Package 'SGPR'

January 20, 2025

Type Package

Title Sparse Group Penalized Regression for Bi-Level Variable Selection

Version 0.1.2

Description Fits the regularization path of regression models (linear and logistic) with additively combined penalty terms. All possible combinations with Least Absolute Shrinkage and Selection Operator (LASSO), Smoothly Clipped Absolute Deviation (SCAD), Minimax Concave Penalty (MCP) and Exponential Penalty (EP) are supported. This includes Sparse Group LASSO (SGL), Sparse Group SCAD (SGS), Sparse Group MCP (SGM) and Sparse Group EP (SGE). For more information, see Buch, G., Schulz, A., Schmidtmann, I., Strauch, K., & Wild, P. S. (2024) <doi:10.1002/bimj.202200334>.

License GPL (>= 3)

Imports Rcpp

LinkingTo Rcpp

Encoding UTF-8

RoxygenNote 7.3.1

NeedsCompilation yes

Author Gregor Buch [aut, cre, cph], Andreas Schulz [ths], Irene Schmidtmann [ths], Konstantin Strauch [ths], Philipp Wild [ths]

Maintainer Gregor Buch <buchgregor@gmail.com>

Repository CRAN

Date/Publication 2024-05-16 14:20:05 UTC

Contents

coef.sgp 2
coef.sgp.cv 3

get.loss 4
plot.sgp 4
plot.sgp.cv 5
predict.sgp 6
predict.sgp.cv 7
process.group 9
process.lambda 9
process.penalty 10
process.X 11
process.y 12
process.Z 12
sgp 13
sgp.cv 16

Index **19**

`coef.sgp` *Coefficients from an SGP model*

Description

A function that extracts the estimated coefficients from an SGP object.

Usage

```
## S3 method for class 'sgp'
coef(object, lambda, index = 1:length(object$lambda), drop = TRUE, ...)
```

Arguments

`object`	A object that was generated with sgp.
`lambda`	The value of lambda at which the coefficients are to be extracted.
`index`	The index that indicates the lambda at which the coefficients are to be extracted (alternative to specifying 'lambda').
`drop`	A Boolean value that specifies whether empty dimensions should be removed.
`...`	Other parameters of underlying basic functions.

Value

A vector or matrix with the estimated coefficients.

Examples

```
n <- 100
p <- 12
nr <- 4
g <- paste0("Group ",ceiling(1:p / nr))
X <- matrix(rnorm(n * p), n, p)
b <- c(-3:3)
y_lin <- X[, 1:length(b)] %*% b + 5 * rnorm(n)
y_log <- rbinom(n, 1, exp(y_lin) / (1 + exp(y_lin)))

lin_fit <- sgp(X, y_lin, g, type = "linear")
coef(lin_fit, index = 5:7)

log_fit <- sgp(X, y_log, g, type = "logit")
coef(log_fit, index = 5:7)
```

`coef.sgp.cv` *Coefficients from SGP models*

Description

A function that extracts the estimated coefficients from an cross-validated SGP object.

Usage

```
## S3 method for class 'sgp.cv'
coef(object, lambda = object$lambda.min, index = object$min, ...)
```

Arguments

`object`	A object that was generated with sgp.cv.
`lambda`	The value of lambda at which the coefficients are to be extracted.
`index`	The index that indicates the lambda at which the coefficients are to be extracted (alternative to specifying 'lambda').
`...`	Other parameters of underlying basic functions.

Value

A vector or matrix with the estimated coefficients.

Examples

```
n <- 100
p <- 12
nr <- 4
g <- paste0("Group ",ceiling(1:p / nr))
X <- matrix(rnorm(n * p), n, p)
```

```
b <- c(-3:3)
y_lin <- X[, 1:length(b)] %*% b + 5 * rnorm(n)
y_log <- rbinom(n, 1, exp(y_lin) / (1 + exp(y_lin)))

lin_fit <- sgp.cv(X, y_lin, g, type = "linear")
coef(lin_fit)

log_fit <- sgp.cv(X, y_log, g, type = "logit")
coef(log_fit)
```

`get.loss` *A function that calculates the loss/cost*

Description

A function that calculates the loss/cost

Usage

```
get.loss(y, pred, type)
```

Arguments

`y`	The response vector.
`pred`	The predicted values for the response.
`type`	A string indicating the type of regression model (linear or binomial).

Value

The loss of the input vectors.

`plot.sgp` *Plots the coefficient path of an SGP object*

Description

Produces a coefficient profile plot of the coefficient paths for a fitted SGP object

Usage

```
## S3 method for class 'sgp'
plot(x, alpha = 1, legend.pos, label = FALSE, log.l = FALSE, norm = FALSE, ...)
```

Arguments

x	A object that was generated with sgp.
alpha	Tuning parameter for the alpha-blending.
legend.pos	Coordinates or keyword for positioning the legend.
label	A Boolean value that specifies whether the plot should be annotated.
log.l	A Boolean value that specifies whether the horizontal axis should be on the log scale.
norm	A Boolean value that specifies whether the norm of each group should be plotted.
...	Other parameters of underlying basic functions.

Value

A plot object with the coefficient path of an SGP.

Examples

```
n <- 100
p <- 12
nr <- 4
g <- paste0("Group ",ceiling(1:p / nr))
X <- matrix(rnorm(n * p), n, p)
b <- c(-3:3)
y_lin <- X[, 1:length(b)] %*% b + 5 * rnorm(n)
y_log <- rbinom(n, 1, exp(y_lin) / (1 + exp(y_lin)))

lin_fit <- sgp(X, y_lin, g, type = "linear")
plot(lin_fit, legend.pos = "topright", label = TRUE)
plot(lin_fit,  label = TRUE, norm = TRUE)

log_fit <- sgp(X, y_log, g, type = "logit")
plot(log_fit, legend.pos = "topright", label = TRUE)
plot(log_fit, label = TRUE, norm = TRUE)
```

---

plot.sgp.cv *Plots the cross-validation curve from a SGP object*

Description

Plots the cross-validation curve as a function of the lambda values used.

Usage

```
## S3 method for class 'sgp.cv'
plot(x, log.l = TRUE, highlight = TRUE, col = "firebrick3", ...)
```

Arguments

x	A object that was generated with sgp.cv.
log.l	A Boolean value that specifies whether the horizontal axis should be on the log scale.
highlight	A Boolean value that specifies whether a vertical line should be added at the value where the cross-validation error is minimized.
col	Controls the color of the dots.
...	Other parameters of underlying basic functions.

Value

A plot object with the cross-validation curve of an SGP.

Examples

```
n <- 100
p <- 12
nr <- 4
g <- paste0("Group ",ceiling(1:p / nr))
X <- matrix(rnorm(n * p), n, p)
b <- c(-3:3)
y_lin <- X[, 1:length(b)] %*% b + 5 * rnorm(n)
y_log <- rbinom(n, 1, exp(y_lin) / (1 + exp(y_lin)))

lin_fit <- sgp.cv(X, y_lin, g, type = "linear")
plot(lin_fit, col = "blue")

log_fit <- sgp.cv(X, y_log, g, type = "logit")
plot(log_fit, col = "blue")
```

---

predict.sgp *Predictions based on a SGP model*

Description

A function that extracts information from a SGP object and performs predictions.

Usage

```
## S3 method for class 'sgp'
predict(
  object,
  X = NULL,
 extract = c("link", "response", "class", "coef", "vars", "groups", "nvars", "ngroups",
    "norm"),
  lambda,
```

```
  index = 1:length(object$lambda),
  ...
)
```

Arguments

object	A object that was generated with sgp.
X	The design matrix for making predictions.
extract	A string indicating the type of information to return.
lambda	The value of lambda at which predictions should be made.
index	The index that indicates the lambda at which predictions should be made (alternative to specifying 'lambda').
...	Other parameters of underlying basic functions.

Value

Different objects depending on the sting indicated by 'extract'.

Examples

```
n <- 100
p <- 12
nr <- 4
g <- paste0("Group ",ceiling(1:p / nr))
X <- matrix(rnorm(n * p), n, p)
b <- c(-3:3)
y_lin <- X[, 1:length(b)] %*% b + 5 * rnorm(n)
y_log <- rbinom(n, 1, exp(y_lin) / (1 + exp(y_lin)))

lin_fit <- sgp(X, y_lin, g, type = "linear")
predict(lin_fit, X = X, extract = "nvars")

log_fit <- sgp(X, y_log, g, type = "logit")
predict(log_fit, X = X, extract = "nvars")
```

---

predict.sgp.cv *Predictions based on a SGP models*

Description

A function that extracts information from a cross-validated SGP object and performs predictions.

Usage

```
## S3 method for class 'sgp.cv'
predict(
  object,
  X,
  lambda = object$lambda.min,
  index = object$min,
 extract = c("link", "response", "class", "coefficients", "vars", "groups", "nvars",
    "ngroups", "norm"),
  ...
)
```

Arguments

object	A object that was generated with sgp.cv.
X	The design matrix for making predictions.
lambda	The value of lambda at which predictions should be made.
index	The index that indicates the lambda at which predictions should be made (alternative to specifying 'lambda').
extract	A string indicating the type of information to return.
...	Other parameters of underlying basic functions.

Value

Different objects depending on the sting indicated by 'extract'.

Examples

```
n <- 100
p <- 12
nr <- 4
g <- paste0("Group ",ceiling(1:p / nr))
X <- matrix(rnorm(n * p), n, p)
b <- c(-3:3)
y_lin <- X[, 1:length(b)] %*% b + 5 * rnorm(n)
y_log <- rbinom(n, 1, exp(y_lin) / (1 + exp(y_lin)))

lin_fit <- sgp.cv(X, y_lin, g, type = "linear")
predict(lin_fit, X = X, extract = "link")

log_fit <- sgp.cv(X, y_log, g, type = "logit")
predict(log_fit, X = X, extract = "class")
```

process.group *Process groupings for a sparse group penalty*

Description

A function that checks the group information for possible errors and processes it.

Usage

```
process.group(group, group.weight)
```

Arguments

group	A vector that specifies the group membership of each variable in X.
group.weight	A vector specifying weights that are multiplied by the group penalty to account for different group sizes.

Value

A structure containing the prepared group structure and, as an attribute, its labels and group weights.

process.lambda *Set up a lambda sequence*

Description

A function that sets up a lambda sequence for a sparse group penalty.

Usage

```
process.lambda(
  X,
  y,
  group,
  Z,
  type,
  alpha,
  lambda.min,
  log.lambda,
  nlambda,
  group.weight,
  ada_mult
)
```

Arguments

X	The design matrix without intercept with the variables to be selected.
y	The response vector.
group	A vector indicating the group membership of each variable in X.
Z	The design matrix of the variables to be included in the model without penalization.
type	A string indicating the type of regression model (linear or binomial).
alpha	Tuning parameter for the mixture of penalties at group and variable level. A value of 0 results in a selection at group level, a value of 1 results in a selection at variable level and everything in between is bi-level selection.
lambda.min	An integer multiplied by the maximum lambda to define the end of the lambda sequence.
log.lambda	A Boolean value that specifies whether the values of the lambda sequence should be on the log scale.
nlambda	An integer that specifies the length of the lambda sequence.
group.weight	A vector specifying weights that are multiplied by the group penalty to account for different group sizes.
ada_mult	An integer that defines the multiplier for adjusting the convergence threshold.

Value

A vector with values for lambda.

---

process.penalty *Process the arguments about the sparse group penalty*

Description

A function that checks arguments about the penalty and translates them to integer (for the C++ code).

Usage

```
process.penalty(penalty, pvar, pgr, vargamma, grgamma, vartau, grtau, alpha)
```

Arguments

penalty	A string that specifies the sparse group penalty to be used.
pvar	A string that specifies the penalty used at the variable level.
pgr	A string that specifies the penalty used at the group level.
vargamma	An integer that defines the value of gamma for the penalty at the variable level.
grgamma	An integer that specifies the value of gamma for the penalty at the group level.

vartau	An integer that defines the value of tau for the penalty at the variable level.
grtau	An integer that specifies the value of tau for the penalty at the group level.
alpha	Tuning parameter for the mixture of penalties at group and variable level. A value of 0 results in a selection at group level, a value of 1 results in a selection at variable level and everything in between is bi-level selection.

Value

A list of two integers indicating the penalty for the C++ code.

---

process.X *Process X for a sparse group penalty*

Description

A function that checks the design matrix X for possible errors and scales it.

Usage

```
process.X(X, group)
```

Arguments

X	The design matrix without intercept with the variables to be selected.
group	A vector that specifies the group membership of each variable in X.

Value

A list containing:

X The standardized design matrix X.

vars The variable names of the matrix.

center The center of the variables before the transformation.

scale The scale of the variables before the transformation.

process.y *Process y for a sparse group penalty*

Description

A function that checks the response vector y for possible errors.

Usage

```
process.y(y, type)
```

Arguments

y	The response vector.
type	A string indicating the type of regression model (linear or binomial).

Value

The verified response vector y.

process.Z *Process Z for a sparse group penalty*

Description

A function that checks the design matrix Z for possible errors and scales it.

Usage

```
process.Z(Z)
```

Arguments

Z	The design matrix of the variables to be included in the model without penalization.

Value

A list containing:

Z The standardized design matrix Z.

vars The variable names of the matrix.

center The center of the variables before the transformation.

scale The scale of the variables before the transformation.

sgp *Fit a sparse group regularized regression path*

Description

A function that determines the regularization paths for models with sparse group penalties at a grid of values for the regularization parameter lambda.

Usage

```
sgp(
  X,
  y,
  group = 1:ncol(X),
  penalty = c("sgl", "sgs", "sgm", "sge"),
  alpha = 1/3,
  type = c("linear", "logit"),
  Z = NULL,
  nlambda = 100,
  lambda.min = {
     if (nrow(X) > ncol(X))
         1e-04
     else 0.05
 },
  log.lambda = TRUE,
  lambdas,
  prec = 1e-04,
  ada_mult = 2,
  max.iter = 10000,
  standardize = TRUE,
  vargamma = ifelse(pvar == "scad" | penalty == "sgs", 4, 3),
  grgamma = ifelse(pgr == "scad" | penalty == "sgs", 4, 3),
  vartau = 1,
  grtau = 1,
  pvar = c("lasso", "scad", "mcp", "exp"),
  pgr = c("lasso", "scad", "mcp", "exp"),
  group.weight = rep(1, length(unique(group))),
  returnX = FALSE,
  ...
)
```

Arguments

X	The design matrix without intercept with the variables to be selected.
y	The response vector.
group	A vector indicating the group membership of each variable in X.

penalty	A string that specifies the sparse group penalty to be used.
alpha	Tuning parameter for the mixture of penalties at group and variable level. A value of 0 results in a selection at group level, a value of 1 results in a selection at variable level and everything in between is bi-level selection.
type	A string indicating the type of regression model (linear or binomial).
Z	The design matrix of the variables to be included in the model without penalization.
nlambda	An integer that specifies the length of the lambda sequence.
lambda.min	An integer multiplied by the maximum lambda to define the end of the lambda sequence.
log.lambda	A Boolean value that specifies whether the values of the lambda sequence should be on the log scale.
lambdas	A user supplied vector with values for lambda.
prec	The convergence threshold for the algorithm.
ada_mult	An integer that defines the multiplier for adjusting the convergence threshold.
max.iter	The convergence threshold for the algorithm.
standardize	An integer that defines the multiplier for adjusting the convergence threshold.
vargamma	An integer that defines the value of gamma for the penalty at the variable level.
grgamma	An integer that specifies the value of gamma for the penalty at the group level.
vartau	An integer that defines the value of tau for the penalty at the variable level.
grtau	An integer that specifies the value of tau for the penalty at the group level.
pvar	A string that specifies the penalty used at the variable level.
pgr	A string that specifies the penalty used at the group level.
group.weight	A vector specifying weights that are multiplied by the group penalty to account for different group sizes.
returnX	A Boolean value that specifies whether standardized design matrix should be returned.
...	Other parameters of underlying basic functions.

Details

Two options are available for choosing a penalty. With the argument `penalty`, the methods Sparse Group LASSO, Sparse Group SCAD, Sparse Group MCP and Sparse Group EP can be selected with the abbreviations `sgl`, `sgs`, `sgm` and `sge`. Alternatively, penalties can be combined additively with the arguments `pvar` and `pgr`, where `pvar` is the penalty applied at the variable level and `pgr` is the penalty applied at the group level. The options are `lasso`, `scad`, `mcp` and `exp` for Least Absolute Shrinkage and Selection Operator, Smoothly Clipped Absolute Deviation, Minimax Concave Penalty and Exponential Penalty.

Value

A list containing:

beta A vector with estimated coefficients.

type A string indicating the type of regression model (linear or binomial).

group A vector indicating the group membership of the individual variables in X.

lambdas The sequence of lambda values.

alpha Tuning parameter for the mixture of penalties at group and variable level.

loss A vector containing either the residual sum of squares (linear) or the negative log-likelihood (binomial).

prec The convergence threshold used for each lambda.

n Number of observations.

penalty A string indicating the sparse group penalty used.

df A vector of pseudo degrees of freedom for each lambda.

iter A vector of the number of iterations for each lambda.

group.weight A vector of weights multiplied by the group penalty.

y The response vector.

X The design matrix without intercept.

References

- Buch, G., Schulz, A., Schmidtmann, I., Strauch, K., and Wild, P. S. (2024) Sparse Group Penalties for bi-level variable selection. Biometrical Journal, 66, 2200334. doi:10.1002/bimj.202200334
- Simon, N., Friedman, J., Hastie, T., and Tibshirani, R. (2011) A Sparse-Group Lasso. Journal of computational and graphical statistics, 22(2), 231-245. doi:10.1080/10618600.2012.681250
- Breheny, P., and Huang J. (2009) Penalized methods for bi-level variable selection. Statistics and its interface, 2: 369-380. doi:10.4310/sii.2009.v2.n3.a10

Examples

```
# Generate data
 n <- 100
 p <- 200
 nr <- 10
 g <- ceiling(1:p / nr)
 X <- matrix(rnorm(n * p), n, p)
 b <- c(-3:3)
 y_lin <- X[, 1:length(b)] %*% b + 5 * rnorm(n)
 y_log <- rbinom(n, 1, exp(y_lin) / (1 + exp(y_lin)))

# Linear regression
 lin_fit <- sgp(X, y_lin, g, type = "linear", penalty = "sgl")
 plot(lin_fit)
 lin_fit <- sgp(X, y_lin, g, type = "linear", penalty = "sgs")
```

```
plot(lin_fit)
lin_fit <- sgp(X, y_lin, g, type = "linear", penalty = "sgm")
plot(lin_fit)
lin_fit <- sgp(X, y_lin, g, type = "linear", penalty = "sge")
plot(lin_fit)

# Logistic regression
log_fit <- sgp(X, y_log, g, type = "logit", penalty = "sgl")
plot(log_fit)
log_fit <- sgp(X, y_log, g, type = "logit", penalty = "sgs")
plot(log_fit)
log_fit <- sgp(X, y_log, g, type = "logit", penalty = "sgm")
plot(log_fit)
log_fit <- sgp(X, y_log, g, type = "logit", penalty = "sge")
plot(log_fit)
```

`sgp.cv` *Cross-validation for sparse group penalties*

Description

A function that performs k-fold cross-validation for sparse group penalties for a lambda sequence.

Usage

```
sgp.cv(
  X,
  y,
  group = 1:ncol(X),
  Z = NULL,
  ...,
  nfolds = 10,
  seed,
  fold,
  type,
  returnY = FALSE,
  print.trace = FALSE
)
```

Arguments

`X`	The design matrix without intercept with the variables to be selected.
`y`	The response vector.
`group`	A vector indicating the group membership of each variable in X.
`Z`	The design matrix of the variables to be included in the model without penalization.

...	Other parameters of underlying basic functions.
nfolds	The number of folds for cross-validation.
seed	A seed provided by the user for the random number generator.
fold	A vector of folds specified by the user (default is a random assignment).
type	A string indicating the type of regression model (linear or binomial).
returnY	A Boolean value indicating whether the fitted values should be returned.
print.trace	A Boolean value that specifies whether the beginning of a fold should be printed.

Value

A list containing:

cve The average cross-validation error for each value of lambda.

cvse The estimated standard error for each value of cve.

lambdas The sequence of lambda values.

fit The sparse group penalty model fitted to the entire data.

fold The fold assignments for each observation for the cross-validation procedure.

min The index of lambda corresponding to the minimum cross-validation error.

lambda.min The value of lambda with the minimum cross-validation error.

null.dev The deviance for the empty model.

pe The cross-validation prediction error for each value of lambda (for binomial only).

pred The fitted values from the cross-validation folds.

Examples

```
# Generate data
 n <- 100
 p <- 200
 nr <- 10
 g <- ceiling(1:p / nr)
 X <- matrix(rnorm(n * p), n, p)
 b <- c(-3:3)
 y_lin <- X[, 1:length(b)] %*% b + 5 * rnorm(n)
 y_log <- rbinom(n, 1, exp(y_lin) / (1 + exp(y_lin)))

# Linear regression
 lin_fit <- sgp.cv(X, y_lin, g, type = "linear", penalty = "sgl")
 plot(lin_fit)
 predict(lin_fit, extract = "vars")
 lin_fit <- sgp.cv(X, y_lin, g, type = "linear", penalty = "sgs")
 plot(lin_fit)
 predict(lin_fit, extract = "vars")
 lin_fit <- sgp.cv(X, y_lin, g, type = "linear", penalty = "sgm")
 plot(lin_fit)
 predict(lin_fit, extract = "vars")
 lin_fit <- sgp.cv(X, y_lin, g, type = "linear", penalty = "sge")
```

```
plot(lin_fit)
predict(lin_fit, extract = "vars")

# Logistic regression
log_fit <- sgp.cv(X, y_log, g, type = "logit", penalty = "sgl")
plot(log_fit)
predict(log_fit, extract = "vars")
log_fit <- sgp.cv(X, y_log, g, type = "logit", penalty = "sgs")
plot(log_fit)
predict(log_fit, extract = "vars")
log_fit <- sgp.cv(X, y_log, g, type = "logit", penalty = "sgm")
plot(log_fit)
predict(log_fit, extract = "vars")
log_fit <- sgp.cv(X, y_log, g, type = "logit", penalty = "sge")
plot(log_fit)
predict(log_fit, extract = "vars")
```

Index

coef.sgp, 2
coef.sgp.cv, 3

get.loss, 4

plot.sgp, 4
plot.sgp.cv, 5
predict.sgp, 6
predict.sgp.cv, 7
process.group, 9
process.lambda, 9
process.penalty, 10
process.X, 11
process.y, 12
process.Z, 12

sgp, 13
sgp.cv, 16