the full data
`lambda.min`	value of `lambda` that minimizes `cvm`
`lambda.1se`	largest `lambda` such that `cvm` is within one `cvsd` from the minimum (see Note)
`index`	two-element vector giving the indices of `lambda.min` and `lambda.1se` in the `lambda` vector, i.e., `c(minid, se1id)` as defined in the Note
`type.measure`	loss function for cross-validation (used for plot label)
`call`	matched call
`time`	runtime in seconds to perform k-fold CV tuning
`tune`	data frame containing the tuning results, i.e., min(cvm) for each combo of `alpha` and/or `gamma`

Note

When `adaptive = TRUE`, the adaptive group elastic net is used:
(1) an initial fit with `alpha = 0` estimates the `penalty.factor`
(2) a second fit using estimated `penalty.factor` is returned

`lambda.1se` is defined as follows:

```
minid <- which.min(cvm)
min1se <- cvm[minid] + cvsd[minid]
se1id <- which(cvm <= min1se)[1]
lambda.1se <- lambda[se1id]
```

Author(s)

Nathaniel E. Helwig <helwig@umn.edu>

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. doi:10.18637/jss.v033.i01

Helwig, N. E. (2024). Versatile descent algorithms for group regularization and variable selection in generalized linear models. *Journal of Computational and Graphical Statistics*. doi:10.1080/10618600.2024.2362232

See Also

`plot.cv.grpnet` for plotting the cross-validation error curve

`predict.cv.grpnet` for predicting from `cv.grpnet` objects

`grpnet` for fitting group elastic net regularization paths

Examples

```
################   family = "gaussian"   ##########**######

# load data
data(auto)

# 10-fold cv (formula method, response = mpg)
set.seed(1)
mod <- cv.grpnet(mpg ~ ., data = auto)

# print min and 1se solution info
mod

# plot cv error curve
plot(mod)



################   family = "multigaussian"   ##########**######
```

```
# load data
data(auto)

# 10-fold cv (formula method, response = (mpg, displacement))
y <- as.matrix(auto[,c(1,3)])
set.seed(1)
mod <- cv.grpnet(y ~ ., data = auto[,-c(1,3)], family = "multigaussian",
                 standardize.response = TRUE)

# print min and 1se solution info
mod

# plot cv error curve
plot(mod)



##########   family = "binomial"   ##########

# load data
data(auto)

# redefine origin (Domestic vs Foreign)
auto$origin <- ifelse(auto$origin == "American", "Domestic", "Foreign")

# 10-fold cv (default method, response = origin with 2 levels)
set.seed(1)
mod <- cv.grpnet(origin ~ ., data = auto, family = "binomial")

# print min and 1se solution info
mod

# plot cv error curve
plot(mod)



##########   family = "multinomial"   ##########

# load data
data(auto)

# 10-fold cv (formula method, response = origin with 3 levels)
set.seed(1)
mod <- cv.grpnet(origin ~ ., data = auto, family = "multinomial")

# print min and 1se solution info
mod

# plot cv error curve
plot(mod)
```

```
################   family = "poisson"   ################

# load data
data(auto)

# 10-fold cv (formula method, response = horsepower)
set.seed(1)
mod <- cv.grpnet(horsepower ~ ., data = auto, family = "poisson")

# print min and 1se solution info
mod

# plot cv error curve
plot(mod)



################   family = "negative.binomial"   ################

# load data
data(auto)

# 10-fold cv (formula method, response = horsepower)
set.seed(1)
mod <- cv.grpnet(horsepower ~ ., data = auto, family = "negative.binomial")

# print min and 1se solution info
mod

# plot cv error curve
plot(mod)



################   family = "Gamma"   ################

# load data
data(auto)

# 10-fold cv (formula method, response = origin)
set.seed(1)
mod <- cv.grpnet(mpg ~ ., data = auto, family = "Gamma")

# print min and 1se solution info
mod

# plot cv error curve
plot(mod)
```

```
################   family = "inverse.gaussian"   ################

# load data
data(auto)

# 10-fold cv (formula method, response = origin)
set.seed(1)
mod <- cv.grpnet(mpg ~ ., data = auto, family = "inverse.gaussian")

# print min and 1se solution info
mod

# plot cv error curve
plot(mod)
```

`family.grpnet` *Prepare 'family' Argument for grpnet*

Description

Takes in the `family` argument from `grpnet` and returns a list containing the information needed for fitting and/or tuning the model.

Usage

```
family.grpnet(object, theta = 1)
```

Arguments

`object`	one of the following characters specifying the exponential family: `"gaussian"`, `"multigaussian"`, `"binomial"`, `"multinomial"`, `"poisson"`, `"negative.binomial"`, `"Gamma"`, `"inverse.gaussian"`
`theta`	Additional ("size") parameter for negative binomial responses, where the variance function is defined as $V(\mu) = \mu + \mu^2/\theta$

Details

There is only one available link function for each `family`:
* gaussian (identity): $\mu = \mathbf{X}^\top \boldsymbol{\beta}$
* multigaussian (identity): $\mu = \mathbf{X}^\top \boldsymbol{\beta}$
* binomial (logit): $\log(\frac{\pi}{1-\pi}) = \mathbf{X}^\top \boldsymbol{\beta}$
* multinomial (symmetric): $\pi_\ell = \frac{\exp(\mathbf{X}^\top \boldsymbol{\beta}_\ell)}{\sum_{l=1}^{m} \exp(\mathbf{X}^\top \boldsymbol{\beta}_l)}$
* poisson (log): $\log(\mu) = \mathbf{X}^\top \boldsymbol{\beta}$
* negative.binomial (log): $\log(\mu) = \mathbf{X}^\top \boldsymbol{\beta}$
* Gamma (log): $\log(\mu) = \mathbf{X}^\top \boldsymbol{\beta}$
* inverse.gaussian (log): $\log(\mu) = \mathbf{X}^\top \boldsymbol{\beta}$

Value

List with components:

`family`	same as input object, i.e., character specifying the family
`linkinv`	function for computing inverse of link function
`dev.resids`	function for computing deviance residuals

Note

For `gaussian` family, this returns the full output produced by `gaussian`.

Author(s)

Nathaniel E. Helwig <helwig@umn.edu>

References

Helwig, N. E. (2024). Versatile descent algorithms for group regularization and variable selection in generalized linear models. *Journal of Computational and Graphical Statistics*. doi:10.1080/10618600.2024.2362232

See Also

`grpnet` for fitting group elastic net regularization paths

`cv.grpnet` for k-fold cross-validation of `lambda`

Examples

```
family.grpnet("gaussian")

family.grpnet("multigaussian")

family.grpnet("binomial")

family.grpnet("multinomial")

family.grpnet("poisson")

family.grpnet("negbin", theta = 10)

family.grpnet("Gamma")

family.grpnet("inverse.gaussian")
```

grpnet *Fit a Group Elastic Net Regularized GLM/GAM*

Description

Fits generalized linear/additive models with a group elastic net penalty using an adaptively bounded gradient descent (ABGD) algorithm (Helwig, 2024). Predictor groups can be manually input (default S3 method) or inferred from the model (S3 "formula" method). The regularization path is computed at a data-generated (default) or user-provided sequence of lambda values.

Usage

```
grpnet(x, ...)

## Default S3 method:
grpnet(x,
       y,
       group,
       family = c("gaussian", "multigaussian",
                  "binomial", "multinomial",
                  "poisson", "negative.binomial",
                  "Gamma", "inverse.gaussian"),
       weights = NULL,
       offset = NULL,
       alpha = 1,
       nlambda = 100,
       lambda.min.ratio = ifelse(nobs < nvars, 0.05, 0.0001),
       lambda = NULL,
       penalty.factor = NULL,
       penalty = c("LASSO", "MCP", "SCAD"),
       gamma = 4,
       theta = 1,
       standardized = !orthogonalized,
       orthogonalized = TRUE,
       intercept = TRUE,
       thresh = 1e-04,
       maxit = 1e05,
       proglang = c("Fortran", "R"),
       standardize.response = FALSE,
       ...)

## S3 method for class 'formula'
grpnet(formula,
       data,
       use.rk = TRUE,
       family = c("gaussian", "multigaussian",
                  "binomial", "multinomial",
```

```
                "poisson", "negative.binomial",
                "Gamma", "inverse.gaussian"),
    weights = NULL,
    offset = NULL,
    alpha = 1,
    nlambda = 100,
    lambda.min.ratio = ifelse(nobs < nvars, 0.05, 0.0001),
    lambda = NULL,
    penalty.factor = NULL,
    penalty = c("LASSO", "MCP", "SCAD"),
    gamma = 4,
    theta = 1,
    standardized = !orthogonalized,
    orthogonalized = TRUE,
    thresh = 1e-04,
    maxit = 1e05,
    proglang = c("Fortran", "R"),
    standardize.response = FALSE,
    ...)
```

Arguments

`x`	Model (design) matrix of dimension `nobs` by `nvars` ($n \times p$).
`y`	Response vector of length n. Matrix inputs are allowed for binomial and multinomial families (see "Binomial and multinomial" section).
`group`	Group label vector (factor, character, or integer) of length p. Predictors with the same label are grouped together for regularization.
`formula`	Model formula: a symbolic description of the model to be fitted. Uses the same syntax as `lm` and `glm`.
`data`	Optional data frame containing the variables referenced in `formula`.
`use.rk`	If `TRUE` (default), the `rk.model.matrix` function is used to build the model matrix. Otherwise, the `model.matrix` function is used to build the model matrix. Additional arguments to the `rk.model.matrix` function can be passed via the `...` argument.
`family`	Character specifying the assumed distribution for the response variable. Partial matching is allowed. Options include `"gaussian"` (real-valued vector), `"multigaussian"` (real-valued matrix), `"binomial"` (binary response), `"multinomial"` (multi-class response), `"poisson"` (count response), `"negative.binomial"` (count response), `"Gamma"` (positive real-valued), or `"inverse.gaussian"` (positive real-valued).
`weights`	Optional vector of length n with non-negative weights to use for weighted (penalized) likelihood estimation. Defaults to a vector of ones.
`offset`	Optional vector of length n with an a priori known term to be included in the model's linear predictor. Defaults to a vector of zeros.
`alpha`	Regularization hyperparameter satisfying $0 \leq \alpha \leq 1$ that gives the balance between the group L1 (lasso) and group L2 (ridge) penalty. Setting $\alpha = 1$ uses

a group lasso penalty, setting $\alpha = 0$ uses a group ridge penalty, and setting $0 < \alpha < 1$ uses a group elastic net group penalty.

`nlambda` Number of λ values to use in the regularization path. Ignored if `lambda` is provided.

`lambda.min.ratio`
The proportion $0 < \pi < 1$ that defines the minimum regularization parameter $\lambda_{\min}$ as a fraction of the maximum regularization parameter $\lambda_{\max}$ via the relationship $\lambda_{\min} = \pi\lambda_{\max}$. Ignored if `lambda` is provided. Note that $\lambda_{\max}$ is defined such that all penalized effects are shrunk to zero.

`lambda` Optional vector of user-supplied regularization parameter values.

`penalty.factor` Default S3 method: vector of length K giving the non-negative penalty weight for each predictor group. The order of the weights should correspond to the order of `levels(as.factor(group))`. Defaults to $\sqrt{p_k}$ for all $k = 1, \ldots, K$, where p_k is the number of coefficients in the k-th group. If `penalty.factor[k] = 0`, then the k-th group is unpenalized, and the corresponding term is always included in the model.

S3 "formula" method: named list giving the non-negative penalty weight for terms specified in the formula. Incomplete lists are allowed. Any term that is specified in `formula` but not in `penalty.factor` will be assigned the default penalty weight of $\sqrt{p_k}$. If `penalty.factor$z = 0`, then the variable `z` is unpenalized and always included in the model.

`penalty` Character specifying which (group) penalty to use: LASSO , MCP, or SCAD.

`gamma` Penalty hyperparameter that satisfies $\gamma > 1$ for MCP and $\gamma > 2$ for SCAD. Ignored for LASSO penalty.

`theta` Additional ("size") parameter for negative binomial responses, where the variance function is defined as $V(\mu) = \mu + \mu^2/\theta$

`standardized` Logical indicating whether the predictors should be groupwise standardized. If `TRUE`, each column of `x` is mean-centered and each predictor group's design matrix is scaled to have a mean-square of one before fitting the model. Regardless of whether standardization is used, the coefficients are always returned on the original data scale.

`orthogonalized` Logical indicating whether the predictors should be groupwise orthogonalized. If `TRUE` (default), each predictor group's design matrix is orthonormalized (i.e., $\mathbf{X}_k^\top \mathbf{X}_k = n\mathbf{I}_k$) before fitting the model. Regardless of whether orthogonalization is used, the coefficients are always returned on the original data scale.

`intercept` Logical indicating whether an intercept term should be included in the model. Note that the intercept is always unpenalized.

`thresh` Convergence threshold (tolerance). The algorithm is determined to have converged once the maximum relative change in the coefficients is below this threshold. See "Convergence" section.

`maxit` Maximum number of iterations to allow.

`proglang` Which programming language should be used to implement the ABGD algorithm? Options include `"Fortran"` (default) or `"R"`.

`standardize.response`
: Should columns of response be standardized to have unit variance before fitting the model? Only applicable when `family = "multigaussian"`. Note that coefficients are returned on the original (unstandardized) scale regardless of this input.

`...`
: Additional arguments used by the default or formula method.

Details

Consider a generalized linear model of the form

$$g(\mu) = \mathbf{X}^\top \boldsymbol{\beta}$$

where $\mu = E(Y|\mathbf{X})$ is the conditional expectation of the response Y given the predictor vector $\mathbf{X}$, the function $g(\cdot)$ is a user-specified (invertible) link function, and $\boldsymbol{\beta}$ are the unknown regression coefficients. Furthermore, suppose that the predictors are grouped, such as

$$\mathbf{X}^\top \boldsymbol{\beta} = \sum_{k=1}^{K} \mathbf{X}_k^\top \boldsymbol{\beta}_k$$

where $\mathbf{X} = (\mathbf{X}_1, \ldots, \mathbf{X}_K)$ is the grouped predictor vector, and $\boldsymbol{\beta} = (\boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_K)$ is the grouped coefficient vector.

Given n observations, this function finds the $\boldsymbol{\beta}$ that minimizes

$$L(\boldsymbol{\beta}|\mathbf{D}) + \lambda P_\alpha(\boldsymbol{\beta})$$

where $L(\boldsymbol{\beta}|\mathbf{D})$ is the loss function with $\mathbf{D} = \{\mathbf{y}, \mathbf{X}\}$ denoting the observed data, $P_\alpha(\boldsymbol{\beta})$ is the group elastic net penalty, and $\lambda \geq 0$ is the regularization parameter.

The loss function has the form

$$L(\boldsymbol{\beta}|\mathbf{D}) = \frac{1}{n} \sum_{i=1}^{n} w_i \ell_i(\boldsymbol{\beta}|\mathbf{D}_i)$$

where $w_i > 0$ are the user-supplied `weights`, and $\ell_i(\boldsymbol{\beta}|\mathbf{D}_i)$ is the i-th observation's contribution to the loss function. Note that $\ell(\cdot) = -\log(f_Y(\cdot))$ denotes the negative log-likelihood function for the given `family`.

The group elastic net penalty function has the form

$$P_\alpha(\boldsymbol{\beta}) = \alpha P_1(\boldsymbol{\beta}) + (1 - \alpha) P_2(\boldsymbol{\beta})$$

where $\alpha \in [0, 1]$ is the user-specified `alpha` value,

$$P_1(\boldsymbol{\beta}) = \sum_{k=1}^{K} \omega_k \|\boldsymbol{\beta}_k\|$$

is the group lasso penalty with $\omega_k \geq 0$ denoting the k-th group's `penalty.factor`, and

$$P_2(\boldsymbol{\beta}) = \frac{1}{2} \sum_{k=1}^{K} \omega_k \|\boldsymbol{\beta}_k\|^2$$

is the group ridge penalty. Note that $\|\boldsymbol{\beta}_k\|^2 = \boldsymbol{\beta}_k^\top \boldsymbol{\beta}_k$ denotes the squared Euclidean norm. When `penalty %in% c("MCP", "SCAD")`, the group L1 penalty $P_1(\boldsymbol{\beta})$ is replaced by the group MCP or group SCAD penalty.

Value

An object of class "grpnet" with the following elements:

call	matched call
a0	intercept sequence of length nlambda
beta	coefficient matrix of dimension nvars by nlambda
alpha	balance between the group L1 (lasso) and group L2 (ridge) penalty
lambda	sequence of regularization parameter values
family	exponential family defining the loss function
dev.ratio	proportion of (null) deviance explained for each lambda (= 1 - dev / nulldev)
nulldev	null deviance for each lambda
df	effective degrees of freedom for each lambda
nzgrp	number of non-zero groups for each lambda
nzcoef	number of non-zero coefficients for each lambda
xsd	standard deviation of x for each group
ylev	levels of response variable (only for binomial and multinomial families)
nobs	number of observations
group	group label vector
ngroups	number of groups K
npasses	number of iterations for each lambda
time	runtime in seconds to compute regularization path
offset	logical indicating if an offset was included
args	list of input argument values
formula	input formula (possibly after expansion)
term.labels	terms that appear in formula (if applicable)
rk.args	arguments for rk.model.matrix function (if applicable)

S3 "formula" method

Important: When using the S3 "formula" method, the S3 "predict" method forms the model matrix for the predictions by applying the model formula to the new data. As a result, to ensure that the corresponding S3 "predict" method works correctly, some formulaic features should be avoided.

Polynomials: When including polynomial terms, the poly function should be used with option raw = TRUE. Default use of the poly function (with raw = FALSE) will work for fitting the model, but will result in invalid predictions for new data. Polynomials can also be included via the I function, but this isn't recommended because the polynomials terms wouldn't be grouped together, i.e., the terms x and I(x^2) would be treated as two separate groups of size one instead of a single group of size two.

Splines: B-splines (and other spline bases) can be included via the S3 "formula" method. However, to ensure reasonable predictions for new data, it is necessary to specify the knots directly. For example, if x is a vector with entries between zero and one, the code bs(x, df = 5) will

not produce valid predictions for new data, but the code `bs(x, knots = c(0.25, 0.5, 0.75), Boundary.knots = c(0, 1))` will work as intended. Instead of attempting to integrate a call to `bs()` or `rk()` into the model formula, it is recommended that splines be included via the `use.rk = TRUE` argument.

Family argument and link functions

Unlike the `glm` function, the `family` argument of the `grpnet` function
* should be a character vector (not a `family` object)
* does not allow for specification of a link function

Currently, there is only one available link function for each `family`:
* gaussian (identity): $\mu = \mathbf{X}^\top \boldsymbol{\beta}$
* binomial (logit): $\log(\frac{\pi}{1-\pi}) = \mathbf{X}^\top \boldsymbol{\beta}$
* multinomial (symmetric): $\pi_\ell = \frac{\exp(\mathbf{X}^\top \boldsymbol{\beta}_\ell)}{\sum_{l=1}^{m} \exp(\mathbf{X}^\top \boldsymbol{\beta}_l)}$
* poisson (log): $\log(\mu) = \mathbf{X}^\top \boldsymbol{\beta}$
* negative.binomial (log): $\log(\mu) = \mathbf{X}^\top \boldsymbol{\beta}$
* Gamma (log): $\log(\mu) = \mathbf{X}^\top \boldsymbol{\beta}$
* inverse.gaussian (log): $\log(\mu) = \mathbf{X}^\top \boldsymbol{\beta}$

Binomial and multinomial

For `"binomial"` responses, three different possibilities exist for the input response:
1. vector coercible into a factor with two levels
2. matrix with two columns (# successes, # failures)
3. numeric vector with entries between 0 and 1

In this case, the `weights` argument should be used specify the total number of trials.

For `"multinomial"` responses, two different possibilities exist for the input reponse:
1. vector coercible into a factor with more than two levels
2. matrix of integers (counts) for each category level

Convergence

The algorithm is determined to have converged once

$\max_j \frac{|\beta_j - \beta_j^{\text{old}}|}{1+|\beta_j^{\text{old}}|} < \epsilon$

where $j \in \{1, \ldots, p\}$ and ϵ is the `thresh` argument.

Note

The syntax of (the default S3 method for) this function closely mimics that of the `glmnet` function in the **glmnet** package (Friedman, Hastie, & Tibshirani, 2010).

Author(s)

Nathaniel E. Helwig <helwig@umn.edu>

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. doi:10.18637/jss.v033.i01

Helwig, N. E. (2024). Versatile descent algorithms for group regularization and variable selection in generalized linear models. *Journal of Computational and Graphical Statistics*. doi:10.1080/10618600.2024.2362232

See Also

`plot.grpnet` for plotting the regularization path

`predict.grpnet` for predicting from `grpnet` objects

`cv.grpnet` for k-fold cross-validation of `lambda`

Examples

```
################   family = "gaussian"   ################

# load data
data(auto)

# fit model (formula method, response = mpg)
mod <- grpnet(mpg ~ ., data = auto)

# print regularization path info
mod

# plot coefficient paths
plot(mod)



################   family = "multigaussian"   ################

# load data
data(auto)

# fit model (formula method, response = (mpg, displacement))
y <- as.matrix(auto[,c(1,3)])
mod <- grpnet(y ~ ., data = auto[,-c(1,3)], family = "multigaussian",
              standardize.response = TRUE)

# print regularization path info
mod

# plot coefficient paths
plot(mod)



################   family = "binomial"   ################
```

```
# load data
data(auto)

# redefine origin (Domestic vs Foreign)
auto$origin <- ifelse(auto$origin == "American", "Domestic", "Foreign")

# fit model (formula method, response = origin with 2 levels)
mod <- grpnet(origin ~ ., data = auto, family = "binomial")

# print regularization path info
mod

# plot coefficient paths
plot(mod)



#################   family = "multinomial"   ##################

# load data
data(auto)

# fit model (formula method, response = origin with 3 levels)
mod <- grpnet(origin ~ ., data = auto, family = "multinomial")

# print regularization path info
mod

# plot coefficient paths
plot(mod)



#################   family = "poisson"   #################

# load data
data(auto)

# fit model (formula method, response = horsepower)
mod <- grpnet(horsepower ~ ., data = auto, family = "poisson")

# print regularization path info
mod

# plot coefficient paths
plot(mod)



#################   family = "negative.binomial"   ##################

# load data
```

```
data(auto)

# fit model (formula method, response = horsepower)
mod <- grpnet(horsepower ~ ., data = auto, family = "negative.binomial")

# print regularization path info
mod

# plot coefficient paths
plot(mod)



################   family = "Gamma"   ################

# load data
data(auto)

# fit model (formula method, response = mpg)
mod <- grpnet(mpg ~ ., data = auto, family = "Gamma")

# print regularization path info
mod

# plot coefficient paths
plot(mod)



################   family = "inverse.gaussian"   ################

# load data
data(auto)

# fit model (formula method, response = mpg)
mod <- grpnet(mpg ~ ., data = auto, family = "inverse.gaussian")

# print regularization path info
mod

# plot coefficient paths
plot(mod)
```

`plot.cv.grpnet` *Plot Cross-Validation Curve for cv.grpnet Fits*

Description

Plots the mean cross-validation error, along with lower and upper standard deviation curves, as a function of `log(lambda)`.

Usage

```
## S3 method for class 'cv.grpnet'
plot(x, sign.lambda = 1, nzero = TRUE, ...)
```

Arguments

x	Object of class "cv.grpnet"
sign.lambda	Default plots log(lambda) on the x-axis. Set to -1 to plot -1*log(lambda) on the x-axis instead.
nzero	Should the number of non-zero groups be printed on the top of the x-axis?
...	Additional arguments passed to the plot function.

Details

Produces cross-validation plot only (i.e., nothing is returned).

Value

No return value (produces a plot)

Note

Syntax and functionality were modeled after the plot.cv.glmnet function in the **glmnet** package (Friedman, Hastie, & Tibshirani, 2010).

Author(s)

Nathaniel E. Helwig <helwig@umn.edu>

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. doi:10.18637/jss.v033.i01

Helwig, N. E. (2024). Versatile descent algorithms for group regularization and variable selection in generalized linear models. *Journal of Computational and Graphical Statistics*. doi:10.1080/10618600.2024.2362232

See Also

cv.grpnet for k-fold cross-validation of lambda

plot.grpnet for plotting the regularization path

Examples

```
# see 'cv.grpnet' for plotting examples
?cv.grpnet
```

`plot.grpnet` *Plot Regularization Path for grpnet Fits*

Description

Creates a profile plot of the reguarlization paths for a fit group elastic net regularized GLM (grpnet) object.

Usage

```
## S3 method for class 'grpnet'
plot(x, type = c("coef", "imp", "norm", "znorm"),
     newx, newdata, intercept = FALSE,
     color.by.group = TRUE, col = NULL, ...)
```

Arguments

`x`	Object of class "grpnet"
`type`	What to plot on the Y-axis: "coef" for coefficient values, "imp" for importance of each group's contribution, "norm" for L2 norm of coefficients for each group, or "znorm" for L2 norm of standardized coefficients for each group.
`newx`	Matrix of new `x` scores for prediction (default S3 method). Ignored unless `type = "imp"`.
`newdata`	Data frame of new `data` scores for prediction (S3 "formula" method). Ignored unless `type = "imp"`.
`intercept`	Should the intercept be included in the plot?
`color.by.group`	If `TRUE` (default), the coefficient paths are colored according to their group membership using the colors in `col`. If `FALSE`, all coefficient paths are plotted the same color.
`col`	If `color.by.group = TRUE`, this should be a vector of length K giving a color label for each group. If `color.by.group = FASLE`, this should be a character specifying a single (common) color. Default of `col = NULL` is the same as `col = 1:K` or `col = "black"`.
`...`	Additional arguments passed to the `plot` function.

Details

Syntax and functionality were modeled after the `plot.glmnet` function in the **glmnet** package (Friedman, Hastie, & Tibshirani, 2010).

Value

Produces a profile plot showing the requested type (y-axis) as a function of `log(lambda)` (x-axis).

Note

If x is a multigaussian or multinomial model, the coefficients for each response dimension/class are plotted in a separate plot.

Author(s)

Nathaniel E. Helwig <helwig@umn.edu>

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. doi:10.18637/jss.v033.i01

Helwig, N. E. (2024). Versatile descent algorithms for group regularization and variable selection in generalized linear models. *Journal of Computational and Graphical Statistics*. doi:10.1080/10618600.2024.2362232

See Also

`grpnet` for fitting grpnet regularization paths

`plot.cv.grpnet` for plotting `cv.grpnet` objects

Examples

```
# see 'grpnet' for plotting examples
?grpnet
```

---

predict.cv.grpnet *Predict Method for cv.grpnet Fits*

Description

Obtain predictions from a cross-validated group elastic net regularized GLM (cv.grpnet) object.

Usage

```
## S3 method for class 'cv.grpnet'
predict(object,
        newx,
        newdata,
        s = c("lambda.1se", "lambda.min"),
        type = c("link", "response", "class", "terms",
                 "importance", "coefficients", "nonzero", "groups",
                 "ncoefs", "ngroups", "norm", "znorm"),
        ...)
```

Arguments

`object`	Object of class "cv.grpnet"
`newx`	Matrix of new `x` scores for prediction (default S3 method). Must have p columns arranged in the same order as the `x` matrix used to fit the model.
`newdata`	Data frame of new `data` scores for prediction (S3 "formula" method). Must contain all variables in the `formula` used to fit the model.
`s`	Lambda value(s) at which predictions should be obtained. Can input a character ("lambda.min" or "lambda.1se") or a numeric vector. Default of "lambda.min" uses the `lambda` value that minimizes the mean cross-validated error.
`type`	Type of prediction to return. "link" gives predictions on the link scale (η). "response" gives predictions on the mean scale (μ). "class" gives predicted class labels (for "binomial" and "multinomial" families). "terms" gives the predictions for each term (group) in the model (η_k). "importance" gives the variable importance index for each term (group) in the model. "coefficients" returns the coefficients used for predictions. "nonzero" returns a list giving the indices of non-zero coefficients for each `s`. "groups" returns a list giving the labels of non-zero groups for each `s`. "ncoefs" returns the number of non-zero coefficients for each `s`. "ngroups" returns the number of non-zero groups for each `s`. "norm" returns the L2 norm of each group's (raw) coefficients for each `s`. "znorm" returns the L2 norm of each group's standardized coefficients for each `s`.
`...`	Additional arguments (ignored)

Details

Predictions are calculated from the `grpnet` object fit to the full sample of data, which is stored as `object$grpnet.fit`

See `predict.grpnet` for further details on the calculation of the different types of predictions.

Value

Depends on three factors...
1. the exponential family distribution
2. the length of the input `s`
3. the `type` of prediction requested

See `predict.grpnet` for details

Note

Syntax is inspired by the `predict.cv.glmnet` function in the **glmnet** package (Friedman, Hastie, & Tibshirani, 2010).

Author(s)

Nathaniel E. Helwig <helwig@umn.edu>

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. doi:10.18637/jss.v033.i01

Helwig, N. E. (2024). Versatile descent algorithms for group regularization and variable selection in generalized linear models. *Journal of Computational and Graphical Statistics*. doi:10.1080/10618600.2024.2362232

See Also

`cv.grpnet` for k-fold cross-validation of `lambda`

`predict.grpnet` for predicting from `grpnet` objects

Examples

```
################   family = "gaussian"   ################

# load data
data(auto)

# 10-fold cv (formula method, response = mpg)
set.seed(1)
mod <- cv.grpnet(mpg ~ ., data = auto)

# get fitted values at "lambda.1se"
fit.1se <- predict(mod, newdata = auto)

# get fitted values at "lambda.min"
fit.min <- predict(mod, newdata = auto, s = "lambda.min")

# compare mean absolute error for two solutions
mean(abs(auto$mpg - fit.1se))
mean(abs(auto$mpg - fit.min))



################   family = "multigaussian"   ################

# load data
data(auto)

# 10-fold cv (formula method, response = (mpg, displacement))
y <- as.matrix(auto[,c(1,3)])
set.seed(1)
mod <- cv.grpnet(y ~ ., data = auto[,-c(1,3)], family = "multigaussian",
                 standardize.response = TRUE)

# get fitted values at "lambda.1se"
fit.1se <- predict(mod, newdata = auto)

# get fitted values at "lambda.min"
fit.min <- predict(mod, newdata = auto, s = "lambda.min")
```

```
# compare mean absolute error for two solutions
mean(abs(y - fit.1se))
mean(abs(y - fit.min))



#################   family = "binomial"   #################

# load data
data(auto)

# redefine origin (Domestic vs Foreign)
auto$origin <- ifelse(auto$origin == "American", "Domestic", "Foreign")

# 10-fold cv (default method, response = origin with 2 levels)
set.seed(1)
mod <- cv.grpnet(origin ~ ., data = auto, family = "binomial")

# get predicted classes at "lambda.1se"
fit.1se <- predict(mod, newdata = auto, type = "class")

# get predicted classes at "lambda.min"
fit.min <- predict(mod, newdata = auto, type = "class", s = "lambda.min")

# compare misclassification rate for two solutions
1 - mean(auto$origin == fit.1se)
1 - mean(auto$origin == fit.min)



#################   family = "multinomial"   #################

# load data
data(auto)

# 10-fold cv (formula method, response = origin with 3 levels)
set.seed(1)
mod <- cv.grpnet(origin ~ ., data = auto, family = "multinomial")

# get predicted classes at "lambda.1se"
fit.1se <- predict(mod, newdata = auto, type = "class")

# get predicted classes at "lambda.min"
fit.min <- predict(mod, newdata = auto, type = "class", s = "lambda.min")

# compare misclassification rate for two solutions
1 - mean(auto$origin == fit.1se)
1 - mean(auto$origin == fit.min)



#################   family = "poisson"   #################
```

```
# load data
data(auto)

# 10-fold cv (formula method, response = horsepower)
set.seed(1)
mod <- cv.grpnet(horsepower ~ ., data = auto, family = "poisson")

# get fitted values at "lambda.1se"
fit.1se <- predict(mod, newdata = auto, type = "response")

# get fitted values at "lambda.min"
fit.min <- predict(mod, newdata = auto, type = "response", s = "lambda.min")

# compare mean absolute error for two solutions
mean(abs(auto$horsepower - fit.1se))
mean(abs(auto$horsepower - fit.min))



##############   family = "negative.binomial"   ##############

# load data
data(auto)

# 10-fold cv (formula method, response = horsepower)
set.seed(1)
mod <- cv.grpnet(horsepower ~ ., data = auto, family = "negative.binomial")

# get fitted values at "lambda.1se"
fit.1se <- predict(mod, newdata = auto, type = "response")

# get fitted values at "lambda.min"
fit.min <- predict(mod, newdata = auto, type = "response", s = "lambda.min")

# compare mean absolute error for two solutions
mean(abs(auto$horsepower - fit.1se))
mean(abs(auto$horsepower - fit.min))



##############   family = "Gamma"   ##############

# load data
data(auto)

# 10-fold cv (formula method, response = origin)
set.seed(1)
mod <- cv.grpnet(mpg ~ ., data = auto, family = "Gamma")

# get fitted values at "lambda.1se"
fit.1se <- predict(mod, newdata = auto, type = "response")
```

```
# get fitted values at "lambda.min"
fit.min <- predict(mod, newdata = auto, type = "response", s = "lambda.min")

# compare mean absolute error for two solutions
mean(abs(auto$mpg - fit.1se))
mean(abs(auto$mpg - fit.min))



#######************   family = "inverse.gaussian"   #######************

# load data
data(auto)

# 10-fold cv (formula method, response = origin)
set.seed(1)
mod <- cv.grpnet(mpg ~ ., data = auto, family = "inverse.gaussian")

# get fitted values at "lambda.1se"
fit.1se <- predict(mod, newdata = auto, type = "response")

# get fitted values at "lambda.min"
fit.min <- predict(mod, newdata = auto, type = "response", s = "lambda.min")

# compare mean absolute error for two solutions
mean(abs(auto$mpg - fit.1se))
mean(abs(auto$mpg - fit.min))
```

`predict.grpnet` *Predict Method for grpnet Fits*

Description

Obtain predictions from a fit group elastic net regularized GLM (grpnet) object.

Usage

```
## S3 method for class 'grpnet'
predict(object,
        newx,
        newdata,
        s = NULL,
        type = c("link", "response", "class", "terms",
                 "importance", "coefficients", "nonzero", "groups",
                 "ncoefs", "ngroups", "norm", "znorm"),
        ...)
```

Arguments

`object`	Object of class "grpnet"
`newx`	Matrix of new `x` scores for prediction (default S3 method). Must have p columns arranged in the same order as the `x` matrix used to fit the model. Ignored for the last six types of predictions.
`newdata`	Data frame of new `data` scores for prediction (S3 "formula" method). Must contain all variables in the `formula` used to fit the model. Ignored for the last six types of predictions.
`s`	Lambda value(s) at which predictions should be obtained. Default uses `s = object$lambda`. Interpolation is used for `s` values that are not included in `object$lambda`.
`type`	Type of prediction to return. "link" gives predictions on the link scale (η). "response" gives predictions on the mean scale (μ). "class" gives predicted class labels (for "binomial" and "multinomial" families). "terms" gives the predictions for each term (group) in the model (η_k). "importance" gives the variable importance index for each term (group) in the model. "coefficients" returns the coefficients used for predictions. "nonzero" returns a list giving the indices of non-zero coefficients for each `s`. "groups" returns a list giving the labels of non-zero groups for each `s`. "ncoefs" returns the number of non-zero coefficients for each `s`. "ngroups" returns the number of non-zero groups for each `s`. "norm" returns the L2 norm of each group's (raw) coefficients for each `s`. "znorm" returns the L2 norm of each group's standardized coefficients for each `s`.
`...`	Additional arguments (ignored)

Details

When `type == "link"`, the predictions for each λ have the form

$$\boldsymbol{\eta}_\lambda = \mathbf{X}_{\text{new}}\boldsymbol{\beta}_\lambda$$

where $\mathbf{X}_{\text{new}}$ is the argument `newx` (or the design matrix created from `newdata` by applying `object$formula`) and $\boldsymbol{\beta}_\lambda$ is the coefficient vector corresponding to λ.

When `type == "response"`, the predictions for each λ have the form

$$\boldsymbol{\mu}_\lambda = g^{-1}(\boldsymbol{\eta}_\lambda)$$

where $g^{-1}(\cdot)$ is the inverse link function stored in `object$family$linkinv`.

When `type == "class"`, the predictions for each λ have the form

$$\mathbf{y}_\lambda = \arg\max_l \boldsymbol{\mu}_\lambda(l)$$

where $\boldsymbol{\mu}_\lambda(l)$ gives the predicted probability that each observation belongs to the l-th category (for $l = 1, \ldots, m$) using the regularization parameter λ.

When `type == "terms"`, the groupwise predictions for each λ have the form

$$\boldsymbol{\eta}_{k\lambda} = \mathbf{X}_k^{(\text{new})}\boldsymbol{\beta}_{k\lambda}$$

where $\mathbf{X}_k^{(\text{new})}$ is the portion of the argument `newx` (or the design matrix created from `newdata` by applying `object$formula`) that corresponds to the k-th term/group, and $\boldsymbol{\beta}_{k\lambda}$ are the corresponding coefficients.

When `type == "importance"`, the variable importance indices are defined as

$$\pi_k = \left(\boldsymbol{\eta}_{k\lambda}^\top \mathbf{C} \boldsymbol{\eta}_{0\lambda}\right) \left(\boldsymbol{\eta}_{0\lambda}^\top \mathbf{C} \boldsymbol{\eta}_{0\lambda}\right)^{-1}$$

where $\mathbf{C} = (\mathbf{I}_n - \frac{1}{n}\mathbf{1}_n\mathbf{1}_n^\top)$ denotes a centering matrix, and $\boldsymbol{\eta}_{0\lambda} = \sum_{k=1}^K \boldsymbol{\eta}_{k\lambda}$. Note that $\sum_{k=1}^K \pi_k = 1$, but some π_k could be negative. When they are positive, π_k gives the approximate proportion of model (explained) variation that is attributed to the k-th term.

Value

Depends on three factors...
1. the exponential family distribution
2. the length of the input `s`
3. the `type` of prediction requested

For most response variables, the typical output will be...

* a matrix of dimension `c(newnobs, length(s))` if `length(s) > 1`
* a vector of length `newnobs` if length(s) == 1

For multigaussian and multinomial response variables, the typical output will be...

* an array of dimension `c(newnobs, length(object$ylev), length(s))` if `type %in% c("link", "response")`
* a matrix of dimension `c(newobs, length(s))` if `type == "class"`

Note: if `type == "class"`, then the output will be the same class as `object$ylev`. Otherwise, the output will be real-valued (or integer for the counts).

If `type == "terms"` and `!(family %in% c("multigaussian","multinomial"))`, the output will be...

* an array of dimension `c(newnobs, nterms, length(s))` if `length(s) > 1`
* a matrix of dimension `c(newnobs, nterms)` if `length(s) == 1`

If `type == "terms"` and `family %in% c("multigaussian","multinomial")`, the output will be a list of length `length(object$ylev)` where each element gives the terms for the corresponding response dimension/class.

If `type == "importance"` and `family != "multinomial"`, the output will be...

* a matrix of dimension `c(nterms, length(s))` if `length(s) > 1`
* a vector of length `nterms` if `length(s) == 1`

If `type == "importance"` and `family == "multinomial"`, the output will be a list of length `length(object$ylev)` where each element gives the importance for the corresponding response class. If `length(s) == 1`, the output will be simplified to matrix.

If `type == "coefficients"`, the output will be the same as that produced by `coef.grpnet`.

If `type == "nonzero"`, the output will be a list of length `length(s)` where each element is a vector of integers (indices).

If `type == "groups"`, the output will be a list of length `length(s)` where each element is a vector of characters (`term.labels`).

If `type %in% c("ncoefs", "ngroups")`, the output will be a vector of length `length(s)` where each element is an integer.

If `type == "norm"`, the output will be a matrix of dimension `c(K, length(s))`, where each cell gives the L2 norm for the corresponding group and smoothing parameter. Note that `K` denotes the number of groups.

Note

Some internal code (e.g., used for the interpolation) is borrowed from the `predict.glmnet` function in the **glmnet** package (Friedman, Hastie, & Tibshirani, 2010).

Author(s)

Nathaniel E. Helwig <helwig@umn.edu>

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. doi:10.18637/jss.v033.i01

Helwig, N. E. (2024). Versatile descent algorithms for group regularization and variable selection in generalized linear models. *Journal of Computational and Graphical Statistics*. doi:10.1080/10618600.2024.2362232

See Also

`grpnet` for fitting grpnet regularization paths

`predict.cv.grpnet` for predicting from `cv.grpnet` objects

Examples

```
#########------#########   family = "gaussian"   #########------#########

# load data
data(auto)

# fit model (formula method, response = mpg)
mod <- grpnet(mpg ~ ., data = auto)

# get fitted values for regularization path (output = 392 x 100 matrix)
fit.path <- predict(mod, newdata = auto)

# get fitted values at 3 particular points (output = 392 x 3 matrix)
fit.some <- predict(mod, newdata = auto, s = c(1.5, 1, 0.5))

# compare rmse for solutions
```

```
rmse.path <- sqrt(colMeans((auto$mpg - fit.path)^2))
rmse.some <- sqrt(colMeans((auto$mpg - fit.some)^2))
plot(log(mod$lambda), rmse.path, cex = 0.5)
points(log(c(1.5, 1, 0.5)), rmse.some, pch = 0, col = "red")



##############   family = "binomial"   ##############

# load data
data(auto)

# redefine origin (Domestic vs Foreign)
auto$origin <- ifelse(auto$origin == "American", "Domestic", "Foreign")

# fit model (formula method, response = origin with 2 levels)
mod <- grpnet(origin ~ ., data = auto, family = "binomial")

# get predicted classes for regularization path (output = 392 x 100 matrix)
fit.path <- predict(mod, newdata = auto, type = "class")

# get predicted classes at 3 particular points (output = 392 x 3 matrix)
fit.some <- predict(mod, newdata = auto, type = "class", s = c(.15, .1, .05))

# compare misclassification rate for solutions
miss.path <- 1 - colMeans(auto$origin == fit.path)
miss.some <- 1 - colMeans(auto$origin == fit.some)
plot(log(mod$lambda), miss.path, cex = 0.5)
points(log(c(.15, .1, .05)), miss.some, pch = 0, col = "red")



###############   family = "multinomial"   ###############

# load data
data(auto)

# fit model (formula method, response = origin with 3 levels)
mod <- grpnet(origin ~ ., data = auto, family = "multinomial")

# get predicted classes for regularization path (output = 392 x 100 matrix)
fit.path <- predict(mod, newdata = auto, type = "class")

# get predicted classes at 3 particular points (output = 392 x 3 matrix)
fit.some <- predict(mod, newdata = auto, type = "class", s = c(.1, .01, .001))

# compare misclassification rate for solutions
miss.path <- 1 - colMeans(auto$origin == fit.path)
miss.some <- 1 - colMeans(auto$origin == fit.some)
plot(log(mod$lambda), miss.path, cex = 0.5)
points(log(c(.1, .01, .001)), miss.some, pch = 0, col = "red")
```

```
################   family = "poisson"   ########**########

# load data
data(auto)

# fit model (formula method, response = horsepower)
mod <- grpnet(horsepower ~ ., data = auto, family = "poisson")

# get fitted values for regularization path (output = 392 x 100 matrix)
fit.path <- predict(mod, newdata = auto, type = "response")

# get fitted values at 3 particular points (output = 392 x 3 matrix)
fit.some <- predict(mod, newdata = auto, type = "response", s = c(15, 10, 5))

# compare rmse for solutions
rmse.path <- sqrt(colMeans((auto$horsepower - fit.path)^2))
rmse.some <- sqrt(colMeans((auto$horsepower - fit.some)^2))
plot(log(mod$lambda), rmse.path, cex = 0.5)
points(log(c(15, 10, 5)), rmse.some, pch = 0, col = "red")



################   family = "negative.binomial"   ########**########

# load data
data(auto)

# fit model (formula method, response = horsepower)
mod <- grpnet(horsepower ~ ., data = auto, family = "negative.binomial")

# get fitted values for regularization path (output = 392 x 100 matrix)
fit.path <- predict(mod, newdata = auto, type = "response")

# get fitted values at 3 particular points (output = 392 x 3 matrix)
fit.some <- predict(mod, newdata = auto, type = "response", s = c(0.1, 0.01, 0.001))

# compare rmse for solutions
rmse.path <- sqrt(colMeans((auto$horsepower - fit.path)^2))
rmse.some <- sqrt(colMeans((auto$horsepower - fit.some)^2))
plot(log(mod$lambda), rmse.path, cex = 0.5)
points(log(c(0.1, 0.01, 0.001)), rmse.some, pch = 0, col = "red")



################   family = "Gamma"   ########**########

# load data
data(auto)

# fit model (formula method, response = mpg)
mod <- grpnet(mpg ~ ., data = auto, family = "Gamma")
```

```
# get fitted values for regularization path (output = 392 x 100 matrix)
fit.path <- predict(mod, newdata = auto, type = "response")

# get fitted values at 3 particular points (output = 392 x 3 matrix)
fit.some <- predict(mod, newdata = auto, type = "response", s = c(0.1, 0.01, 0.001))

# compare rmse for solutions
rmse.path <- sqrt(colMeans((auto$mpg - fit.path)^2))
rmse.some <- sqrt(colMeans((auto$mpg - fit.some)^2))
plot(log(mod$lambda), rmse.path, cex = 0.5)
points(log(c(0.1, 0.01, 0.001)), rmse.some, pch = 0, col = "red")



################   family = "inverse.gaussian"   ################

# load data
data(auto)

# fit model (formula method, response = mpg)
mod <- grpnet(mpg ~ ., data = auto, family = "inverse.gaussian")

# get fitted values for regularization path (output = 392 x 100 matrix)
fit.path <- predict(mod, newdata = auto, type = "response")

# get fitted values at 3 particular points (output = 392 x 3 matrix)
fit.some <- predict(mod, newdata = auto, type = "response", s = c(0.005, 0.001, 0.0001))

# compare rmse for solutions
rmse.path <- sqrt(colMeans((auto$mpg - fit.path)^2))
rmse.some <- sqrt(colMeans((auto$mpg - fit.some)^2))
plot(log(mod$lambda), rmse.path, cex = 0.5)
points(log(c(0.005, 0.001, 0.0001)), rmse.some, pch = 0, col = "red")
```

`print` *S3 'print' Methods for grpnet*

Description

Prints some basic information about the coefficients (for `coef.grpnet` objects), observed cross-validation error (for `cv.grpnet` objects), or the computed regularization path (for `grpnet` objects).

Usage

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

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

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

Arguments

x	an object of class coef.grpnet, cv.grpnet, or grpnet
digits	the number of digits to print (must be a positive integer)
...	additional arguments for print (currently ignored)

Details

For coef.grpnet objects, prints the non-zero coefficients and uses "." for coefficients shrunk to zero.

For cv.grpnet objects, prints the function call, the cross-validation type.measure, and a two-row table with information about the min and 1se solutions.

For grpnet objects, prints a data frame with columns
* nGrp: number of non-zero groups for each lambda
* Df: effective degrees of freedom for each lambda
* %Dev: percentage of null deviance explained for each lambda
* Lambda: the values of lambda

Value

No return value (produces a printout)

Note

Some syntax and functionality were modeled after the print functions in the **glmnet** package (Friedman, Hastie, & Tibshirani, 2010).

Author(s)

Nathaniel E. Helwig <helwig@umn.edu>

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. doi:10.18637/jss.v033.i01

Helwig, N. E. (2024). Versatile descent algorithms for group regularization and variable selection in generalized linear models. *Journal of Computational and Graphical Statistics*. doi:10.1080/10618600.2024.2362232

See Also

coef.grpnet for extracting coefficients

cv.grpnet for k-fold cross-validation of lambda

grpnet for fitting grpnet regularization paths

Examples

```
# see 'coef.grpnet' for coefficient printing examples
?coef.grpnet

# see 'cv.grpnet' for cross-validation error printing examples
?cv.grpnet

# see 'grpnet' for regularization path printing examples
?grpnet
```

---

rk *Reproducing Kernel Basis*

Description

Generate a reproducing kernel basis matrix for a nominal, ordinal, or polynomial smoothing spline.

Usage

```
rk(x, df = NULL, knots = NULL, m = NULL, intercept = FALSE,
   Boundary.knots = NULL, warn.outside = TRUE,
   periodic = FALSE, xlev = levels(x))
```

Arguments

x	the predictor vector of length n. Can be a factor, integer, or numeric, see Note.
df	the degrees of freedom, i.e., number of knots to place at quantiles of x. Defaults to 5 but ignored if knots are provided.
knots	the breakpoints (knots) defining the spline. If knots are provided, the df is defined as length(unique(c(knots, Boundary.knots))).
m	the derivative penalty order: 0 = ordinal spline, 1 = linear spline, 2 = cubic spline, 3 = quintic spline
intercept	should an intercept be included in the basis?
Boundary.knots	the boundary points for spline basis. Defaults to range(x).
warn.outside	if TRUE, a warning is provided when x values are outside of the Boundary.knots
periodic	should the spline basis functions be constrained to be periodic with respect to the Boundary.knots?
xlev	levels of x (only applicable if x is a factor)

Details

Given a vector of function realizations f, suppose that $f = X\beta$, where X is the (unregularized) spline basis and β is the coefficient vector. Let Q denote the postive semi-definite penalty matrix, such that $\beta^\top Q\beta$ defines the roughness penalty for the spline. See Helwig (2017) for the form of X and Q for the various types of splines.

Consider the spectral parameterization of the form $f = Z\alpha$ where

$$Z = XQ^{-1/2}$$

is the regularized spline basis (that is returned by this function), and $\alpha = Q^{1/2}\beta$ are the reparameterized coefficients. Note that $X\beta = Z\alpha$ and $\beta^\top Q\beta = \alpha^\top\alpha$, so the spectral parameterization absorbs the penalty into the coefficients (see Helwig, 2021, 2024).

Syntax of this function is designed to mimic the syntax of the `bs` function.

Value

Returns a basis function matrix of dimension `n` by `df` (plus 1 if an `intercept` is included) with the following attributes:

`df`	degrees of freedom
`knots`	knots for spline basis
`m`	derivative penalty order
`intercept`	was an intercept included?
`Boundary.knots`	boundary points of `x`
`periodic`	is the basis periodic?
`xlev`	factor levels (if applicable)

Note

The (default) type of spline basis depends on the `class` of the input `x` object:

* If `x` is an unordered factor, then a nominal spline basis is used

* If `x` is an ordered factor (and `m = NULL`), then an ordinal spline basis is used

* If `x` is an integer or numeric (and `m = NULL`), then a cubic spline basis is used

Note that you can override the default behavior by specifying the `m` argument.

Author(s)

Nathaniel E. Helwig <helwig@umn.edu>

References

Helwig, N. E. (2017). Regression with ordered predictors via ordinal smoothing splines. Frontiers in Applied Mathematics and Statistics, 3(15), 1-13. doi:10.3389/fams.2017.00015

Helwig, N. E. (2021). Spectrally sparse nonparametric regression via elastic net regularized smoothers. Journal of Computational and Graphical Statistics, 30(1), 182-191. doi:10.1080/10618600.2020.1806855

Helwig, N. E. (2024). Precise tensor product smoothing via spectral splines. Stats, 7(1), 34-53. doi:10.3390/stats7010003

Examples

```
################   NOMINAL SPLINE BASIS   ################

x <- as.factor(LETTERS[1:5])
basis <- rk(x)
plot(1:5, basis[,1], t = "l", ylim = extendrange(basis))
for(j in 2:ncol(basis)){
  lines(1:5, basis[,j], col = j)
}


################   ORDINAL SPLINE BASIS   ################

x <- as.ordered(LETTERS[1:5])
basis <- rk(x)
plot(1:5, basis[,1], t = "l", ylim = extendrange(basis))
for(j in 2:ncol(basis)){
  lines(1:5, basis[,j], col = j)
}


################   LINEAR SPLINE BASIS   #########**######

x <- seq(0, 1, length.out = 101)
basis <- rk(x, m = 1)
plot(x, basis[,1], t = "l", ylim = extendrange(basis))
for(j in 2:ncol(basis)){
  lines(x, basis[,j], col = j)
}


################   CUBIC SPLINE BASIS   #########**######

x <- seq(0, 1, length.out = 101)
basis <- rk(x)
basis <- scale(basis)  # for visualization only!
plot(x, basis[,1], t = "l", ylim = extendrange(basis))
for(j in 2:ncol(basis)){
  lines(x, basis[,j], col = j)
}


################   QUINTIC SPLINE BASIS   ########**######

x <- seq(0, 1, length.out = 101)
basis <- rk(x, m = 3)
basis <- scale(basis)  # for visualization only!
plot(x, basis[,1], t = "l", ylim = extendrange(basis))
for(j in 2:ncol(basis)){
  lines(x, basis[,j], col = j)
}
```

`rk.model.matrix` *Construct Design Matrices via Reproducing Kernels*

Description

Creates a design (or model) matrix using the `rk` function to expand variables via a reproducing kernel basis.

Usage

```
rk.model.matrix(object, data = environment(object), ...)
```

Arguments

`object`	a `formula` or `terms` object describing the fit model
`data`	a data frame containing the variables referenced in `object`
`...`	additional arguments passed to the `rk` function, e.g., `df`, `knots`, `m`, etc. Arguments must be passed as a named list, see Examples.

Details

Designed to be a more flexible alternative to the `model.matrix` function. The `rk` function is used to construct a marginal basis for each variable that appears in the input `object`. Tensor product interactions are formed by taking a `row.kronecker` product of marginal basis matrices. Interactions of any order are supported using standard formulaic conventions, see Note.

Value

The design matrix corresponding to the input formula and data, which has the following attributes:

`assign`	an integer vector with an entry for each column in the matrix giving the term in the formula which gave rise to the column
`term.labels`	a character vector containing the labels for each of the terms in the model
`knots`	a named list giving the knots used for each variable in the formula
`m`	a named list giving the penalty order used for each variable in the formula
`periodic`	a named list giving the periodicity used for each variable in the formula
`xlev`	a named list giving the factor levels used for each variable in the formula

Note

For formulas of the form `y ~ x + z`, the constructed model matrix has the form `cbind(rk(x), rk(z))`, which simply concatenates the two marginal basis matrices. For formulas of the form `y ~ x : z`, the constructed model matrix has the form `row.kronecker(rk(x), rk(z))`, where `row.kronecker` denotes the row-wise kronecker product. The formula `y ~ x * z` is a shorthand for `y ~ x + z + x : z`, which concatenates the two previous results. Unless it is suppressed (using `0+`), the first column of the basis will be a column of ones named `(Intercept)`.

Author(s)

Nathaniel E. Helwig <helwig@umn.edu>

References

Helwig, N. E. (2017). Regression with ordered predictors via ordinal smoothing splines. Frontiers in Applied Mathematics and Statistics, 3(15), 1-13. doi:10.3389/fams.2017.00015

Helwig, N. E. (2021). Spectrally sparse nonparametric regression via elastic net regularized smoothers. Journal of Computational and Graphical Statistics, 30(1), 182-191. doi:10.1080/10618600.2020.1806855

Helwig, N. E. (2024). Precise tensor product smoothing via spectral splines. Stats, 7(1), 34-53. doi:10.3390/stats7010003

See Also

See rk for details on the reproducing kernel basis

Examples

```
# load auto data
data(auto)

# additive effects
x <- rk.model.matrix(mpg ~ ., data = auto)
dim(x)                      # check dimensions
attr(x, "assign")           # check group assignments
attr(x, "term.labels")      # check term labels

# two-way interactions
x <- rk.model.matrix(mpg ~ . * ., data = auto)
dim(x)                      # check dimensions
attr(x, "assign")           # check group assignments
attr(x, "term.labels")      # check term labels

# specify df for horsepower, weight, and acceleration
# note: default df = 5 is used for displacement and model.year
df <- list(horsepower = 6, weight = 7, acceleration = 8)
x <- rk.model.matrix(mpg ~ ., data = auto, df = df)
sapply(attr(x, "knots"), length)   # check df

# specify knots for model.year
# note: default knots are selected for other variables
knots <- list(model.year = c(1970, 1974, 1978, 1982))
x <- rk.model.matrix(mpg ~ ., data = auto, knots = knots)
sapply(attr(x, "knots"), length)   # check df
```

`row.kronecker` *Row-Wise Kronecker Product*

Description

Calculates the row-wise Kronecker product between two matrices with the same number of rows.

Usage

```
row.kronecker(X, Y)
```

Arguments

`X`	matrix of dimension $n \times p$
`Y`	matrix of dimension $n \times q$

Details

Given `X` of dimension `c(n, p)` and `Y` of dimension `c(n, q)`, this function returns

`cbind(x[,1] * Y, x[,2] * Y, ..., x[,p] * Y)`

which is a matrix of dimension `c(n, p*q)`

Value

Matrix of dimension $n \times pq$ where each row contains the Kronecker product between the corresponding rows of `X` and `Y`.

Author(s)

Nathaniel E. Helwig <helwig@umn.edu>

See Also

Used by the `rk.model.matrix` to construct basis functions for interaction terms

See `kronecker` for the regular kronecker product

Examples

```
X <- matrix(c(1, 1, 2, 2), nrow = 2, ncol = 2)
Y <- matrix(1:6, nrow = 2, ncol = 3)
row.kronecker(X, Y)
```

StartupMessage *Startup Message for grpnet*

Description

Prints the startup message when grpnet is loaded. Not intended to be called by the user.

Details

The 'grpnet' ascii start-up message was created using the taag software.

References

https://patorjk.com/software/taag/

visualize.penalty *Plots grpnet Penalty Function or its Derivative*

Description

Makes a plot or returns a data frame containing the group elastic net penalty (or its derivative) evaluated at a sequence of input values.

Usage

```
visualize.penalty(x = seq(-5, 5, length.out = 1001),
                  penalty = c("LASSO", "MCP", "SCAD"),
                  alpha = 1,
                  lambda = 1,
                  gamma = 4,
                  derivative = FALSE,
                  plot = TRUE,
                  subtitle = TRUE,
                  legend = TRUE,
                  location = ifelse(derivative, "bottom", "top"),
                  ...)
```

Arguments

x	sequence of values at which to evaluate the penalty.
penalty	which penalty or penalties should be plotted?
alpha	elastic net tuning parameter (between 0 and 1).
lambda	overall tuning parameter (non-negative).
gamma	additional hyperparameter for MCP (>1) or SCAD (>2).

`derivative`	if `FALSE` (default), then the penalty is plotted; otherwise the derivative of the penalty is plotted.
`plot`	if `TRUE` (default), then the result is plotted; otherwise the result is returned as a data frame.
`subtitle`	if `TRUE` (default), then the hyperparameter values are displayed in the subtitle.
`legend`	if `TRUE` (default), then a legend is included to distinguish the different `penalty` types.
`location`	the legend's location; ignored if `legend = FALSE`.
`...`	addition arguments passed to `plot` function, e.g., `xlim`, `ylim`, etc.

Details

The group elastic net penalty is defined as

$$P_{\alpha,\lambda}(\boldsymbol{\beta}) = Q_{\lambda_1}(\|\boldsymbol{\beta}\|) + \frac{\lambda_2}{2}\|\boldsymbol{\beta}\|^2$$

where $Q_\lambda()$ denotes the L1 penalty (LASSO, MCP, or SCAD), $\|\boldsymbol{\beta}\| = (\boldsymbol{\beta}^\top\boldsymbol{\beta})^{1/2}$ denotes the Euclidean norm, $\lambda_1 = \lambda\alpha$ is the L1 tuning parameter, and $\lambda_2 = \lambda(1-\alpha)$ is the L2 tuning parameter. Note that λ and α denote the `lambda` and `alpha` arguments.

Value

If `plot = TRUE`, then produces a plot.

If `plot = FALSE`, then returns a data frame.

Author(s)

Nathaniel E. Helwig <helwig@umn.edu>

References

Fan J, & Li, R. (2001). Variable selection via nonconcave penalized likelihood and its oracle properties. *Journal of the American Statistical Association, 96*(456), 1348-1360. doi:10.1198/016214501753382273

Helwig, N. E. (2024). Versatile descent algorithms for group regularization and variable selection in generalized linear models. *Journal of Computational and Graphical Statistics*. doi:10.1080/10618600.2024.2362232

Tibshirani, R. (1996). Regression and shrinkage via the Lasso. *Journal of the Royal Statistical Society, Series B, 58*, 267-288. doi:10.1111/j.25176161.1996.tb02080.x

Zhang CH (2010). Nearly unbiased variable selection under minimax concave penalty. *The Annals of Statistics, 38*(2), 894-942. doi:10.1214/09AOS729

See Also

`visualize.shrink` for plotting shrinkage operator

Examples

```
# plot penalty functions
visualize.penalty()

# plot penalty derivatives
visualize.penalty(derivative = TRUE)
```

`visualize.shrink` *Plots grpnet Shrinkage Operator or its Estimator*

Description

Makes a plot or returns a data frame containing the group elastic net shrinkage operator (or its estimator) evaluated at a sequence of input values.

Usage

```
visualize.shrink(x = seq(-5, 5, length.out = 1001),
                 penalty = c("LASSO", "MCP", "SCAD"),
                 alpha = 1,
                 lambda = 1,
                 gamma = 4,
                 fitted = FALSE,
                 plot = TRUE,
                 subtitle = TRUE,
                 legend = TRUE,
                 location = "top",
                 ...)
```

Arguments

`x`	sequence of values at which to evaluate the penalty.
`penalty`	which penalty or penalties should be plotted?
`alpha`	elastic net tuning parameter (between 0 and 1).
`lambda`	overall tuning parameter (non-negative).
`gamma`	additional hyperparameter for MCP (>1) or SCAD (>2).
`fitted`	if `FALSE` (default), then the shrinkage operator is plotted; otherwise the shrunken estimator is plotted.
`plot`	if `TRUE` (default), then the result is plotted; otherwise the result is returned as a data frame.
`subtitle`	if `TRUE` (default), then the hyperparameter values are displayed in the subtitle.
`legend`	if `TRUE` (default), then a legend is included to distinguish the different `penalty` types.
`location`	the legend's location; ignored if `legend = FALSE`.
`...`	addition arguments passed to `plot` function, e.g., `xlim`, `ylim`, etc.

Details

The updates for the group elastic net estimator have the form

$$\boldsymbol{\beta}_{\alpha,\lambda}^{(t+1)} = S_{\lambda_1,\lambda_2}(\|\mathbf{b}_{\alpha,\lambda}^{(t+1)}\|)\mathbf{b}_{\alpha,\lambda}^{(t+1)}$$

where $S_{\lambda_1,\lambda_2}(\cdot)$ is a shrinkage and selection operator, and

$$\mathbf{b}_{\alpha,\lambda}^{(t+1)} = \boldsymbol{\beta}_{\alpha,\lambda}^{(t)} + (\delta_{(t)}\epsilon)^{-1}\mathbf{g}^{(t)}$$

is the unpenalized update with $\mathbf{g}^{(t)}$ denoting the current gradient.

Note that $\lambda_1 = \lambda\alpha$ is the L1 tuning parameter, $\lambda_2 = \lambda(1-\alpha)$ is the L2 tuning parameter, $\delta_{(t)}$ is an upper-bound on the weights appearing in the Fisher information matrix, and ϵ is the largest eigenvalue of the Gramm matrix $n^{-1}\mathbf{X}^\top\mathbf{X}$.

Value

If `plot = TRUE`, then produces a plot.

If `plot = FALSE`, then returns a data frame.

Author(s)

Nathaniel E. Helwig <helwig@umn.edu>

References

Fan J, & Li, R. (2001). Variable selection via nonconcave penalized likelihood and its oracle properties. *Journal of the American Statistical Association, 96*(456), 1348-1360. doi:10.1198/016214501753382273

Helwig, N. E. (2024). Versatile descent algorithms for group regularization and variable selection in generalized linear models. *Journal of Computational and Graphical Statistics*. doi:10.1080/10618600.2024.2362232

Tibshirani, R. (1996). Regression and shrinkage via the Lasso. *Journal of the Royal Statistical Society, Series B, 58*, 267-288. doi:10.1111/j.25176161.1996.tb02080.x

Zhang CH (2010). Nearly unbiased variable selection under minimax concave penalty. *The Annals of Statistics, 38*(2), 894-942. doi:10.1214/09AOS729

See Also

`visualize.penalty` for plotting penalty function

Examples

```
# plot shrinkage operator
visualize.shrink()

# plot shrunken estimator
visualize.shrink(fitted = TRUE)
```

Index