

Package ‘grpnet’

February 21, 2024

Type Package

Title Group Elastic Net Regularized GLMs and GAMs

Version 0.3

Date 2024-02-20

Description Efficient algorithms for fitting generalized linear and additive models with group elastic net penalties. Implements group lasso, group MCP, and group SCAD with an optional group ridge penalty. Computes the regularization path for linear regression (gaussian), 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 2024-02-20 23:00:02 UTC

R topics documented:

auto	2
coef	3
cv.grpnet	5
grpnet	11
plot.cv.grpnet	19
plot.grpnet	21
predict.cv.grpnet	22
predict.grpnet	27
print	33

rk	35
rk.model.matrix	37
row.kronecker	39
StartupMessage	40

Index	41
--------------	-----------

auto	<i>Auto MPG Data Set</i>
------	--------------------------

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.min", "lambda.1se"),
      ...)

## 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 lambda that minimizes the cross-validation loss function. For "grpnet" objects, default uses <code>s = object\$lambda</code> . Interpolation is used for s values that are not included in <code>object\$lambda</code> .
...	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 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

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.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),
          type.measure = NULL,
          nfolds = 10,
          foldid = NULL,
          same.lambda = FALSE,
          parallel = FALSE,
          cluster = NULL,
          verbose = interactive(),
          ...)

```

```
## 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),
  type.measure = NULL,
  nfolds = 10,
  foldid = NULL,
  same.lambda = FALSE,
  parallel = FALSE,
  cluster = NULL,
  verbose = interactive(),
  ...)
```

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 <code>...</code> 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 <code>foldid</code> is used to compute the cross-validation error for each α , and (b) the solution for the optimal α 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 coercion, the <code>nfolds</code> argument is defined as <code>nfolds = nlevels(foldid)</code> .

same.lambda	Logical specifying 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.
...	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 $\beta_{\lambda_{[u]}}$ 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

$$E_u(\lambda) = C(\beta_{\lambda_{[u]}}, \mathbf{D}_u)$$

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

$$C(\beta_{\lambda_{[u]}}, \mathbf{D}_u) = \|\mathbf{y}_u - \mathbf{X}_u \beta_{\lambda_{[u]}}\|^2$$

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 `cvstd` 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
alpha	input alpha vector and min(cvm) for each alpha (if alpha is a vector)

Note

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

- Breheny, P., & Huang, J. (2015). Group descent algorithms for nonconvex penalized linear and logistic regression models with grouped predictors. *Statistics and Computing*, 25, 173-187. doi:10.1007/s1122201394242
- 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
- Yang, Y., & Zou, H. (2015). A fast unified algorithm for solving group-lasso penalize learning problems. *Statistics and Computing*, 25, 1129-1141. doi:10.1007/s1122201494985

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)

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

# print min and 1se solution info
mod

# plot cv error curve
plot(mod)

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

# load data
data(auto)

# define response (1 = American, 0 = other)
y <- ifelse(auto$origin == "American", 1, 0)

# define predictors
x <- rk.model.matrix(~ 0 + ., data = auto[,1:7])

# define group
g <- attr(x, "assign")

# 10-fold cv (default method, response = y)
set.seed(1)
mod <- cv.grpnet(x, y, g, family = "binomial", alpha = 1)

# 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", alpha = 1)
```

```
# 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",
                 alpha = 1, theta = 100)

# 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)
set.seed(1)
mod <- cv.grpnet(origin ~ ., data = auto, family = "multinomial", alpha = 1)

# 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", alpha = 1)
```

```

# 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", alpha = 1)

# print min and 1se solution info
mod

# plot cv error curve
plot(mod)

```

 grpnet

Fit a Group Elastic Net Regularized GLM/GAM

Description

Fits generalized linear/additive models with a group elastic net penalty. 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", "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 = ifelse(penalty == "MCP", 3, 4),
theta = 1,
standardize = TRUE,
orthogonalize = FALSE,
intercept = TRUE,
thresh = 1e-04,
maxit = 1e05,
...)

## S3 method for class 'formula'
grpnet(formula,
  data,
  use.rk = TRUE,
  family = c("gaussian", "binomial", "multinomial", "poisson",
            "negative.binomial", "Gamma", "inverse.gaussian"),
  weights = NULL,
  offset = NULL,
  alpha = 1,
  nlambdas = 100,
  lambda.min.ratio = ifelse(nobs < nvars, 0.05, 0.0001),
  lambda = NULL,
  penalty.factor = NULL,
  penalty = c("LASSO", "MCP", "SCAD"),
  gamma = ifelse(penalty == "MCP", 3, 4),
  theta = 1,
  standardize = TRUE,
  orthogonalize = FALSE,
  thresh = 1e-04,
  maxit = 1e05,
  ...)

```

Arguments

<code>x</code>	Model (design) matrix of dimension <code>nobs</code> by <code>nvars</code> ($n \times p$).
<code>y</code>	Response vector of length n . Matrix inputs are allowed for binomial and multinomial families (see "Binomial and multinomial" section).
<code>group</code>	Group label vector (factor, character, or integer) of length p . Predictors with the same label are grouped together for regularization.
<code>formula</code>	Model formula: a symbolic description of the model to be fitted. Uses the same syntax as <code>lm</code> and <code>glm</code> .
<code>data</code>	Optional data frame containing the variables referenced in <code>formula</code> .
<code>use.rk</code>	If TRUE (default), the <code>rk.model.matrix</code> function is used to build the model matrix. Otherwise, the <code>model.matrix</code> function is used to build the model matrix.

	Additional arguments to the <code>rk.model.matrix</code> function can be passed via the <code>...</code> argument.
<code>family</code>	Character specifying the assumed distribution for the response variable. Partial matching is allowed. Options include "gaussian" (real-valued response), "binomial" (binary response), "multinomial" (multi-class response), "poisson" (count response), "negative.binomial" (count response), "Gamma" (positive real-valued), or "inverse.gaussian" (positive real-valued).
<code>weights</code>	Optional vector of length n with non-negative weights to use for weighted (penalized) likelihood estimation. Defaults to a vector of ones.
<code>offset</code>	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.
<code>alpha</code>	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.
<code>nlambda</code>	Number of λ values to use in the regularization path. Ignored if <code>lambda</code> is provided.
<code>lambda.min.ratio</code>	The proportion $0 < \pi < 1$ that defines the minimum regularization parameter λ_{\min} as a fraction of the maximum regularization parameter λ_{\max} via the relationship $\lambda_{\min} = \pi \lambda_{\max}$. Ignored if <code>lambda</code> is provided. Note that λ_{\max} is defined such that all penalized effects are shrunk to zero.
<code>lambda</code>	Optional vector of user-supplied regularization parameter values.
<code>penalty.factor</code>	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 <code>levels(as.factor(group))</code> . Defaults to $\sqrt{p_k}$ for all $k = 1, \dots, K$, where p_k is the number of coefficients in the k -th group. If <code>penalty.factor[k] = 0</code> , 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 <code>penalty.factor</code> will be assigned the default penalty weight of $\sqrt{p_k}$. If <code>penalty.factor\$z = 0</code> , then the variable <code>z</code> is unpenalized and always included in the model.
<code>penalty</code>	Character specifying which (group) penalty to use: LASSO, MCP, or SCAD.
<code>gamma</code>	Penalty hyperparameter that satisfies $\gamma > 1$ for MCP and $\gamma > 2$ for SCAD. Ignored for LASSO penalty.
<code>theta</code>	Additional ("size") parameter for negative binomial responses, where the variance function is defined as $V(\mu) = \mu + \mu^2/\theta$
<code>standardize</code>	Logical indicating whether the predictors should be groupwise standardized. If TRUE (default), each column of <code>x</code> 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.

orthogonalize	Logical indicating whether the predictors should be groupwise orthogonalized. If TRUE, each predictor group's design matrix is orthonormalized (i.e., $\mathbf{X}_k^\top \mathbf{X}_k = \mathbf{I}_k$) 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.
...	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 $\mathbf{X} = (\mathbf{X}_1, \dots, \mathbf{X}_K)$ is the grouped predictor vector, and $\boldsymbol{\beta} = (\boldsymbol{\beta}_1, \dots, \boldsymbol{\beta}_K)$ 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
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 polynomial 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): $\log\left(\frac{\pi}{1-\pi}\right) = \mathbf{X}^\top \boldsymbol{\beta}$
- * 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 to specify the total number of trials.

For "multinomial" responses, two different possibilities exist for the input response:

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 $j \in \{1, \dots, p\}$ 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

- Breheny, P., & Huang, J. (2015). Group descent algorithms for nonconvex penalized linear and logistic regression models with grouped predictors. *Statistics and Computing*, 25, 173-187. doi:10.1007/s1122201394242
- 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
- Yang, Y., & Zou, H. (2015). A fast unified algorithm for solving group-lasso penalize learning problems. *Statistics and Computing*, 25, 1129-1141. doi:10.1007/s1122201494985

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 = "binomial" #####*#####

# load data
data(auto)

# define response (1 = American, 0 = other)
y <- ifelse(auto$origin == "American", 1, 0)
```

```

# define predictors
x <- rk.model.matrix(~ 0 + ., data = auto[,1:7])

# define group
g <- attr(x, "assign")

# fit model (default method, response = y)
mod <- grpnet(x, y, g, family = "binomial")

# 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", theta = 100)

# print regularization path info
mod

# plot coefficient paths
plot(mod)

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

# load data

```

```

data(auto)

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

# 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 $-1 \cdot \log(\lambda)$ 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 <code>plot</code> 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

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 Coefficients for grpnet Fits***Description**

Creates a profile plot of the coefficient 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 = FALSE, 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 multinomial model, the coefficients for each response 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

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.min", "lambda.1se"),
        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). "terms" gives the predictions for each term (group) in the model (η_k). "class" gives predicted class labels (for "binomial" and "multinomial" families). "coefficients" returns the coefficients used for predictions. "nonzero" returns a list giving the indices of non-zero coefficients 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

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, alpha = 1)

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

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

# compare rmse for two solutions
sqrt(mean((auto$mpg - fit.min)^2))
sqrt(mean((auto$mpg - fit.1se)^2))

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

# load data
data(auto)

# define response (1 = American, 0 = other)
y <- ifelse(auto$origin == "American", 1, 0)

# define predictors
x <- rk.model.matrix(~ 0 + ., data = auto[,1:7])

# define group
g <- attr(x, "assign")

# 10-fold cv (default method, response = y)
set.seed(1)
mod <- cv.grpnet(x, y, g, family = "binomial", alpha = 1)

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

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



```

# compare rmse for two solutions
sqrt(mean((y - fit.min)^2))
sqrt(mean((y - fit.1se)^2))

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

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

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

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

# load data
data(auto)

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

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

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

# compare rmse for two solutions
sqrt(mean((auto$horsepower - fit.min)^2))
sqrt(mean((auto$horsepower - fit.1se)^2))

#####**##### 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",
                 alpha = 1, theta = 100)

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

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

```

```

# compare rmse for two solutions
sqrt(mean((auto$horsepower - fit.min)^2))
sqrt(mean((auto$horsepower - fit.1se)^2))

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

# load data
data(auto)

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

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

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

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

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

# load data
data(auto)

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

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

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

# compare rmse for two solutions
sqrt(mean((auto$mpg - fit.min)^2))
sqrt(mean((auto$mpg - fit.1se)^2))

##### 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", alpha = 1)

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

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

# compare rmse for two solutions
sqrt(mean((auto$mpg - fit.min)^2))
sqrt(mean((auto$mpg - fit.1se)^2))

```

predict.grpnet	<i>Predict Method for grpnet Fits</i>
----------------	---------------------------------------

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 <code>s = object\$lambda</code> . Interpolation is used for s values that are not included in <code>object\$lambda</code> .

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}_{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

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

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

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

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 $c(\text{newnobs}, \text{length}(s))$ if $\text{length}(s) > 1$
- * a vector of length newnobs if $\text{length}(s) == 1$

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

- * an array of dimension $c(\text{newnobs}, \text{length}(\text{object}\$ylev), \text{length}(s))$ if type $\in\{c("link", "response")\}$
- * a matrix of dimension $c(\text{newnobs}, \text{length}(s))$ if $\text{type} == "class"$

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

If $\text{type} == "terms"$ and $\text{family} != "multinomial"$, the output will be...

- * an array of dimension $c(\text{newnobs}, \text{nterms}, \text{length}(s))$ if $\text{length}(s) > 1$
- * a matrix of dimension $c(\text{newnobs}, \text{nterms})$ if $\text{length}(s) == 1$

If $\text{type} == "terms"$ and $\text{family} == "multinomial"$, the output will be a list of length $\text{length}(\text{object}\$ylev)$ where each element gives the terms for the corresponding response class.

If $\text{type} == "importance"$ and $\text{family} != "multinomial"$, the output will be...

- * a matrix of dimension $c(\text{nterms}, \text{length}(s))$ if $\text{length}(s) > 1$
- * a vector of length nterms if $\text{length}(s) == 1$

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

If $\text{type} == "coefficients"$, the output will be the same as that produced by [coef.gprnet](#).

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

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

If $\text{type} \in\{c("ncoefs", "ngroups")\}$, the output will be a vector of length $\text{length}(s)$ where each element is an integer.

If $\text{type} == "norm"$, the output will be a matrix of dimension $c(K, \text{length}(s))$, 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

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)

# define response (1 = American, 0 = other)
y <- ifelse(auto$origin == "American", 1, 0)

# define predictors
x <- model.matrix(~ ., data = auto[,1:7])[,,-1]

# define group (according to colnames(x))
g <- c(1, 2, 2, 2, 2, 3, 4, 5, 6, 7)
```

```

# fit model (default method, response = y)
mod <- grpnet(x, y, g, family = "binomial")

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

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

# compare misclassification rate for solutions
miss.path <- 1 - colMeans(y == fit.path)
miss.some <- 1 - colMeans(y == fit.some)
plot(log(mod$lambda), miss.path, cex = 0.5)
points(log(c(.15, .1, .05)), 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", theta = 100)

# 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 = "multinomial" #####**#####

# load data
data(auto)

# fit model (formula method, response = origin)
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 = "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(0.2, 0.1, 0.05))

# 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(0.2, 0.1, 0.05)), 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(0.2, 0.1, 0.05))

# 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(0.2, 0.1, 0.05)), 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

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

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 <code>length(unique(c(knots, Boundary.knots)))</code> .
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 <code>range(x)</code> .
warn.outside	if TRUE, a warning is provided when x values are outside of the <code>Boundary.knots</code>
periodic	should the spline basis functions be constrained to be periodic with respect to the <code>Boundary.knots</code> ?
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 positive 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:

<code>df</code>	degrees of freedom
<code>knots</code>	knots for spline basis
<code>m</code>	derivative penalty order
<code>intercept</code>	was an intercept included?
<code>Boundary.knots</code>	boundary points of x
<code>periodic</code>	is the basis periodic?
<code>xlev</code>	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 $m = \text{NULL}$), then an ordinal spline basis is used
- * If x is an integer or numeric (and $m = \text{NULL}$), 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](https://doi.org/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](https://doi.org/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](https://doi.org/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)
}

```

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., <code>df</code> , <code>knots</code> , <code>m</code> , 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 $y \sim x + z$, 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 $y \sim x : z$, the constructed model matrix has the form `row.kronecker(rk(x), rk(z))`, where [row.kronecker](#) denotes the row-wise kronecker product. The formula $y \sim x * z$ is a shorthand for $y \sim x + z + x : z$, 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/>

Index

- * **algebra**
 - row.kronecker, 39
- * **array**
 - row.kronecker, 39
- * **datasets**
 - auto, 2
- * **graphs**
 - plot.cv.grpnet, 19
 - plot.grpnet, 21
- * **print**
 - print, 33
- * **regression**
 - coef, 3
 - cv.grpnet, 5
 - grpnet, 11
 - predict.cv.grpnet, 22
 - predict.grpnet, 27
 - rk, 35
 - rk.model.matrix, 37
- * **smooth**
 - cv.grpnet, 5
 - grpnet, 11
 - rk, 35
 - rk.model.matrix, 37
- auto, 2
- bs, 35
- class, 36
- coef, 3
- coef.grpnet, 4, 29, 33, 34
- cv.grpnet, 4, 5, 8, 17, 20, 22, 24, 30, 33, 34
- factor, 35
- family, 16
- formula, 38
- glm, 6, 12, 16
- grpnet, 4–8, 11, 16, 17, 22–24, 30, 33, 34
- grpnetStartupMessage (StartupMessage), 40
- I, 16
- kronecker, 40
- lm, 6, 12
- model.matrix, 6, 12, 38
- plot, 20, 21
- plot.cv.grpnet, 8, 19, 22
- plot.grpnet, 17, 20, 21
- poly, 15
- predict.cv.grpnet, 4, 8, 22, 30
- predict.grpnet, 4, 17, 23, 24, 27
- print, 33, 33
- print.coef.grpnet, 4
- rk, 35, 37–39
- rk.model.matrix, 6, 12, 13, 37, 40
- row.kronecker, 38, 39
- StartupMessage, 40
- terms, 38