Package 'sgs'

August 22, 2023

Title Sparse-Group SLOPE: Adaptive Bi-Level Selection with FDR Control

Version 0.1.1

Date 2023-08-21

Maintainer Fabio Feser <ff120@ic.ac.uk>

Description Implementation of Sparse-group SLOPE: Adaptive bi-level with FDR-control (Feser et al. (2023) <arXiv:2305.09467>). Linear and logistic regression models are supported, both of which can be fit using k-fold cross-validation. Dense and sparse input matrices are supported. In addition, a general adaptive three operator splitting (ATOS) implementation is provided.

Imports Matrix, MASS, caret, grDevices, graphics, methods, stats, faux, SLOPE, Rlab, Rcpp (>= 1.0.10)

LinkingTo Rcpp, RcppArmadillo

Suggests SGL, gglasso, glmnet, testthat, knitr, rmarkdown

RoxygenNote 7.2.3

License GPL (>= 3)

Encoding UTF-8

URL https://github.com/ff1201/sgs

BugReports https://github.com/ff1201/sgs/issues

VignetteBuilder knitr

NeedsCompilation yes

Author Fabio Feser [aut, cre] (<https://orcid.org/0009-0007-3088-9727>), Marina Evangelou [aut] (<https://orcid.org/0000-0003-0789-8944>)

Repository CRAN

Date/Publication 2023-08-22 15:50:05 UTC

R topics documented:

`arma_mv` *Matrix Product in RcppArmadillo.*

Description

Matrix Product in RcppArmadillo.

Usage

```
arma_mv(m, v)
```

Arguments

`m`	numeric matrix
`v`	numeric vector

Value

matrix product of m and v

`as_sgs` *fits the adaptively scaled SGS model (AS-SGS)*

Description

Fits an SGS model using the noise estimation procedure, termed adaptively scaled SGS (Algorithm 2 from Feser et al (2023)). This adaptively estimates λ and then fits the model using the estimated value. It is an alternative approach to cross-validation (`fit_sgs_cv()`). The approach is only compatible with the SGS penalties.

Usage

```
as_sgs(
  X,
  y,
  groups,
  type = "linear",
  pen_method = 2,
  alpha = 0.95,
  vFDR = 0.1,
  gFDR = 0.1,
  standardise = "l2",
  intercept = TRUE,
  verbose = FALSE
)
```

Arguments

X	Input matrix of dimensions $n \times p$. Can be a sparse matrix (using class "sparseMatrix" from the Matrix package).
y	Output vector of dimension n. For type="linear" should be continuous and for type="logistic" should be a binary variable.
groups	A grouping structure for the input data. Should take the form of a vector of group indices.
type	The type of regression to perform. Supported values are: "linear" and "logistic".
pen_method	The type of penalty sequences to use. • "1" uses the vMean and gMean SGS sequences. • "2" uses the vMax and gMax SGS sequences.
alpha	The value of α, which defines the convex balance between SLOPE and gSLOPE. Must be between 0 and 1.
vFDR	Defines the desired variable false discovery rate (FDR) level, which determines the shape of the variable penalties. Must be between 0 and 1.
gFDR	Defines the desired group false discovery rate (FDR) level, which determines the shape of the group penalties. Must be between 0 and 1.
standardise	Type of standardisation to perform on X: • "l2" standardises the input data to have ℓ_2 norms of one. • "l1" standardises the input data to have ℓ_1 norms of one. • "sd" standardises the input data to have standard deviation of one. • "none" no standardisation applied.
intercept	Logical flag for whether to fit an intercept.
verbose	Logical flag for whether to print fitting information.

Value

An object of type "sgs" containing model fit information (see fit_sgs()).

References

F. Feser, M. Evangelou *Sparse-group SLOPE: adaptive bi-level selection with FDR-control*, https://arxiv.org/abs/2305.09467

`atos` *adaptive three operator splitting (ATOS)*

Description

Function for fitting adaptive three operator splitting (ATOS) with general convex penalties. Supports both linear and logistic regression, both with dense and sparse matrix implementations.

Usage

```
atos(
  X,
  y,
  type = "linear",
  prox_1,
  prox_2,
  pen_prox_1 = 0.5,
  pen_prox_2 = 0.5,
  max_iter = 5000,
  backtracking = 0.7,
  max_iter_backtracking = 100,
  tol = 1e-05,
  prox_1_opts = NULL,
  prox_2_opts = NULL,
  standardise = "l2",
  intercept = TRUE,
  x0 = NULL,
  u = NULL,
  verbose = FALSE
)
```

Arguments

`X`	Input matrix of dimensions $n \times p$. Can be a sparse matrix (using class `"sparseMatrix"` from the `Matrix` package)
`y`	Output vector of dimension n. For `type="linear"` needs to be continuous and for `type="logistic"` needs to be a binary variable.
`type`	The type of regression to perform. Supported values are: `"linear"` and `"logistic"`.
`prox_1`	The proximal operator for the first function, $h(x)$.
`prox_2`	The proximal operator for the second function, $g(x)$.

`pen_prox_1`	The penalty for the first proximal operator. For the lasso, this would be the sparsity parameter, λ. If operator does not include a penalty, set to 1.
`pen_prox_2`	The penalty for the second proximal operator.
`max_iter`	Maximum number of ATOS iterations to perform.
`backtracking`	The backtracking parameter, τ, as defined in Pedregosa et. al. (2018).
`max_iter_backtracking`	Maximum number of backtracking line search iterations to perform per global iteration.
`tol`	Convergence tolerance for the stopping criteria.
`prox_1_opts`	Optional argument for first proximal operator. For the group lasso, this would be the group IDs. Note: this must be inserted as a list.
`prox_2_opts`	Optional argument for second proximal operator.
`standardise`	Type of standardisation to perform on `X`: • `"l2"` standardises the input data to have ℓ_2 norms of one. • `"l1"` standardises the input data to have ℓ_1 norms of one. • `"sd"` standardises the input data to have standard deviation of one. • `"none"` no standardisation applied.
`intercept`	Logical flag for whether to fit an intercept.
`x0`	Optional initial vector for x_0.
`u`	Optional initial vector for u.
`verbose`	Logical flag for whether to print fitting information.

Details

`atos()` solves convex minimization problems of the form

$$f(x) + g(x) + h(x),$$

where f is convex and differentiable with L_f-Lipschitz gradient, and g and h are both convex. The algorithm is not symmetrical, but usually the difference between variations are only small numerical values, which are filtered out. However, both variations should be checked regardless, by looking at `x` and `u`. An example for the sparse-group lasso (SGL) is given.

Value

An object of class `"atos"` containing:

`beta`	The fitted values from the regression. Taken to be the more stable fit between `x` and `u`, which is usually the former.
`x`	The solution to the original problem (see Pedregosa et. al. (2018)).
`u`	The solution to the dual problem (see Pedregosa et. al. (2018)).
`z`	The updated values from applying the first proximal operator (see Pedregosa et. al. (2018)).
`type`	Indicates which type of regression was performed.

success	Logical flag indicating whether ATOS converged, according to tol.
num_it	Number of iterations performed. If convergence is not reached, this will be max_iter.
certificate	Final value of convergence criteria.
intercept	Logical flag indicating whether an intercept was fit.

References

F. Pedregosa, G. Gidel (2018) *Adaptive Three Operator Splitting*, https://proceedings.mlr.press/v80/pedregosa18a.html

Examples

```
# specify a grouping structure
groups = c(rep(1:20, each=3),
          rep(21:40, each=4),
          rep(41:60, each=5),
          rep(61:80, each=6),
          rep(81:100, each=7))
# define proximal operators
L1_prox <- function(input, lambda){ # Lasso proximal operator
 out = sign(input) * pmax(0, abs(input) - lambda)
 return(out)
}
group_L1_prox = function(input,lambda,group_info){
 n_groups = length(unique(group_info))
 out = rep(0,length(input))
 for (i in 1:n_groups){
   grp_idx = which(group_info == unique(group_info)[i])
   if (lambda == 0 & norm(input[grp_idx],type="2") == 0){ # 0/0 = 0
     out[grp_idx] = 0
   } else {
     out[grp_idx] = max((1-(lambda/norm(input[grp_idx],type="2"))),0) * input[grp_idx]}
 }
 return(out)
}
# generate data
data = generate_toy_data(p=500, n=400, groups = groups, seed_id=3)
# run atos (the proximal functions can be found in utils.R)
out = atos(X=data$X, y=data$y, type="linear", prox_1 = L1_prox, prox_2 = group_L1_prox,
standardise="none", intercept=FALSE, prox_2_opts = list(groups))
```

---

fit_sgs *fit an SGS model*

Description

Sparse-group SLOPE (SGS) main fitting function. Supports both linear and logistic regression, both with dense and sparse matrix implementations.

Usage

```
fit_sgs(
  X,
  y,
  groups,
  pen_method = 1,
  type = "linear",
  lambda,
  alpha = 0.95,
  vFDR = 0.1,
  gFDR = 0.1,
  max_iter = 5000,
  backtracking = 0.7,
  max_iter_backtracking = 100,
  tol = 1e-05,
  standardise = "l2",
  intercept = TRUE,
  w_weights = NULL,
  v_weights = NULL,
  x0 = NULL,
  u = NULL,
  verbose = FALSE
)
```

Arguments

X	Input matrix of dimensions $n \times p$. Can be a sparse matrix (using class "sparseMatrix" from the Matrix package).
y	Output vector of dimension n. For type="linear" should be continuous and for type="logistic" should be a binary variable.
groups	A grouping structure for the input data. Should take the form of a vector of group indices.
pen_method	The type of penalty sequences to use (see Feser et al. (2023)): • "1" uses the vMean SGS and gMean gSLOPE sequences. • "2" uses the vMax SGS and gMean gSLOPE sequences. • "3" uses the BH SLOPE and gMean gSLOPE sequences, also known as SGS Original.
type	The type of regression to perform. Supported values are: "linear" and "logistic".
lambda	The value of λ, which defines the level of sparsity in the model. Can be picked using cross-validation (see fit_sgs_cv()). Must be a positive value.
alpha	The value of α, which defines the convex balance between SLOPE and gSLOPE. Must be between 0 and 1.
vFDR	Defines the desired variable false discovery rate (FDR) level, which determines the shape of the variable penalties. Must be between 0 and 1.
gFDR	Defines the desired group false discovery rate (FDR) level, which determines the shape of the group penalties. Must be between 0 and 1.

`max_iter`	Maximum number of ATOS iterations to perform.
`backtracking`	The backtracking parameter, τ, as defined in Pedregosa et. al. (2018).
`max_iter_backtracking`	Maximum number of backtracking line search iterations to perform per global iteration.
`tol`	Convergence tolerance for the stopping criteria.
`standardise`	Type of standardisation to perform on `X`: • `"l2"` standardises the input data to have ℓ_2 norms of one. • `"l1"` standardises the input data to have ℓ_1 norms of one. • `"sd"` standardises the input data to have standard deviation of one. • `"none"` no standardisation applied.
`intercept`	Logical flag for whether to fit an intercept.
`w_weights`	Optional vector for the group penalty weights. Overrides the penalties from `pen_method` if specified. When entering custom weights, these are multiplied internally by λ and $1 - \alpha$. To void this behaviour, set $\lambda = 2$ and $\alpha = 0.5$.
`v_weights`	Optional vector for the variable penalty weights. Overrides the penalties from `pen_method` if specified. When entering custom weights, these are multiplied internally by λ and α. To void this behaviour, set $\lambda = 2$ and $\alpha = 0.5$.
`x0`	Optional initial vector for x_0.
`u`	Optional initial vector for u.
`verbose`	Logical flag for whether to print fitting information.

Details

`fit_sgs()` fits an SGS model using adaptive three operator splitting (ATOS). SGS is a sparse-group method, so that it selects both variables and groups. Unlike group selection approaches, not every variable within a group is set as active. It solves the convex optimisation problem given by

$$\frac{1}{2n} f(b; y, \mathbf{X}) + \lambda\alpha \sum_{i=1}^{p} v_i |b|_{(i)} + \lambda(1-\alpha) \sum_{g=1}^{m} w_g \sqrt{p_g} \|b^{(g)}\|_2,$$

where $f(\cdot)$ is the loss function. In the case of the linear model, the loss function is given by the mean-squared error loss:

$$f(b; y, \mathbf{X}) = \|y - \mathbf{X}b\|_2^2 .$$

In the logistic model, the loss function is given by

$$f(b; y, \mathbf{X}) = -1/n \log(\mathcal{L}(b; y, \mathbf{X})).$$

where the log-likelihood is given by

$$\mathcal{L}(b; y, \mathbf{X}) = \sum_{i=1}^{n} \left\{ y_i b^\intercal x_i - \log(1 + \exp(b^\intercal x_i)) \right\}.$$

SGS can be seen to be a convex combination of SLOPE and gSLOPE, balanced through `alpha`, such that it reduces to SLOPE for `alpha = 0` and to gSLOPE for `alpha = 1`. The penalty parameters in SGS are sorted so that the largest coefficients are matched with the largest penalties, to reduce the FDR.

Value

A list containing:

`beta`	The fitted values from the regression. Taken to be the more stable fit between `x` and `u`, which is usually the former.
`x`	The solution to the original problem (see Pedregosa et. al. (2018)).
`u`	The solution to the dual problem (see Pedregosa et. al. (2018)).
`z`	The updated values from applying the first proximal operator (see Pedregosa et. al. (2018)).
`type`	Indicates which type of regression was performed.
`pen_slope`	Vector of the variable penalty sequence.
`pen_gslope`	Vector of the group penalty sequence.
`lambda`	Value of λ used to fit the model.
`success`	Logical flag indicating whether ATOS converged, according to `tol`.
`num_it`	Number of iterations performed. If convergence is not reached, this will be `max_iter`.
`certificate`	Final value of convergence criteria.
`intercept`	Logical flag indicating whether an intercept was fit.

References

F. Feser, M. Evangelou *Sparse-group SLOPE: adaptive bi-level selection with FDR-control*, https://arxiv.org/abs/2305.09467

F. Pedregosa, G. Gidel (2018) *Adaptive Three Operator Splitting*, https://proceedings.mlr.press/v80/pedregosa18a.html

Examples

```
# specify a grouping structure
groups = c(1,1,1,2,2,3,3,3,4,4)
# generate data
data = generate_toy_data(p=10, n=5, groups = groups, seed_id=3,group_sparsity=1)
# run SGS
model = fit_sgs(X = data$X, y = data$y, groups = groups, type="linear", lambda = 1, alpha=0.95,
vFDR=0.1, gFDR=0.1, standardise = "l2", intercept = TRUE, verbose=FALSE)
```

`fit_sgs_cv` *fit an SGS model using CV*

Description

Function to fit a pathwise solution of sparse-group SLOPE (SGS) models using k-fold cross-validation. Supports both linear and logistic regression, both with dense and sparse matrix implementations.

Usage

```
fit_sgs_cv(
  X,
  y,
  groups,
  pen_method = 1,
  type = "linear",
  nlambda = 20,
  nfolds = 10,
  alpha = 0.95,
  vFDR = 0.1,
  gFDR = 0.1,
  backtracking = 0.7,
  max_iter = 5000,
  max_iter_backtracking = 100,
  tol = 1e-05,
  min_frac = 0.05,
  standardise = "l2",
  intercept = TRUE,
  verbose = FALSE,
  v_weights = NULL,
  w_weights = NULL,
  error_criteria = "mse",
  max_lambda = NULL
)
```

Arguments

X	Input matrix of dimensions $n \times p$. Can be a sparse matrix (using class "sparseMatrix" from the Matrix package).
y	Output vector of dimension n. For type="linear" should be continuous and for type="logistic" should be a binary variable.
groups	A grouping structure for the input data. Should take the form of a vector of group indices.
pen_method	The type of penalty sequences to use (see Feser et al. (2023)): • "1" uses the vMean SGS and gMean gSLOPE sequences. • "2" uses the vMax SGS and gMean gSLOPE sequences. • "3" uses the BH SLOPE and gMean gSLOPE sequences, also known as SGS Original.
type	The type of regression to perform. Supported values are: "linear" and "logistic".
nlambda	The number of pathwise λ values to fit.
nfolds	The number of folds to use in cross-validation.
alpha	The value of α, which defines the convex balance between SLOPE and gSLOPE. Must be between 0 and 1.
vFDR	Defines the desired variable false discovery rate (FDR) level, which determines the shape of the variable penalties. Must be between 0 and 1.

gFDR	Defines the desired group false discovery rate (FDR) level, which determines the shape of the group penalties. Must be between 0 and 1.
backtracking	The backtracking parameter, τ, as defined in Pedregosa et. al. (2018).
max_iter	Maximum number of ATOS iterations to perform.
max_iter_backtracking	Maximum number of backtracking line search iterations to perform per global iteration.
tol	Convergence tolerance for the stopping criteria.
min_frac	Defines the termination point of the pathwise solution, so that $\lambda_{\text{min}} = min_frac \cdot \lambda_{\text{max}}$.
standardise	Type of standardisation to perform on X: • "l2" standardises the input data to have ℓ_2 norms of one. • "l1" standardises the input data to have ℓ_1 norms of one. • "sd" standardises the input data to have standard deviation of one. • "none" no standardisation applied.
intercept	Logical flag for whether to fit an intercept.
verbose	Logical flag for whether to print fitting information.
v_weights	Optional vector for the variable penalty weights. Overrides the penalties from pen_method if specified. When entering custom weights, these are multiplied internally by λ and α. To void this behaviour, set $\lambda = 2$ and $\alpha = 0.5$
w_weights	Optional vector for the group penalty weights. Overrides the penalties from pen_method if specified. When entering custom weights, these are multiplied internally by λ and $1 - \alpha$. To void this behaviour, set $\lambda = 2$ and $\alpha = 0.5$
error_criteria	The criteria used to discriminate between models along the path. Supported values are: "mse" (mean squared error) and "mae" (mean absolute error).
max_lambda	Optional parameter, λ_{max}, which is used to fit the first model on the path. If not specificed, it is chosen to be just above the value which lets in the first variable (so that it is the null model).

Details

Fits SGS models under a pathwise solution using adaptive three operator splitting (ATOS), picking the 1se model as optimum. Warm starts are implemented.

Value

A list containing:

all_models	A list of all the models fitted along the path.
fit	The 1se chosen model, which is a "sgs" object type.
best_lambda	The value of λ which generated the chosen model.
best_lambda_id	The path index for the chosen model.
errors	A table containing fitting information about the models on the path.
type	Indicates which type of regression was performed.

References

F. Feser, M. Evangelou *Sparse-group SLOPE: adaptive bi-level selection with FDR-control*, https://arxiv.org/abs/2305.09467

F. Pedregosa, G. Gidel (2018) *Adaptive Three Operator Splitting*, https://proceedings.mlr.press/v80/pedregosa18a.html

Examples

```
# specify a grouping structure
groups = c(1,1,1,2,2,3,3,3,4,4)
# generate data
data = generate_toy_data(p=10, n=5, groups = groups, seed_id=3,group_sparsity=1)
# run SGS with cross-validation (the proximal functions can be found in utils.R)
cv_model = fit_sgs_cv(X = data$X, y = data$y, groups=groups, type = "linear",
nlambda = 5, nfolds=10, alpha = 0.95, vFDR = 0.1, gFDR = 0.1, min_frac = 0.05,
standardise="l2",intercept=TRUE,verbose=TRUE)
```

---

generate_penalties *generate penalty sequences for SGS*

Description

Generates variable and group penalties for SGS.

Usage

```
generate_penalties(gFDR, vFDR, pen_method, groups, alpha)
```

Arguments

gFDR	Defines the desired group false discovery rate (FDR) level, which determines the shape of the group penalties.
vFDR	Defines the desired variable false discovery rate (FDR) level, which determines the shape of the variable penalties.
pen_method	The type of penalty sequences to use (see Feser et al. (2023)): • "1" uses the vMean SGS and gMean gSLOPE sequences. • "2" uses the vMax SGS and gMean gSLOPE sequences. • "3" uses the BH SLOPE and gMean gSLOPE sequences, also known as SGS Original.
groups	A grouping structure for the input data. Should take the form of a vector of group indices.
alpha	The value of α, defines the convex balance between SLOPE and gSLOPE.

Details

The vMean and vMax SGS sequences are variable sequences derived specifically to give variable false discovery rate (FDR) control for SGS under orthogonal designs (see Feser et al. (2023)). The BH SLOPE sequence is derived in Bodgan et. al. (2015) and has links to the Benjamini-Hochberg critical values. The sequence provides variable FDR-control for SLOPE under orthogonal designs. The gMean gSLOPE sequence is derived in Brzyski et. al. (2015) and provides group FDR-control for gSLOPE under orthogonal designs.

Value

A list containing:

`pen_slope_org` A vector of the variable penalty sequence.

`pen_gslope_org` A vector of the group penalty sequence.

References

F. Feser, M. Evangelou *Sparse-group SLOPE: adaptive bi-level selection with FDR-control*, https://arxiv.org/abs/2305.09467

M. Bogdan, E. Van den Berg, C. Sabatti, W. Su, E. Candes (2015) *SLOPE — Adaptive variable selection via convex optimization*, https://projecteuclid.org/journals/annals-of-applied-statistics/volume-9/issue-3/SLOPEAdaptive-variable-selection-via-convex-optimization/10.1214/15-AOAS842.full

D. Brzyski, W. Su, M. Bodgdan (2015) *Group SLOPE - adaptive selection of groups of predictors*, https://arxiv.org/abs/1511.09078

Examples

```
# specify a grouping structure
groups = c(rep(1:20, each=3),
          rep(21:40, each=4),
          rep(41:60, each=5),
          rep(61:80, each=6),
          rep(81:100, each=7))
# generate sequences
sequences = generate_penalties(gFDR=0.1, vFDR=0.1, pen_method=1, groups=groups, alpha=0.5)
```

---

generate_toy_data *generate toy data*

Description

Generates different types of datasets, which can then be fitted using sparse-group SLOPE.

Usage

```
generate_toy_data(
  p,
  n,
  rho = 0,
  seed_id = 2,
  grouped = TRUE,
  groups,
  noise_level = 1,
  group_sparsity = 0.1,
  var_sparsity = 0.5,
  orthogonal = FALSE,
  data_mean = 0,
  data_sd = 1,
  signal_mean = 0,
  signal_sd = sqrt(10)
)
```

Arguments

p	The number of input variables.
n	The number of observations.
rho	Correlation coefficient. Must be in range $[0, 1]$.
seed_id	Seed to be used to generate the data matrix X.
grouped	A logical flag indicating whether grouped data is required.
groups	If itemgrouped=TRUE, the grouping structure is required. Each input variable should have a group id.
noise_level	Defines the level of noise ($sigma$) to be used in generating the response vector y.
group_sparsity	Defines the level of group sparsity. Must be in the range $[0, 1]$.
var_sparsity	Defines the level of variable sparsity. Must be in the range $[0, 1]$. If grouped=TRUE, this defines the level of sparsity within each group, not globally.
orthogonal	Logical flag as to whether the input matrix should be orthogonal.
data_mean	Defines the mean of input predictors.
data_sd	Defines the standard deviation of the signal ($beta$).
signal_mean	Defines the mean of the signal ($beta$).
signal_sd	Defines the standard deviation of the signal ($beta$).

Details

The data is generated under a Gaussian linear model. The generated data can be grouped and sparsity can be provided at both a group and/or variable level.

Value

A list containing:

y	The response vector.
X	The input matrix.
true_beta	The true values of $beta$ used to generate the response.
true_grp_id	Indices of which groups are non-zero in itemtrue_beta.

Examples

```
# specify a grouping structure
groups = c(rep(1:20, each=3),
          rep(21:40, each=4),
          rep(41:60, each=5),
          rep(61:80, each=6),
          rep(81:100, each=7))
# generate data
data = generate_toy_data(p=500, n=400, groups = groups, seed_id=3)
```

---

plot.sgs_cv *plot a "sgs_cv" object*

Description

Plots the pathwise solution of a cross-validation fit, from a call to `fit_sgs_cv()`

Usage

```
## S3 method for class 'sgs_cv'
plot(x, how_many = 10, ...)
```

Arguments

x	Object an object of class "sgs_cv" from a call to `fit_sgs()`.
how_many	Defines how many predictors to plot. Plots the predictors in decreasing order of largest absolute value.
...	further arguments passed to base function.

Value

A list containing:

response	The predicted response. In the logistic case, this represents the predicted class probabilities.
class	The predicted class assignments. Only returned if type = "logistic" in the "sgs" object.

See Also

fit_sgs_cv()

Other SGS-methods: predict.sgs(), print.sgs()

Examples

```
# specify a grouping structure
groups = c(1,1,2,2,3)
# generate data
data = generate_toy_data(p=5, n=4, groups = groups, seed_id=3,signal_mean=20,group_sparsity=1)
# run SGS
cv_model = fit_sgs_cv(X = data$X, y = data$y, groups=groups, type = "linear",
nlambda = 20, nfolds=10, alpha = 0.95, vFDR = 0.1, gFDR = 0.1,
min_frac = 0.05, standardise="l2",intercept=TRUE,verbose=FALSE)
plot(cv_model, how_many = 10)
```

---

predict.sgs *predict using a "sgs" object*

Description

Performs prediction from an fit_sgs() model fit.

Usage

```
## S3 method for class 'sgs'
predict(object, x, ...)
```

Arguments

object	an object of class "sgs" from a call to fit_sgs().
x	Input data to use for prediction.
...	further arguments passed to stats function.

Value

A list containing: itemresponseThe predicted response. In the logistic case, this represents the predicted class probabilities. itemclassThe predicted class assignments. Only returned if type = "logistic" in the "sgs" object.

See Also

fit_sgs()

Other SGS-methods: plot.sgs_cv(), print.sgs()

Examples

```
# specify a grouping structure
groups = c(1,1,1,2,2,3,3,3,4,4)
# generate data
data = generate_toy_data(p=10, n=5, groups = groups, seed_id=3,group_sparsity=1)
# run SGS
model = fit_sgs(X = data$X, y = data$y, groups = groups, type="linear", lambda = 1, alpha=0.95,
vFDR=0.1, gFDR=0.1, standardise = "l2", intercept = TRUE, verbose=FALSE)
# use predict function
model_predictions = predict(model, x = data$X)
```

`print.sgs` *print a* `"sgs"` *object*

Description

Performs prediction from an `fit_sgs()` model fit.

Usage

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

Arguments

`x`	Object an object of class `"sgs"` from a call to `fit_sgs()` or `fit_sgs_cv()`.
`...`	further arguments passed to base function.

Value

A summary of the model fit.

See Also

`fit_sgs()`, `fit_sgs_cv()`

Other SGS-methods: `plot.sgs_cv()`, `predict.sgs()`

Examples

```
# specify a grouping structure
groups = c(rep(1:20, each=3),
          rep(21:40, each=4),
          rep(41:60, each=5),
          rep(61:80, each=6),
          rep(81:100, each=7))
# generate data
data = generate_toy_data(p=500, n=400, groups = groups, seed_id=3)
# run SGS
```

```
model = fit_sgs(X = data$X, y = data$y, groups = groups, type="linear", lambda = 1, alpha=0.95,
vFDR=0.1, gFDR=0.1, standardise = "l2", intercept = TRUE, verbose=FALSE)
# print model
print(model)
```

`scaled_sgs` *fits a scaled SGS model*

Description

Fits an SGS model using the noise estimation procedure (Algorithm 5 from Bogdan et. al. (2015)). This estimates λ and then fits the model using the estimated value. It is an alternative approach to cross-validation (`fit_sgs_cv()`).

Usage

```
scaled_sgs(
  X,
  y,
  groups,
  type = "linear",
  pen_method = 1,
  alpha = 0.95,
  vFDR = 0.1,
  gFDR = 0.1,
  standardise = "l2",
  intercept = TRUE,
  verbose = FALSE
)
```

Arguments

`X`	Input matrix of dimensions $n \times p$. Can be a sparse matrix (using class `"sparseMatrix"` from the `Matrix` package).
`y`	Output vector of dimension n. For `type="linear"` should be continuous and for `type="logistic"` should be a binary variable.
`groups`	A grouping structure for the input data. Should take the form of a vector of group indices.
`type`	The type of regression to perform. Supported values are: `"linear"` and `"logistic"`.
`pen_method`	The type of penalty sequences to use. • `"1"` uses the vMean SGS and gMean gSLOPE sequences. • `"2"` uses the vMax SGS and gMean gSLOPE sequences. • `"1"` uses the BH SLOPE and gMean gSLOPE sequences, also known as SGS Original.
`alpha`	The value of α, which defines the convex balance between SLOPE and gSLOPE. Must be between 0 and 1.

vFDR	Defines the desired variable false discovery rate (FDR) level, which determines the shape of the variable penalties. Must be between 0 and 1.
gFDR	Defines the desired group false discovery rate (FDR) level, which determines the shape of the group penalties. Must be between 0 and 1.
standardise	Type of standardisation to perform on X: • "l2" standardises the input data to have ℓ_2 norms of one. • "l1" standardises the input data to have ℓ_1 norms of one. • "sd" standardises the input data to have standard deviation of one. • "none" no standardisation applied.
intercept	Logical flag for whether to fit an intercept.
verbose	Logical flag for whether to print fitting information.

Value

An object of type "sgs" containing model fit information (see fit_sgs()).

References

M. Bogdan, E. Van den Berg, C. Sabatti, W. Su, E. Candes (2015) *SLOPE — Adaptive variable selection via convex optimization*, https://projecteuclid.org/journals/annals-of-applied-statistics/volume-9/issue-3/SLOPEAdaptive-variable-selection-via-convex-optimization/10.1214/15-AOAS842.full

Examples

```
# specify a grouping structure
groups = c(1,1,2,2,3)
# generate data
data = generate_toy_data(p=5, n=4, groups = groups, seed_id=3,
signal_mean=20,group_sparsity=1,var_sparsity=1)
# run noise estimation
model = scaled_sgs(X=data$X, y=data$y, groups=groups,pen_method=1)
```

Index