Package ‘sparseGAM’

May 31, 2021

Type Package
Title Sparse Generalized Additive Models
Version 1.0
Date 2021-05-29
Author Ray Bai
Maintainer Ray Bai <raybaistat@gmail.com>
Description Fits sparse frequentist GAMs (SF-GAM) for continuous and discrete responses in the exponential dispersion family with the group lasso, group smoothly clipped absolute deviation (SCAD), and group minimax concave (MCP) penalties <doi:10.1007/s11222-013-9424-2>. Also fits sparse Bayesian generalized additive models (SB-GAM) with the spike-and-slab group lasso (SSGL) penalty of Bai et al. (2021) <doi:10.1080/01621459.2020.1765784>. B-spline basis functions are used to model the sparse additive functions. Stand-alone functions for group-regularized negative binomial regression, group-regularized gamma regression, and group-regularized regression in the exponential dispersion family with the SSGL penalty are also provided.
License GPL-3
Depends R (>= 3.6.0)
Imports stats, splines, MASS, pracma, grpreg
NeedsCompilation yes
Repository CRAN
Date/Publication 2021-05-31 10:10:02 UTC

R topics documented:

cv.grpreg.gamma ................................................................. 2
cv.grpreg.nb ................................................................. 4
cv.SBGAM ................................................................. 5
cv.SFGAM ................................................................. 7
cv.SSGL ................................................................. 9
gpreg.gamma ............................................................. 12
gpreg.nb ................................................................. 14
SBGAM ................................................................. 17
cv.grpreg.gamma

Cross-validation for Group-regularized Gamma Regression

Description

This function implements $K$-fold cross-validation for group-regularized gamma regression with a known shape parameter $\nu$ and the log link. For a description of group-regularized gamma regression, see the description for the grpreg.gamma function.

Our implementation is based on the least squares approximation approach of Wang and Leng (2007), and hence, the function does not allow the total number of covariates $p$ to be greater than \( \frac{K-1}{K} \times \text{sample size} \), where $K$ is the number of folds.

Usage

cv.grpreg.gamma(y, X, groups, gamma.shape=1, penalty=c("gLASSO","gSCAD","gMCP"), nfolds=10, weights, taper, nlambda=100, lambda, max.iter=10000, tol=1e-4)

Arguments

- **y**: $n \times 1$ vector of responses.
- **X**: $n \times p$ design matrix, where the $j$th column of $X$ corresponds to the $j$th overall covariate.
- **groups**: $p$-dimensional vector of group labels. The $j$th entry in `groups` should contain either the group number or the name of the factor level that the $j$th covariate belongs to. `groups` must be either a vector of integers or factors.
- **gamma.shape**: known shape parameter $\nu$ in $\text{Gamma}(\mu_i, \nu)$ distribution for the responses. Default is `gamma.shape=1`.
- **penalty**: group regularization method to use on the groups of coefficients. The options are "gLASSO", "gSCAD", and "gMCP". To implement cross-validation for gamma regression with the SSGL penalty, use the `cv.SSGL` function.
- **nfolds**: number of folds $K$ to use in $K$-fold cross-validation. Default is `nfolds=10`.
- **weights**: group-specific, nonnegative weights for the penalty. Default is to use the square roots of the group sizes.
- **taper**: tapering term $\gamma$ in group SCAD and group MCP controlling how rapidly the penalty tapers off. Default is `taper=4` for group SCAD and `taper=3` for group MCP. Ignored if "gLASSO" is specified as the penalty.
- **nlambda**: number of regularization parameters $L$. Default is `nlambda=100`.
- **lambda**: grid of $L$ regularization parameters. The user may specify either a scalar or a vector. If the user does not provide this, the program chooses the grid automatically.
cv.grpreg.gamma

max.iter maximum number of iterations in the algorithm. Default is max.iter=10000.

tol convergence threshold for algorithm. Default is tol=1e-4.

Value

The function returns a list containing the following components:

- **lambda**: $L \times 1$ vector of regularization parameters $\lambda$ used to fit the model. $\lambda$ is displayed in descending order.
- **cve**: $L \times 1$ vector of mean cross-validation error across all $K$ folds. The $k$th entry in cve corresponds to the $k$th regularization parameter in $\lambda$.
- **cvse**: $L \times 1$ vector of standard errors for cross-validation error across all $K$ folds. The $k$th entry in cvse corresponds to the $k$th regularization parameter in $\lambda$.
- **lambda.min**: value of $\lambda$ that minimizes mean cross-validation error cve.

References


Examples

```r
## Generate data
set.seed(12345)
X = matrix(runif(100*11), nrow=100)
n = dim(X)[1]
groups = c(1,1,1,2,2,2,3,3,4,5,5)
true.beta = c(-1,1,1,0,0,0,0,0,0,1.5,-1.5)
## Generate responses from gamma regression with known shape parameter 1
eta = crossprod(t(X), true.beta)
shape = 1
y = rgamma(n, rate=shape/exp(eta), shape=shape)
## 10-fold cross-validation for group-regularized gamma regression
## with the group LASSO penalty
gamma.cv = cv.grpreg.gamma(y, X, groups, penalty="gLASSO")
## Plot cross-validation curve
plot(gamma.cv$lambda, gamma.cv$cve, type="l", xlab="lambda", ylab="CVE")
## lambda which minimizes mean CVE
gamma.cv$lambda.min
```
Cross-validation for Group-regularized Negative Binomial Regression

Description

This function implements $K$-fold cross-validation for group-regularized negative binomial regression with a known size parameter $\alpha$ and the log link. For a description of group-regularized negative binomial regression, see the description for the `grpreg.nb` function.

Our implementation is based on the least squares approximation approach of Wang and Leng (2007), and hence, the function does not allow the total number of covariates $p$ to be greater than $\frac{p}{K} \times$ sample size, where $K$ is the number of folds.

Usage

```r
cv.grpreg.nb(y, X, groups, nb.size=1, penalty=c("gLASSO","gSCAD","gMCP"), nfolds=10, weights, taper, nlambda=100, lambda, max.iter=10000, tol=1e-4)
```

Arguments

- **y**: $n \times 1$ vector of responses.
- **X**: $n \times p$ design matrix, where the $j$th column of $X$ corresponds to the $j$th overall covariate.
- **groups**: $p$-dimensional vector of group labels. The $j$th entry in `groups` should contain either the group number or the name of the factor level that the $j$th covariate belongs to. `groups` must be either a vector of integers or factors.
- **nb.size**: known size parameter $\alpha$ in $NB(\alpha, \mu_i)$ distribution for the responses. Default is `nb.size=1`.
- **penalty**: group regularization method to use on the groups of coefficients. The options are "gLASSO", "gSCAD", and "gMCP". To implement cross-validation for negative binomial regression with the SSGL penalty, use the `cv.SSGL` function.
- **nfolds**: number of folds $K$ to use in $K$-fold cross-validation. Default is `nfolds=10`.
- **weights**: group-specific, nonnegative weights for the penalty. Default is to use the square roots of the group sizes.
- **taper**: tapering term $\gamma$ in group SCAD and group MCP controlling how rapidly the penalty tapers off. Default is `taper=4` for group SCAD and `taper=3` for group MCP. Ignored if "gLASSO" is specified as the penalty.
- **nlambda**: number of regularization parameters $L$. Default is `nlambda=100`.
- **lambda**: grid of $L$ regularization parameters. The user may specify either a scalar or a vector. If the user does not provide this, the program chooses the grid automatically.
- **max.iter**: maximum number of iterations in the algorithm. Default is `max.iter=10000`.
- **tol**: convergence threshold for algorithm. Default is `tol=1e-4`. 
Value

The function returns a list containing the following components:

- **lambda**: $L \times 1$ vector of regularization parameters used to fit the model. $\lambda$ is displayed in descending order.
- **cve**: $L \times 1$ vector of mean cross-validation error across all $K$ folds. The $k$th entry in `cve` corresponds to the $k$th regularization parameter in `lambda`.
- **cvse**: $L \times 1$ vector of standard errors for cross-validation error across all $K$ folds. The $k$th entry in `cvse` corresponds to the $k$th regularization parameter in `lambda`.
- **lambda.min**: value of $\lambda$ that minimizes mean cross-validation error `cve`.

References


Examples

```r
## Generate data
set.seed(1234)
X = matrix(runif(100*16), nrow=100)
n = dim(X)[1]
groups = c(1,1,1,2,2,2,3,4,5,5,6,7,8,8,8)
true.beta = c(-2,2,2,0,0,0,0,0,0,1.5,-1.5,0,0,-2,2,2)

## Generate count responses from negative binomial regression
eta = crossprod(t(X), true.beta)
y = rnbinom(n,size=1, mu=exp(eta))

## 10-fold cross-validation for group-regularized negative binomial
## regression with the group SCAD penalty
nb.cv = cv.grpreg.nb(y,X,groups,penalty="gMCP")

## Plot cross-validation curve
plot(nb.cv$lambda, nb.cv$cve, type="l", xlab="lambda", ylab="CVE")
## lambda which minimizes mean CVE
nb.cv$lambda.min
```

Description

This function implements $K$-fold cross-validation for sparse Bayesian generalized additive models (GAMs) with the spike-and-slab group lasso (SSGL) penalty. The identity link function is used for Gaussian GAMs, the logit link is used for binomial GAMs, and the log link is used for Poisson, negative binomial, and gamma GAMs.
Usage

cv.SBGAM(y, X, df=6, 
family=c("gaussian","binomial","poisson","negativebinomial","gamma"),
nb.size=1, gamma.shape=1, nfolds=5, nlambda0=20, lambda0, lambda1, 
a, b, max.iter=100, tol = 1e-6, print.fold=TRUE)

Arguments

y  \ n \times 1 \text{ vector of responses.}
X  \ n \times p \text{ design matrix, where the } j \text{th column of } X \text{ corresponds to the } j \text{th overall covariate.}
df number of B-spline basis functions to use in each basis expansion. Default is df=6, but the user may specify degrees of freedom as any integer greater than or equal to 3.
family exponential dispersion family. Allows for "gaussian", "binomial", "poisson", "negativebinomial", and "gamma". Note that for "negativebinomial", the size parameter must be specified, while for "gamma", the shape parameter must be specified.
nb.size known size parameter $\alpha$ in $NB(\alpha,\mu_i)$ distribution for negative binomial responses. Default is nb.size=1. Ignored if family is not "negativebinomial".
gamma.shape known shape parameter $\nu$ in $Gamma(\mu_i,\nu)$ distribution for gamma responses. Default is gamma.shape=1. Ignored if family is not "gamma".
nfolds number of folds $K$ to use in $K$-fold cross-validation. Default is nfolds=5.
nlambda0 number of spike hyperparameter $L$. Default is nlambda0=20.
lambda0 grid of $L$ spike hyperparameters $\lambda_0$. The user may specify either a scalar or a vector. If the user does not provide this, the program chooses the grid automatically.
lambda1 slab hyperparameter $\lambda_1$ in the SSGL prior. Default is lambda1=1.
a shape hyperparameter for the $Beta(a,b)$ prior on the mixing proportion in the SSGL prior. Default is a=1.
b shape hyperparameter for the $Beta(a,b)$ prior on the mixing proportion in the SSGL prior. Default is b=dim(X)[2].
max.iter maximum number of iterations in the algorithm. Default is max.iter=100.
tol convergence threshold for algorithm. Default is tol=1e-6.
print.fold Boolean variable for whether or not to print the current fold in the algorithm. Default is print.fold=TRUE.

Value

The function returns a list containing the following components:

lambda0  \ L \times 1 \text{ vector of spike hyperparameters } \lambda_0 \text{ used to fit the model. } \lambda_0 \text{ is displayed in descending order.}
cve  \ L \times 1 \text{ vector of mean cross-validation error across all } K \text{ folds. The } k \text{th entry in cve corresponds to the } k \text{th regularization parameter in } \lambda_0 \text{.}
cv.SFGAM

References


Examples

```r
## Generate data
set.seed(12345)
X = matrix(runif(30*3), nrow=30)
n = dim(X)[1]
y = 2.5*sin(pi*X[,1]) + rnorm(n)

## K-fold cross-validation for 4 degrees of freedom and 4 values of lambda0
## Note that if user does not specify lambda0, cv.SFGAM chooses a grid automatically.
cv.mod = cv.SFGAM(y, X, df=4, family="gaussian", lambda0=seq(from=25,to=5,by=-10))

cv.mod$lambda0.min
```

Description

This function implements $K$-fold cross-validation for sparse frequentist generalized additive models (GAMs) with the group LASSO, group SCAD, and group MCP penalties. The identity link function is used for Gaussian GAMs, the logit link is used for binomial GAMs, and the log link is used for Poisson, negative binomial, and gamma GAMs.

Usage

```r
cv.SFGAM(y, X, df=6,
    family=c("gaussian","binomial","poisson","negativebinomial","gamma"),
    nb.size=1, gamma.shape=1, penalty=c("gLASSO","gMCP","gSCAD"), taper,
    nfolds=10, nlambda=100, lambda, max.iter=10000, tol=1e-4)
```
Arguments

- `y` \( n \times 1 \) vector of responses.
- `X` \( n \times p \) design matrix, where the \( j \)th column of \( X \) corresponds to the \( j \)th overall covariate.
- `df` number of B-spline basis functions to use in each basis expansion. Default is \( df=6 \), but the user may specify degrees of freedom as any integer greater than or equal to 3.
- `family` exponential dispersion family. Allows for "gaussian", "binomial", "poisson", "negativebinomial", and "gamma". Note that for "negativebinomial", the size parameter must be specified, while for "gamma", the shape parameter must be specified.
- `nb.size` known size parameter \( \alpha \) in \( NB(\alpha, \mu_i) \) distribution for negative binomial responses. Default is `nb.size`=1. Ignored if `family` is not "negativebinomial".
- `gamma.shape` known shape parameter \( \nu \) in \( Gamma(\mu_i, \nu) \) distribution for gamma responses. Default is `gamma.shape`=1. Ignored if `family` is not "gamma".
- `penalty` group regularization method to use on the groups of basis coefficients. The options are "gLASSO", "gSCAD", and "gMCP". To implement sparse GAMs with the SSGL penalty, use the SBGAM function.
- `taper` tapering term \( \gamma \) in group SCAD and group MCP controlling how rapidly the penalty tapers off. Default is `taper`=4 for group SCAD and `taper`=3 for group MCP. Ignored if "gLASSO" is specified as the penalty.
- `nfolds` number of folds \( K \) to use in \( K \)-fold cross-validation. Default is `nfolds`=10.
- `nlambda` number of regularization parameters \( L \). Default is `nlambda`=100.
- `lambda` grid of \( L \) regularization parameters. The user may specify either a scalar or a vector. If the user does not provide this, the program chooses the grid automatically.
- `max.iter` maximum number of iterations in the algorithm. Default is `max.iter`=10000.
- `tol` convergence threshold for algorithm. Default is `tol`=1e-4.

Value

The function returns a list containing the following components:

- `lambda` \( L \times 1 \) vector of regularization parameters \( \lambda \) used to fit the model. \( \lambda \) is displayed in descending order.
- `cve` \( L \times 1 \) vector of mean cross-validation error across all \( K \) folds. The \( k \)th entry in `cve` corresponds to the \( k \)th regularization parameter in `lambda`.
- `cvse` \( L \times 1 \) vector of standard errors for cross-validation error across all \( K \) folds. The \( k \)th entry in `cvse` corresponds to the \( k \)th regularization parameter in `lambda`.
- `lambda.min` value of lambda that minimizes mean cross-validation error `cve`.
References


Examples

```r
## Generate data
set.seed(12345)
X = matrix(runif(100*20), nrow=100)
n = dim(X)[1]
y = 5*sin(2*pi*X[,1])-5*cos(2*pi*X[,2]) + rnorm(n)

## Test data with 50 observations
X.test = matrix(runif(50*20), nrow=50)

## Fit sparse Gaussian generalized additive model to data with the MCP penalty
gam.mod = SFGAM(y, X, X.test, family="gaussian", penalty="gMCP")

## The model corresponding to the 75th tuning parameter
(fit <- gam.mod$lambda[75])
(gam.mod$classifications[,75]) ## The covariate index is listed first

## Plot first function f_1(x_1) in 75th model
x1 = X.test[,1]
(f1.hat = gam.mod$f.pred[,75])

## Extract estimates of f_1
f1.hat = f1.hat[,1]

## Plot X_1 against f_1(x_1)
plot(x1[order(x1)], f1.hat[order(x1)], xlab=expression(x[1]), ylab=expression(f_1(x_1)))
```

Cross-Validation for Spike-and-Slab Group Lasso Regression

**Description**

This function implements *K*-fold cross-validation for group-regularized regression in the exponential dispersion family with the spike-and-slab group lasso (SSGL) penalty. The identity link function is used for Gaussian regression, the logit link is used for binomial regression, and the log link is used for Poisson, negative binomial, and gamma regression.
Usage

cv.SSGL(y, X, groups,
   family=c("gaussian","binomial","poisson","negativebinomial","gamma"),
   nb.size=1, gamma.shape=1, weights, nfolds=5, nlambda0=20,
   lambda0, lambda1, a, b, max.iter=100, tol=1e-6, print.fold=TRUE)

Arguments

y  \( n \times 1 \) vector of responses.
X  \( n \times p \) design matrix, where the \( j \)th column of \( X \) corresponds to the \( j \)th overall covariate.
grids  \( p \)-dimensional vector of group labels. The \( j \)th entry in \( \text{groups} \) should contain either the group number or the name of the factor level that the \( j \)th covariate belongs to. \( \text{groups} \) must be either a vector of integers or factors.
family  exponential dispersion family. Allows for "gaussian", "binomial", "poisson", "negativebinomial", and "gamma". Note that for "negativebinomial", the size parameter must be specified, while for "gamma", the shape parameter must be specified.

nb.size  known size parameter \( \alpha \) in \( NB(\alpha, \mu_i) \) distribution for negative binomial responses. Default is \( \text{nb.size}=1 \). Ignored if \( \text{family} \) is not "negativebinomial".
gamma.shape  known shape parameter \( \nu \) in \( \Gamma(\mu_i, \nu) \) distribution for gamma responses. Default is \( \text{gamma.shape}=1 \). Ignored if \( \text{family} \) is not "gamma".
weights  group-specific, nonnegative weights for the penalty. Default is to use the square roots of the group sizes.
nfolds  number of folds \( K \) to use in \( K \)-fold cross-validation. Default is \( \text{nfolds}=5 \).
nlambda0  number of spike hyperparameters \( L \). Default is \( \text{nlambda0}=20 \).
lambda0  grid of \( L \) spike hyperparameters \( \lambda_0 \). The user may specify either a scalar or a vector. If the user does not provide this, the program chooses the grid automatically.
lambda1  slab hyperparameter \( \lambda_1 \) in the SSGL prior. Default is \( \text{lambda1}=1 \).
a  shape hyperparameter for the \( \text{Beta}(a, b) \) prior on the mixing proportion in the SSGL prior. Default is \( a=1 \).
b  shape hyperparameter for the \( \text{Beta}(a, b) \) prior on the mixing proportion in the SSGL prior. Default is \( b=\text{dim}(X)[2] \).
max.iter  maximum number of iterations in the algorithm. Default is \( \text{max.iter}=100 \).
tol  convergence threshold for algorithm. Default is \( \text{tol}=1e-6 \).
print.fold  Boolean variable for whether or not to print the current fold in the algorithm. Default is \( \text{print.fold}=\text{TRUE} \).

Value

The function returns a list containing the following components:

lambda0  \( L \times 1 \) vector of spike hyperparameters \( \lambda_0 \) used to fit the model. \( \lambda_0 \) is displayed in descending order.
cv.SSGL

\[ cve \] \( L \times 1 \) vector of mean cross-validation error across all \( K \) folds. The \( k \)th entry in \( cve \) corresponds to the \( k \)th regularization parameter in \( \lambda_0 \).

\[ cvse \] \( L \times 1 \) vector of standard errors for cross-validation error across all \( K \) folds. The \( k \)th entry in \( cvse \) corresponds to the \( k \)th regularization parameter in \( \lambda_0 \).

\[ \lambda_0.min \] value of \( \lambda_0 \) that minimizes mean cross-validation error \( cve \).

References


Examples

```r
## Generate data
set.seed(12345)
X = matrix(runif(30*6), nrow=30)
n = dim(X)[1]
groups = c(1,1,1,2,2,3)
true.beta = c(-1.5,0.5,-1.5,0,0,0)

## Generate responses from Gaussian distribution
y = crossprod(t(X), true.beta) + rnorm(n)

## K-fold cross-validation for 3 choices of \( \lambda_0 \)
## Note that if user does not specify \( \lambda_0 \), cv.SSGL chooses a grid automatically.
ssgl.mods = cv.SSGL(y, X, groups, family="gaussian", lambda0=seq(from=10,to=2,by=-4))

## Plot cross-validation curve
plot(ssgl.mods$lambda0, ssgl.mods$cve, type="l", xlab="lambda0", ylab="CVE")

## lambda which minimizes mean CVE
ssgl.mods$lambda0.min

## Example with Poisson regression

## Generate count responses
eta = crossprod(t(X), true.beta)
y = rpois(n,exp(eta))

## K-fold cross-validation with 4 choices of \( \lambda_0 \)
## Note that if user does not specify \( \lambda_0 \), cv.SSGL chooses a grid automatically.
ssgl.poisson.mods = cv.SSGL(y, X, groups, family="poisson", lambda0=seq(from=8,to=2,by=-2))

## Plot cross-validation curve
plot(ssgl.poisson.mods$lambda0, ssgl.poisson.mods$cve, type="l", xlab="lambda0", ylab="CVE")

## lambda which minimizes mean CVE
ssgl.poisson.mods$lambda0.min
```

## Generate data
set.seed(12345)
X = matrix(runif(30*6), nrow=30)
n = dim(X)[1]
groups = c(1,1,1,2,2,3)
true.beta = c(-1.5,0.5,-1.5,0,0,0)

## Generate responses from Gaussian distribution
y = crossprod(t(X), true.beta) + rnorm(n)

## K-fold cross-validation for 3 choices of \( \lambda_0 \)
## Note that if user does not specify \( \lambda_0 \), cv.SSGL chooses a grid automatically.
ssgl.mods = cv.SSGL(y, X, groups, family="gaussian", lambda0=seq(from=10,to=2,by=-4))

## Plot cross-validation curve
plot(ssgl.mods$lambda0, ssgl.mods$cve, type="l", xlab="lambda0", ylab="CVE")

## lambda which minimizes mean CVE
ssgl.mods$lambda0.min

## Example with Poisson regression

## Generate count responses
eta = crossprod(t(X), true.beta)
y = rpois(n,exp(eta))

## K-fold cross-validation with 4 choices of \( \lambda_0 \)
## Note that if user does not specify \( \lambda_0 \), cv.SSGL chooses a grid automatically.
ssgl.poisson.mods = cv.SSGL(y, X, groups, family="poisson", lambda0=seq(from=8,to=2,by=-2))

## Plot cross-validation curve
plot(ssgl.poisson.mods$lambda0, ssgl.poisson.mods$cve, type="l", xlab="lambda0", ylab="CVE")

## lambda which minimizes mean CVE
**Description**

This function implements group-regularized gamma regression with a known shape parameter $\nu$ and the log link. In gamma regression, we assume that $y_i \sim Gamma(\mu_i, \nu)$, where

$$f(y_i | \mu_i, \nu) = \frac{1}{\Gamma(\nu)} \left( \frac{\nu}{\mu_i} \right)^\nu \exp\left( -\frac{\nu}{\mu_i} y_i \right) y_i^{\nu - 1}, y > 0.$$  

Then $E(y_i) = \mu_i$, and we relate $\mu_i$ to a set of $p$ covariates $x_i$ through the log link,

$$\log(\mu_i) = \beta_0 + x_i^T \beta, i = 1, ..., n.$$  

If the covariates in each $x_i$ are grouped according to known groups $g = 1, ..., G$, then this function may estimate some of the $G$ groups of coefficients as all zero, depending on the amount of regularization.

Our implementation for regularized gamma regression is based on the least squares approximation approach of Wang and Leng (2007), and hence, the function does not allow the total number of covariates $p$ to be greater than sample size.

**Usage**

```r
grpreg.gamma(y, X, X.test, groups, gamma.shape=1,
             penalty=c("gLASSO","gSCAD","gMCP"),
             weights, taper, nlambda=100, lambda, max.iter=10000, tol=1e-4)
```

**Arguments**

- **y**: $n \times 1$ vector of responses for training data.
- **X**: $n \times p$ design matrix for training data, where the $j$th column of $X$ corresponds to the $j$th overall covariate.
- **X.test**: $n_{test} \times p$ design matrix for test data to calculate predictions. $X.test$ must have the same number of columns as $X$, but not necessarily the same number of rows. If no test data is provided or if in-sample predictions are desired, then the function automatically sets $X.test=X$ in order to calculate in-sample predictions.
- **groups**: $p$-dimensional vector of group labels. The $j$th entry in groups should contain either the group number or the name of the factor level that the $j$th covariate belongs to. groups must be either a vector of integers or factors.
- **gamma.shape**: known shape parameter $\nu$ in $Gamma(\mu_i, \nu)$ distribution for the responses. Default is gamma.shape=1.
**Value**

The function returns a list containing the following components:

- **lambda**: $L \times 1$ vector of regularization parameters $\lambda$ used to fit the model. $\lambda$ is displayed in descending order.
- **beta0**: $L \times 1$ vector of estimated intercepts. The $k$th entry in beta0 corresponds to the $k$th regularization parameter in lambda.
- **beta**: $p \times L$ matrix of estimated regression coefficients. The $k$th column in beta corresponds to the $k$th regularization parameter in lambda.
- **mu.pred**: $n_{test} \times L$ matrix of predicted mean response values $\mu_{test} = E(Y_{test})$ based on the test data in $X_{test}$ (or training data $X$ if no argument was specified for $X_{test}$). The $k$th column in mu.pred corresponds to the predictions for the $k$th regularization parameter in lambda.
- **classifications**: $G \times L$ matrix of classifications, where $G$ is the number of groups. An entry of "1" indicates that the group was classified as nonzero, and an entry of "0" indicates that the group was classified as zero. The $k$th column of classifications corresponds to the $k$th regularization parameter in lambda.
- **loss**: $L \times 1$ vector of negative log-likelihood of the fitted models. The $k$th entry in loss corresponds to the $k$th regularization parameter in lambda.

**References**


Examples

```r
## Generate data
set.seed(12345)
X = matrix(runif(100*11), nrow=100)
n = dim(X)[1]
groups = c("a","a","a","b","b","b","c","c","d","e","e")
groups = as.factor(groups)
true.beta = c(-1,1,1,0,0,0,0,0,0,1.5,-1.5)

## Generate responses from gamma regression with known shape parameter 1
eta = crossprod(t(X), true.beta)
shape = 1
y = rgamma(n, rate=shape/exp(eta), shape=shape)

## Generate test data
n.test = 50
X.test = matrix(runif(n.test*11), nrow=n.test)

## Fit gamma regression models with the group LASSO penalty
gamma.mod = grpreg.gamma(y, X, X.test, groups, penalty="gLASSO")

## Tuning parameters used to fit models
gamma.mod$lambda

# Predicted n.test-dimensional vectors mu=E(Y.test) based on test data, X.test.
# The kth column of 'mu.pred' corresponds to the kth entry in 'lambda.'
gamma.mod$mu.pred

# Classifications of the 5 groups. The kth column of 'classifications'
# corresponds to the kth entry in 'lambda.'
gamma.mod$classifications
```

---

**grpreg.nb**  
*Group-regularized Negative Binomial Regression*

**Description**

This function implements group-regularized negative binomial regression with a known size parameter $\alpha$ and the log link. In negative binomial regression, we assume that $y_i \sim NB(\alpha, \mu_i)$, where

$$f(y_i|\alpha, \mu_i) = \frac{\Gamma(y + \alpha)}{y!\Gamma(\alpha)} \left( \frac{\mu_i}{\mu_i + \alpha} \right)^y \left( \frac{\alpha}{\mu_i + \alpha} \right)^\alpha, y = 0, 1, 2, \ldots$$

Then $E(y_i) = \mu_i$, and we relate $\mu_i$ to a set of $p$ covariates $x_i$ through the log link,

$$\log(\mu_i) = \beta_0 + x_i^T \beta, i = 1, \ldots, n$$
If the covariates in each $x_i$ are grouped according to known groups $g = 1, \ldots, G$, then this function may estimate some of the $G$ groups of coefficients as all zero, depending on the amount of regularization.

Our implementation for regularized negative binomial regression is based on the least squares approximation approach of Wang and Leng (2007), and hence, the function does not allow the total number of covariates $p$ to be greater than sample size.

Usage

```r
grpreg.nb(y, X, X.test, groups, nb.size=1, penalty=c("gLASSO","gSCAD","gMCP"),
weights, taper, nlambda=100, lambda, max.iter=10000, tol=1e-4)
```

Arguments

- **y**: $n \times 1$ vector of responses for training data.
- **X**: $n \times p$ design matrix for training data, where the $j$th column of $X$ corresponds to the $j$th overall covariate.
- **X.test**: $n_{test} \times p$ design matrix for test data to calculate predictions. $X.test$ must have the same number of columns as $X$, but not necessarily the same number of rows. If no test data is provided or if in-sample predictions are desired, then the function automatically sets $X.test=X$ in order to calculate in-sample predictions.
- **groups**: $p$-dimensional vector of group labels. The $j$th entry in `groups` should contain either the group number or the name of the factor level that the $j$th covariate belongs to. `groups` must be either a vector of integers or factors.
- **nb.size**: known size parameter $\alpha$ in $NB(\alpha, \mu_i)$ distribution for the responses. Default is `nb.size=1`.
- **penalty**: group regularization method to use on the groups of coefficients. The options are "gLASSO", "gSCAD", "gMCP". To implement negative binomial regression with the SSGL penalty, use the SSGL function.
- **weights**: group-specific, nonnegative weights for the penalty. Default is to use the square roots of the group sizes.
- **taper**: tapering term $\gamma$ in group SCAD and group MCP controlling how rapidly the penalty tapers off. Default is `taper=4` for group SCAD and `taper=3` for group MCP. Ignored if "gLASSO" is specified as the penalty.
- **nlambda**: number of regularization parameters $L$. Default is `nlambda=100`.
- **lambda**: grid of $L$ regularization parameters. The user may specify either a scalar or a vector. If the user does not provide this, the program chooses the grid automatically.
- **max.iter**: maximum number of iterations in the algorithm. Default is `max.iter=10000`.
- **tol**: convergence threshold for algorithm. Default is `tol=1e-4`.

Value

The function returns a list containing the following components:
\text{lambda} \quad L \times 1 \text{ vector of regularization parameters \lambda used to fit the model. \lambda is displayed in descending order.}

\text{beta0} \quad L \times 1 \text{ vector of estimated intercepts. The } k\text{th entry in beta0 corresponds to the } k\text{th regularization parameter in lambda.}

\text{beta} \quad p \times L \text{ matrix of estimated regression coefficients. The } k\text{th column in beta corresponds to the } k\text{th regularization parameter in lambda.}

\text{mu.pred} \quad n_{test} \times L \text{ matrix of predicted mean response values } \mu_{test} = E(Y_{test}) \text{ based on the test data in } X_{test} \text{ (or training data } X\text{ if no argument was specified for } X_{test}). \text{ The } k\text{th column in mu.pred corresponds to the predictions for the } k\text{th regularization parameter in lambda.}

\text{classifications} \quad G \times L \text{ matrix of classifications, where } G \text{ is the number of groups. An entry of } 1 \text{ indicates that the group was classified as nonzero, and an entry of } 0 \text{ indicates that the group was classified as zero. The } k\text{th column of classifications corresponds to the } k\text{th regularization parameter in lambda.}

\text{loss} \quad L \times 1 \text{ vector of negative log-likelihood of the fitted models. The } k\text{th entry in loss corresponds to the } k\text{th regularization parameter in lambda.}

\text{References}


\text{Examples}

```r
## Generate training data
set.seed(1234)
X = matrix(runif(100*16), nrow=100)
n = dim(X)[1]
groups = c("A","A","A","B","B","B","C","C","D","E","E","F","G","H","H","H")
groups = as.factor(groups)
true.beta = c(-2,2,2,0,0,0,0,0,0,1.5,-1.5,0,0,-2,2,2)

## Generate count responses from negative binomial regression
eta = crossprod(t(X), true.beta)
y = rnbinom(n,size=1, mu=exp(eta))

## Generate test data
n.test = 50
X.test = matrix(runif(n.test*16), nrow=n.test)

## Fit negative binomial regression models with the group SCAD penalty
nb.mod = grpreg.nb(y, X, X.test, groups, penalty="gSCAD")

## Tuning parameters used to fit models
nb.mod$lambda
```
# Predicted n.test-dimensional vectors mu=E(Y.test) based on test data, X.test.
# The kth column of 'mu.pred' corresponds to the kth entry in 'lambda.'
nb.mod$mu.pred

# Classifications of the 8 groups. The kth column of 'classifications'
# corresponds to the kth entry in lambda.
nb.mod$classifications

---

**Sparse Bayesian Generalized Additive Models**

**Description**

This function implements sparse Bayesian generalized additive models (GAMs) with the spike-and-slab group lasso (SSGL) penalty. Let $y_i$ denote the $i$th response and $x_i$ denote a $p$-dimensional vector of covariates. GAMs are of the form,

$$ g(E(y_i)) = \beta_0 + \sum_{j=1}^{p} f_j(x_{ij}), i = 1, ..., n, $$

where $g$ is a monotone increasing link function. The identity link function is used for Gaussian regression, the logit link is used for binomial regression, and the log link is used for Poisson, negative binomial, and gamma regression. With the SSGL penalty, some of the univariate functions $f_j(x_j)$ will be estimated as $\hat{f}_j(x_j) = 0$, depending on the size of the spike hyperparameter $\lambda_0$ in the SSGL prior. The functions $f_j(x_j), j = 1, ..., p,$ are modeled using B-spline basis expansions.

There is another implementation of sparse Gaussian GAMs with the SSGL penalty available at https://github.com/jantonelli111/SSGL, which uses natural cubic splines as the basis functions. This package **sparseGAM** uses B-spline basis functions and also implements sparse GAMs with the SSGL penalty for binomial, Poisson, negative binomial, and gamma regression.

For implementation of sparse **frequentist** GAMs with the group LASSO, group SCAD, and group MCP penalties, use the **SFGAM** function.

**Usage**

```r
SBGAM(y, X, X.test, df=6, 
    family=c("gaussian","binomial","poisson","negativebinomial","gamma"), 
    nb.size=1, gamma.shape=1, nlambda0=20, lambda0, lambda1, a, b, 
    max.iter=100, tol = 1e-6, print.iter=TRUE)
```

**Arguments**

- **y**
  - $n \times 1$ vector of responses for training data.

- **X**
  - $n \times p$ design matrix for training data, where the $j$th column of $X$ corresponds to the $j$th overall covariate.
X.test \( n_{test} \times p \) design matrix for test data to calculate predictions. \( X.test \) must have the same number of columns as \( X \), but not necessarily the same number of rows. If no test data is provided or if in-sample predictions are desired, then the function automatically sets \( X.test=X \) in order to calculate in-sample predictions.

df number of B-spline basis functions to use in each basis expansion. Default is df=6, but the user may specify degrees of freedom as any integer greater than or equal to 3.

family exponential dispersion family. Allows for "gaussian", "binomial", "poisson", "negativebinomial", and "gamma". Note that for "negativebinomial", the size parameter must be specified, while for "gamma", the shape parameter must be specified.

nb.size known size parameter \( \alpha \) in NB(\( \alpha, \mu_i \)) distribution for negative binomial responses. Default is nb.size=1. Ignored if family is not "negativebinomial".

gamma.shape known shape parameter \( \nu \) in Gamma(\( \mu_i, \nu \)) distribution for gamma responses. Default is gamma.shape=1. Ignored if family is not "gamma".

nlambda0 number of spike hyperparameter \( L \). Default is nlambda0=20.

lambda0 grid of \( L \) spike hyperparameters \( \lambda_0 \). The user may specify either a scalar or a vector. If the user does not provide this, the program chooses the grid automatically.

lambda1 slab hyperparameter \( \lambda_1 \) in the SSGL prior. Default is lambda1=1.

a shape hyperparameter for the Beta(\(a,b\)) prior on the mixing proportion in the SSGL prior. Default is a=1.

b shape hyperparameter for the Beta(\(a,b\)) prior on the mixing proportion in the SSGL prior. Default is b=dim(X)[2].

max.iter maximum number of iterations in the algorithm. Default is max.iter=100.

tol convergence threshold for algorithm. Default is tol=1e-6.

print.iter Boolean variable for whether or not to print the current nlambda0 in the algorithm. Default is print.iter=TRUE.

Value

The function returns a list containing the following components:

lambda0 \( L \times 1 \) vector of spike hyperparameters lambda0 used to fit the model. lambda0 is displayed in descending order.

f.pred List of \( L \ n_{test} \times p \) matrices, where the \( k \)th matrix in the list corresponds to the \( k \)th spike hyperparameter in lambda0. The \( j \)th column in each matrix in f.pred is the estimate of the \( j \)th function evaluated on the test data in X.test for the \( j \)th covariate (or training data \( X \) if \( X.test \) was not specified).

mu.pred \( n_{test} \times L \) matrix of predicted mean response values \( \mu_{test} = E(Y_{test}) \) based on the test data in X.test (or training data \( X \) if no argument was specified for X.test). The \( k \)th column in mu.pred corresponds to the predictions for the \( k \)th spike hyperparameter in lambda0.
classifications

$p \times L$ matrix of classifications. An entry of "1" indicates that the corresponding function was classified as nonzero, and an entry of "0" indicates that the function was classified as zero. The $k$th column of classifications corresponds to the $k$th spike hyperparameter in $\lambda_0$.

beta0

$L \times 1$ vector of estimated intercepts. The $k$th entry in beta0 corresponds to the $k$th spike hyperparameter in $\lambda_0$.

beta

dp $\times L$ matrix of estimated basis coefficients. The $k$th column in beta corresponds to the $k$th spike hyperparameter in $\lambda_0$.

loss

vector of either the residual sum of squares ("gaussian") or the negative log-likelihood ("binomial", "poisson", "negativebinomial", "gamma") of the fitted model. The $k$th entry in loss corresponds to the $k$th spike hyperparameter in $\lambda_0$.

References


Examples

## Generate data
set.seed(12345)
X = matrix(runif(100*5), nrow=100)
n = dim(X)[1]
y = 3*sin(2*pi*X[,1])-3*cos(2*pi*X[,2]) + rnorm(n)

## Test data with 30 observations
X.test = matrix(runif(30*5), nrow=30)

## Fit sparse Bayesian generalized additive model to data with the SSGL penalty
## and 5 spike hyperparameters
SBGAM.mod = SBGAM(y, X, X.test, family="gaussian", lambda0=seq(from=50,to=10,by=-10))

## The model corresponding to the 1st spike hyperparameter
SBGAM.mod$lambda[1]
SBGAM.mod$classifications[,1]

## Plot first function f_1(x_1) in 2nd model
x1 = X.test[,1]
## Estimates of all 20 function evaluations on test data
f.hat = SBGAM.mod$f.pred[,1]
## Extract estimates of f_1
f1.hat = f.hat[,1]

## Plot X_1 against f_1(x_1)
plot(x1[order(x1)], f1.hat[order(x1)], xlab=expression(x[1]),
Description

This function implements sparse frequentist generalized additive models (GAMs) with the group LASSO, group SCAD, and group MCP penalties. Let $y_i$ denote the $i$th response and $x_i$ denote a $p$-dimensional vector of covariates. GAMs are of the form,

$$g(E(y_i)) = \beta_0 + \sum_{j=1}^{p} f_j(x_{ij}), \quad i = 1, ..., n,$$

where $g$ is a monotone increasing link function. The identity link function is used for Gaussian regression, the logit link is used for binomial regression, and the log link is used for Poisson, negative binomial, and gamma regression. The univariate functions are estimated using linear combinations of B-spline basis functions. Under group regularization of the basis coefficients, some of the univariate functions $f_j(x_j)$ will be estimated as $\hat{f}_j(x_j) = 0$, depending on the size of the regularization parameter $\lambda$.

For implementation of sparse Bayesian GAMs with the SSGL penalty, use the SBGAM function.

Usage

```r
SFGAM(y, X, X.test, df=6, family=c("gaussian","binomial","poisson","negativebinomial","gamma"), nb.size=1, gamma.shape=1, penalty=c("gLASSO","gMCP","gSCAD"), taper, nlambda=100, lambda, max.iter=10000, tol=1e-4)
```

Arguments

- `y` : $n \times 1$ vector of responses for training data.
- `X` : $n \times p$ design matrix for training data, where the $j$th column of $X$ corresponds to the $j$th overall covariate.
- `X.test` : $n_{test} \times p$ design matrix for test data to calculate predictions. $X.test$ must have the same number of columns as $X$, but not necessarily the same number of rows. If no test data is provided or if in-sample predictions are desired, then the function automatically sets $X.test=X$ in order to calculate in-sample predictions.
- `df` : number of B-spline basis functions to use in each basis expansion. Default is df=6, but the user may specify degrees of freedom as any integer greater than or equal to 3.
- `family` : exponential dispersion family. Allows for "gaussian", "binomial", "poisson", "negativebinomial", and "gamma". Note that for "negativebinomial", the size parameter must be specified, while for "gamma", the shape parameter must be specified.
nb.size: known size parameter \( \alpha \) in \( NB(\alpha, \mu_i) \) distribution for negative binomial responses. Default is nb.size=1. Ignored if family is not "negativebinomial".

gamma.shape: known shape parameter \( \nu \) in \( Gamma(\mu_i, \nu) \) distribution for gamma responses. Default is gamma.shape=1. Ignored if family is not "gamma".

penalty: group regularization method to use on the groups of basis coefficients. The options are "gLASSO", "gSCAD", and "gMCP". To implement sparse GAMs with the SSGL penalty, use the SBGAM function.

taper: tapering term \( \gamma \) in group SCAD and group MCP controlling how rapidly the penalty tapers off. Default is taper=4 for group SCAD and taper=3 for group MCP. Ignored if "gLASSO" is specified as the penalty.

nlambda: number of regularization parameters \( L \). Default is nlambda=100.

lambda: grid of \( L \) regularization parameters. The user may specify either a scalar or a vector. If the user does not provide this, the program chooses the grid automatically.

max.iter: maximum number of iterations in the algorithm. Default is max.iter=10000.

tol: convergence threshold for algorithm. Default is tol=1e-4.

Value

The function returns a list containing the following components:

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lambda</td>
<td>( L \times 1 ) vector of regularization parameters lambda used to fit the model. lambda is displayed in descending order.</td>
</tr>
<tr>
<td>f.pred</td>
<td>List of ( L \times n_{\text{test}} \times p ) matrices, where the ( k )th matrix in the list corresponds to the ( k )th regularization parameter in lambda. The ( j )th column in each matrix in f.pred is the estimate of the ( j )th function evaluated on the test data in ( X.\text{test} ) for the ( j )th covariate (or training data ( X ) if ( X.\text{test} ) was not specified).</td>
</tr>
<tr>
<td>mu.pred</td>
<td>( n_{\text{test}} \times L ) matrix of predicted mean response values ( \mu_{\text{test}} = E(Y_{\text{test}}) ) based on the test data in ( X.\text{test} ) (or training data ( X ) if no argument was specified for ( X.\text{test} )). The ( k )th column in mu.pred corresponds to the predictions for the ( k )th regularization parameter in lambda.</td>
</tr>
<tr>
<td>classifications</td>
<td>( p \times L ) matrix of classifications. An entry of &quot;1&quot; indicates that the corresponding function was classified as nonzero, and an entry of &quot;0&quot; indicates that the function was classified as zero. The ( k )th column of classifications corresponds to the ( k )th regularization parameter in lambda.</td>
</tr>
<tr>
<td>beta0</td>
<td>( L \times 1 ) vector of estimated intercepts. The ( k )th entry in beta0 corresponds to the ( k )th regularization parameter in lambda.</td>
</tr>
<tr>
<td>beta</td>
<td>( d_p \times L ) matrix of estimated basis coefficients. The ( k )th column in beta corresponds to the ( k )th regularization parameter in lambda.</td>
</tr>
<tr>
<td>loss</td>
<td>vector of either the residual sum of squares (&quot;gaussian&quot;) or the negative log-likelihood (&quot;binomial&quot;, &quot;poisson&quot;, &quot;negativebinomial&quot;, &quot;gamma&quot;) of the fitted model. The ( k )th entry in loss corresponds to the ( k )th regularization parameter in lambda.</td>
</tr>
</tbody>
</table>
References


Examples

```r
## Generate data
set.seed(12345)
X = matrix(runif(100*20), nrow=100)
n = dim(X)[1]
y = 5*sin(2*pi*X[,1])-5*cos(2*pi*X[,2]) + rnorm(n)
## Test data with 50 observations
X.test = matrix(runif(50*20), nrow=50)
## K-fold cross-validation with group MCP penalty
cv.mod = cv.SFGAM(y, X, family="gaussian", penalty="gMCP")
## Plot CVE curve
plot(cv.mod$lambda, cv.mod$cve, type="l", xlab="lambda", ylab="CVE")
## lambda which minimizes cross-validation error
lambda.opt = cv.mod$lambda.min
## Fit a single model with lambda.opt
SFGAM.mod = SFGAM(y, X, X.test, penalty="gMCP", lambda=lambda.opt)
## Classifications
SFGAM.mod$classifications
## Predicted function evaluations on test data
f.pred = SFGAM.mod$f.pred
## Plot estimated first function
x1 = X.test[,1]
f1.hat = f.pred[,1]
## Plot x_1 against f_1(x_1)
plot(x1[order(x1)], f1.hat[order(x1)], xlab=expression(x[1]), ylab=expression(f[1](x[1])))
```

SSGL

*Spike-and-Slab Group Lasso Regression*
Description

This is a stand-alone function for group-regularized regression models in the exponential dispersion family with the spike-and-slab group lasso (SSGL) penalty. Let $y_i$ denote the $i$th response and $x_i$ denote a $p$-dimensional vector of covariates. We fit models of the form,

$$ g(E(y_i)) = \beta_0 + x_i^T \beta, i = 1, \ldots, n, $$

where $g$ is a monotone increasing link function. The identity link function is used for Gaussian regression, the logit link is used for binomial regression, and the log link is used for Poisson, negative binomial, and gamma regression.

If the covariates in each $x_i$ are grouped according to known groups $g = 1, \ldots, G$, then this function may estimate some of the $G$ groups of coefficients as all zero, depending on the amount of regularization.

Another implementation of the SSGL model for Gaussian regression models is available on Github at https://github.com/jantonelli111/SSGL. This package sparseGAM also implements the SSGL model for binomial, Poisson, negative binomial, and gamma regression.

Usage

```r
SSGL(y, X, X.test, groups,
     family=c("gaussian","binomial","poisson","negativebinomial","gamma"),
     nb.size=1, gamma.shape=1, weights, nlambda0=20, lambda0, lambda1, a, b,
     max.iter=100, tol = 1e-6, print.iter=TRUE)
```

Arguments

- **y**: $n \times 1$ vector of responses for training data.
- **X**: $n \times p$ design matrix for training data, where the $j$th column of $X$ corresponds to the $j$th overall covariate.
- **X.test**: $n_{test} \times p$ design matrix for test data to calculate predictions. $X.test$ must have the same number of columns as $X$, but not necessarily the same number of rows. If no test data is provided or if in-sample predictions are desired, then the function automatically sets $X.test=X$ in order to calculate in-sample predictions.
- **groups**: $p$-dimensional vector of group labels. The $j$th entry in groups should contain either the group number or the name of the factor level that the $j$th covariate belongs to. groups must be either a vector of integers or factors.
- **family**: exponential dispersion family. Allows for "gaussian", "binomial", "poisson", "negativebinomial", and "gamma". Note that for "negativebinomial", the size parameter must be specified, while for "gamma", the shape parameter must be specified.
- **nb.size**: known size parameter $\alpha$ in $NB(\alpha, \mu_i)$ distribution for the negative binomial responses. Default is nb.size=1. Ignored if family is not "negativebinomial".
- **gamma.shape**: known shape parameter $\nu$ in $Gamma(\mu_i, \nu)$ distribution for gamma responses. Default is gamma.shape=1. Ignored if family is not "gamma".
- **weights**: group-specific, nonnegative weights for the penalty. Default is to use the square roots of the group sizes.
number of spike hyperparameters $L$. Default is $n_{\lambda_0}=20$.

$\lambda_0$ grid of $L$ spike hyperparameters $\lambda_0$. The user may specify either a scalar or a vector. If the user does not provide this, the program chooses the grid automatically.

$\lambda_1$ slab hyperparameter $\lambda_1$ in the SSGL prior. Default is $\lambda_1=1$.

$a$ shape hyperparameter for the $Beta(a,b)$ prior on the mixing proportion in the SSGL prior. Default is $a=1$.

$b$ shape hyperparameter for the $Beta(a,b)$ prior on the mixing proportion in the SSGL prior. Default is $b=\text{dim}(X)[2]$.

$max.iter$ maximum number of iterations in the algorithm. Default is $max.iter=100$.

$tol$ convergence threshold for algorithm. Default is $tol=1e-6$.

$\text{print.iter}$ Boolean variable for whether or not to print the current $n_{\lambda_0}$ in the algorithm. Default is $\text{print.iter}=\text{TRUE}$.

Value

The function returns a list containing the following components:

- $\lambda_0$ $L \times 1$ vector of spike hyperparameters $\lambda_0$ used to fit the model. $\lambda_0$ is displayed in descending order.

- $\beta_0$ $L \times 1$ vector of estimated intercepts. The $k$th entry in $\beta_0$ corresponds to the $k$th spike hyperparameter in $\lambda_0$.

- $\beta$ $p \times L$ matrix of estimated regression coefficients. The $k$th column in $\beta$ corresponds to the $k$th spike hyperparameter in $\lambda_0$.

- $\mu_{\text{pred}}$ $n_{\text{test}} \times L$ matrix of predicted mean response values $\mu_{\text{test}} = E(Y_{\text{test}})$ based on the test data in $X_{\text{test}}$ (or training data $X$ if no argument was specified for $X_{\text{test}}$). The $k$th column in $\mu_{\text{pred}}$ corresponds to the predictions for the $k$th spike hyperparameter in $\lambda_0$.

- $\text{classifications}$ $G \times L$ matrix of classifications, where $G$ is the number of groups. An entry of "1" indicates that the group was classified as nonzero, and an entry of "0" indicates that the group was classified as zero. The $k$th column of $\text{classifications}$ corresponds to the $k$th spike hyperparameter in $\lambda_0$.

- $\text{loss}$ vector of either the residual sum of squares ("gaussian") or the negative log-likelihood ("binomial", "poisson", "negativebinomial", "gamma") of the fitted model. The $k$th entry in $\text{loss}$ corresponds to the $k$th spike hyperparameter in $\lambda_0$.

References


Examples

```r
## Generate data
set.seed(12345)
X = matrix(runif(100*10), nrow=100)
n = dim(X)[1]
groups = c("A","A","A","B","B","B","C","C","D","D")
groups = as.factor(groups)
true.beta = c(-2.5,1.5,1.5,0,0,0,2,-2,0,0)

## Generate responses from Gaussian distribution
y = crossprod(t(X),true.beta) + rnorm(n)

## Generate test data
n.test = 50
X.test = matrix(runif(n.test*10), nrow=n.test)

## Fit SSGL model with 10 spike hyperparameters
## Note that if user does not specify lambda0, the SSGL function chooses a grid automatically.
SSGL.mod = SSGL(y, X, X.test, groups, family="gaussian", lambda0=seq(from=50,to=5,by=-5))

## Regression coefficient estimates
SSGL.mod$beta

# Predicted n.test-dimensional vectors mu=E(Y.test) based on test data, X.test.
# The kth column of 'mu.pred' corresponds to the kth entry in 'lambda.'
SSGL.mod$mu.pred

# Classifications of the 8 groups. The kth column of 'classifications'
# corresponds to the kth entry in 'lambda.'
SSGL.mod$classifications

## Example with binomial regression

## Generate binary responses
eta = crossprod(t(X), true.beta)
y = rbinom(n, size=1, prob=1/(1+exp(-eta)))

## Fit SSGL model with 10 spike hyperparameters
## Note that if user does not specify lambda0, the SSGL function chooses a grid automatically.
SSGL.mod = SSGL(y, X, X.test, groups, family="binomial",
lambda0=seq(from=10,to=1,by=-1))

## Predicted probabilities of success mu=E(Y.test) based on test data, X.test
SSGL.mod$mu.pred

## Classifications of the 8 groups.
SSGL.mod$classifications
```
Index

cv.grpreg.gamma, 2
.cv.grpreg.nb, 4
.cv.SBGAM, 5
.cv.SFGAM, 7
.cv.SSGL, 9

grpreg.gamma, 12
grpreg.nb, 14

SBGAM, 17
SFGAM, 20
SSGL, 22