Package 'msgl'

May 8, 2019

Type Package

Title Multinomial Sparse Group Lasso

Version 2.3.9

Date 2019-05-07

Description Multinomial logistic regression with sparse group lasso penalty. Simultaneous feature selection and parameter estimation for classification. Suitable for high dimensional multiclass classification with many classes. The algorithm computes the sparse group lasso penalized maximum likelihood estimate. Use of parallel computing for cross validation and subsampling is supported through the 'foreach' and 'doParallel' packages. Development version is on GitHub, please report package issues on GitHub.

URL http://www.sciencedirect.com/science/article/pii/S0167947313002168, https://github.com/nielsrhansen/msgl

BugReports https://github.com/nielsrhansen/msgl/issues

License GPL (>= 2)

LazyLoad yes

Imports methods, tools, utils, stats

Depends R (>= 3.2.4), Matrix, sglOptim (>= 1.3.7)

LinkingTo Rcpp, RcppProgress, RcppArmadillo, BH, sglOptim

Suggests knitr, rmarkdown

VignetteBuilder knitr

RoxygenNote 6.1.1

Encoding UTF-8

NeedsCompilation yes

Author Martin Vincent [aut], Niels Richard Hansen [ctb, cre]

Maintainer Niels Richard Hansen <niels.r.hansen@math.ku.dk>

Repository CRAN

Date/Publication 2019-05-08 08:00:10 UTC

R topics documented:

`msgl-package` *Multinomial logistic regression with sparse group lasso penalty.*

Description

Simultaneous feature selection and parameter estimation for classification. Suitable for high dimensional multiclass classification with many classes. The algorithm computes the sparse group lasso penalized maximum likelihood estimate. Use of parallel computing for cross validation and subsampling is supported through the `foreach` and `doParallel` packages. Development version is on GitHub, please report package issues on GitHub.

Details

For a classification problem with K classes and p features (covariates) dived into m groups. The multinomial logistic regression with sparse group lasso penalty estimator is a sequence of minimizers (one for each lambda given in the `lambda` argument) of

$$\hat{R}(\beta) + \lambda \left((1-\alpha) \sum_{J=1}^{m} \gamma_J \|\beta^{(J)}\|_2 + \alpha \sum_{i=1}^{n} \xi_i |\beta_i| \right)$$

where $\hat{R}$ is the weighted empirical log-likelihood risk of the multinomial regression model. The vector $\beta^{(J)}$ denotes the parameters associated with the J'th group of features (default is one covariate per group, hence the default dimension of $\beta^{(J)}$ is K). The group weights $\gamma \in [0, \infty)^m$ and parameter weights $\xi \in [0, \infty)^n$ may be explicitly specified.

Author(s)

Martin Vincent

Examples

```
# Load some data
data(PrimaryCancers)

# A quick look at the data
dim(x)
table(classes)

# A smaller subset with three classes
small <- which(classes %in% c("CCA", "CRC", "Pancreas"))
classes <- classes[small, drop = TRUE]
x <- x[small, ]

#Do cross validation using 2 parallel units
cl <- makeCluster(2)
registerDoParallel(cl)

# Do 4-fold cross validation on a lambda sequence of length 100.
# The sequence is decreasing from the data derived lambda.max to 0.2*lambda.max
fit.cv <- msgl::cv(x, classes, fold = 4, lambda = 0.2, use_parallel = TRUE)

stopCluster(cl)

# Print information about models
# and cross validation errors (estimated expected generalization error)
fit.cv
```

---

best_model.msgl *Index of best model*

Description

Returns the index of the best model, in terms of lowest error rate

Usage

```
## S3 method for class 'msgl'
best_model(object, ...)
```

Arguments

`object`	a msgl object
`...`	additional parameters (ignored)

Value

index of the best model.

Author(s)

Martin Vincent

`classes` *Class vector*

Description

Class vector

`coef.msgl` *Nonzero coefficients*

Description

This function returns the nonzero coefficients (that is the nonzero entries of the $beta$ matrices)

Usage

```
## S3 method for class 'msgl'
coef(object, index = 1:nmod(object), ...)
```

Arguments

`object`	a msgl object
`index`	indices of the models
`...`	ignored

Value

a list of length `length(index)` with nonzero coefficients of the models

Author(s)

Martin Vincent

Examples

```
data(SimData)


lambda <- msgl::lambda(x, classes, alpha = .5, d = 50, lambda.min = 0.05)
fit <- msgl::fit(x, classes, alpha = .5, lambda = lambda)

# the nonzero coefficients of the models 1, 10 and 20
coef(fit, index = c(1,10,20))
```

---

cv *Cross Validation*

Description

Multinomial sparse group lasso cross validation, with or without parallel backend.

Usage

```
cv(x, classes, sampleWeights = NULL, grouping = NULL,
  groupWeights = NULL, parameterWeights = NULL, alpha = 0.5,
  standardize = TRUE, lambda, d = 100, fold = 10L,
  cv.indices = list(), intercept = TRUE, sparse.data = is(x,
  "sparseMatrix"), max.threads = NULL, use_parallel = FALSE,
  algorithm.config = msgl.standard.config)
```

Arguments

`x`	design matrix, matrix of size $N \times p$.
`classes`	classes, factor of length N.
`sampleWeights`	sample weights, a vector of length N.
`grouping`	grouping of features (covariates), a vector of length p. Each element of the vector specifying the group of the feature.
`groupWeights`	the group weights, a vector of length m (the number of groups). If `groupWeights = NULL` default weights will be used. Default weights are 0 for the intercept and $$\sqrt{K \cdot \text{number of features in the group}}$$ for all other weights.
`parameterWeights`	a matrix of size $K \times p$. If `parameterWeights = NULL` default weights will be used. Default weights are is 0 for the intercept weights and 1 for all other weights.#'
`alpha`	the α value 0 for group lasso, 1 for lasso, between 0 and 1 gives a sparse group lasso penalty.

standardize	if TRUE the features are standardize before fitting the model. The model parameters are returned in the original scale.
lambda	lambda.min relative to lambda.max or the lambda sequence for the regularization path.
d	length of lambda sequence (ignored if `length(lambda) > 1`)
fold	the fold of the cross validation, an integer larger than 1 and less than $N + 1$. Ignored if `cv.indices != NULL`. If `fold`$\leq$`max(table(classes))` then the data will be split into `fold` disjoint subsets keeping the ration of classes approximately equal. Otherwise the data will be split into `fold` disjoint subsets without keeping the ration fixed.
cv.indices	a list of indices of a cross validation splitting. If `cv.indices = NULL` then a random splitting will be generated using the `fold` argument.
intercept	should the model include intercept parameters
sparse.data	if TRUE x will be treated as sparse, if x is a sparse matrix it will be treated as sparse by default.
max.threads	Deprecated (will be removed in 2018), instead use `use_parallel = TRUE` and registre parallel backend (see package 'doParallel'). The maximal number of threads to be used.
use_parallel	If `TRUE` the `foreach` loop will use `%dopar%`. The user must registre the parallel backend.
algorithm.config	the algorithm configuration to be used.

Value

link	the linear predictors – a list of length `length(lambda)` one item for each lambda value, with each item a matrix of size $K \times N$ containing the linear predictors.
response	the estimated probabilities - a list of length `length(lambda)` one item for each lambda value, with each item a matrix of size $K \times N$ containing the probabilities.
classes	the estimated classes - a matrix of size $N \times d$ with $d =$`length(lambda)`.
cv.indices	the cross validation splitting used.
features	number of features used in the models.
parameters	number of parameters used in the models.
classes.true	the true classes used for estimation, this is equal to the `classes` argument

Author(s)

Martin Vincent

Examples

```
data(SimData)

# A quick look at the data
dim(x)
```

```
table(classes)

# Setup clusters
cl <- makeCluster(2)
registerDoParallel(cl)

# Run cross validation using 2 clusters
# Using a lambda sequence ranging from the maximal lambda to 0.7 * maximal lambda
fit.cv <- msgl::cv(x, classes, alpha = 0.5, lambda = 0.7, use_parallel = TRUE)

# Stop clusters
stopCluster(cl)

# Print some information
fit.cv

# Cross validation errors (estimated expected generalization error)
# Misclassification rate
Err(fit.cv)

# Negative log likelihood error
Err(fit.cv, type="loglike")
```

`Err.msgl` *Compute error rates*

Description

Compute error rates. If `type = "rate"` then the misclassification rates will be computed. If `type = "count"` then the misclassification counts will be computed. If `type = "loglike"` then the negative log likelihood error will be computed.

Usage

```
## S3 method for class 'msgl'
Err(object, data = NULL, response = object$classes.true,
  classes = response, type = "rate", ...)
```

Arguments

`object`	a msgl object
`data`	a matrix of
`response`	a vector of classes
`classes`	a vector of classes
`type`	type of error rate
`...`	ignored

Value

a vector of error rates

Author(s)

Martin Vincent

Examples

```
data(SimData)

x.all <- x
x.1 <- x[1:50,]
x.2 <- x[51:100,]
classes.all <- classes
classes.1 <- classes[1:50]
classes.2 <- classes[51:100]

#### Fit models using x.1
lambda <- msgl::lambda(x.1, classes.1, alpha = .5, d = 25, lambda.min = 0.075)
fit <- msgl::fit(x.1, classes.1, alpha = .5, lambda = lambda)

#### Training errors:

# Misclassification rate
Err(fit, x.1)

# Misclassification count
Err(fit, x.1, type = "count")

# Negative log likelihood error
Err(fit, x.1, type="loglike")

# Misclassification rate of x.2
Err(fit, x.2, classes.2)

#### Do cross validation
fit.cv <- msgl::cv(x.all, classes.all, alpha = .5, lambda = lambda)

#### Cross validation errors (estimated expected generalization error)

# Misclassification rate
Err(fit.cv)

# Negative log likelihood error
Err(fit.cv, type="loglike")

#### Do subsampling
test <- list(1:20, 21:40)
train <- lapply(test, function(s) (1:length(classes.all))[-s])

fit.sub <- msgl::subsampling(x.all, classes.all, alpha = .5,
```

```
 lambda = lambda, training = train, test = test)

# Mean misclassification error of the tests
Err(fit.sub)

# Negative log likelihood error
Err(fit.sub, type="loglike")
```

`features.msgl` *Nonzero features*

Description

Extracts the nonzero features for each model.

Usage

```
## S3 method for class 'msgl'
features(object, ...)
```

Arguments

`object`	a msgl object
`...`	ignored

Value

a list of of length `nmod(x)` containing the nonzero features (that is nonzero colums of the beta matrices)

Author(s)

Martin Vincent

Examples

```
data(SimData)


lambda <- msgl::lambda(x, classes, alpha = .5, d = 50, lambda.min = 0.05)
fit <- msgl::fit(x, classes, alpha = .5, lambda = lambda)

# the nonzero features of model 1, 10 and 25
features(fit)[c(1,10,25)]

# count the number of nonzero features in each model
sapply(features(fit), length)
```

features_stat.msgl *Extract feature statistics*

Description

Extracts the number of nonzero features (or group) in each model.

Usage

```
## S3 method for class 'msgl'
features_stat(object, ...)
```

Arguments

object	a msgl object
...	ignored

Value

a vector of length nmod(x) or a matrix containing the number of nonzero features (or group) of the models.

Author(s)

Martin Vincent

fit *Fit a multinomial sparse group lasso regularization path.*

Description

Fit a sequence of multinomial logistic regression models using sparse group lasso, group lasso or lasso. In addition to the standard parameter grouping the algorithm supports further grouping of the features.

Usage

```
fit(x, classes, sampleWeights = NULL, grouping = NULL,
  groupWeights = NULL, parameterWeights = NULL, alpha = 0.5,
  standardize = TRUE, lambda, d = 100, return_indices = NULL,
  intercept = TRUE, sparse.data = is(x, "sparseMatrix"),
  algorithm.config = msgl.standard.config)
```

Arguments

`x`	design matrix, matrix of size $N \times p$.
`classes`	classes, factor of length N.
`sampleWeights`	sample weights, a vector of length N.
`grouping`	grouping of features, a vector of length p. Each element of the vector specifying the group of the feature.
`groupWeights`	the group weights, a vector of length m (the number of groups). If `groupWeights = NULL` default weights will be used. Default weights are 0 for the intercept and $$\sqrt{K \cdot \text{number of features in the group}}$$ for all other weights.
`parameterWeights`	a matrix of size $K \times p$. If `parameterWeights = NULL` default weights will be used. Default weights are is 0 for the intercept weights and 1 for all other weights.
`alpha`	the α value 0 for group lasso, 1 for lasso, between 0 and 1 gives a sparse group lasso penalty.
`standardize`	if TRUE the features are standardize before fitting the model. The model parameters are returned in the original scale.
`lambda`	lambda.min relative to lambda.max or the lambda sequence for the regularization path.
`d`	length of lambda sequence (ignored if `length(lambda) > 1`)
`return_indices`	the indices of lambda values for which to return a the fitted parameters.
`intercept`	should the model fit include intercept parameters (note that due to standardization the returned beta matrix will always have an intercept column)
`sparse.data`	if TRUE `x` will be treated as sparse, if `x` is a sparse matrix it will be treated as sparse by default.
`algorithm.config`	the algorithm configuration to be used.

Details

For a classification problem with K classes and p features (covariates) dived into m groups. This function computes a sequence of minimizers (one for each lambda given in the `lambda` argument) of

$$\hat{R}(\beta) + \lambda \left((1-\alpha) \sum_{J=1}^{m} \gamma_J \|\beta^{(J)}\|_2 + \alpha \sum_{i=1}^{n} \xi_i |\beta_i| \right)$$

where $\hat{R}$ is the weighted empirical log-likelihood risk of the multinomial regression model. The vector $\beta^{(J)}$ denotes the parameters associated with the J'th group of features (default is one covariate per group, hence the default dimension of $\beta^{(J)}$ is K). The group weights $\gamma \in [0, \infty)^m$ and parameter weights $\xi \in [0, \infty)^n$ may be explicitly specified.

Value

beta	the fitted parameters – a list of length length(lambda) with each entry a matrix of size $K \times (p+1)$ holding the fitted parameters
loss	the values of the loss function
objective	the values of the objective function (i.e. loss + penalty)
lambda	the lambda values used
classes.true	the true classes used for estimation, this is equal to the classes argument

Author(s)

Martin Vincent

Examples

```
data(SimData)

# A quick look at the data
dim(x)
table(classes)
# Fit multinomial sparse group lasso regularization path
# using a lambda sequence ranging from the maximal lambda to 0.5 * maximal lambda

fit <- msgl::fit(x, classes, alpha = 0.5, lambda = 0.5)

# Print some information about the fit
fit

# Model 10, i.e. the model corresponding to lambda[10]
models(fit)[[10]]

# The nonzero features of model 10
features(fit)[[10]]

# The nonzero parameters of model 10
parameters(fit)[[10]]

# The training errors of the models.
Err(fit, x)
# Note: For high dimensional models the training errors are almost always over optimistic,
# instead use msgl::cv to estimate the expected errors by cross validation
```

---

lambda *Computes a lambda sequence for the regularization path*

Description

Computes a decreasing lambda sequence of length d. The sequence ranges from a data determined maximal lambda $\lambda_{\max}$ to the user inputed lambda.min.

Usage

```
lambda(x, classes, sampleWeights = NULL, grouping = NULL,
  groupWeights = NULL, parameterWeights = NULL, alpha = 0.5,
  d = 100L, standardize = TRUE, lambda.min, intercept = TRUE,
  sparse.data = is(x, "sparseMatrix"), lambda.min.rel = FALSE,
  algorithm.config = msgl.standard.config)
```

Arguments

`x`	design matrix, matrix of size $N \times p$.
`classes`	classes, factor of length N.
`sampleWeights`	sample weights, a vector of length N.
`grouping`	grouping of features, a vector of length p. Each element of the vector specifying the group of the covariate.
`groupWeights`	the group weights, a vector of length $m + 1$ (the number of groups). The first element of the vector is the intercept weight. If `groupWeights = NULL` default weights will be used. Default weights are 0 for the intercept and $$\sqrt{K \cdot \text{number of features in the group}}$$ for all other weights.
`parameterWeights`	a matrix of size $K \times (p + 1)$. The first column of the matrix is the intercept weights. Default weights are is 0 for the intercept weights and 1 for all other weights.
`alpha`	the α value 0 for group lasso, 1 for lasso, between 0 and 1 gives a sparse group lasso penalty.
`d`	the length of lambda sequence
`standardize`	if TRUE the features are standardize before fitting the model. The model parameters are returned in the original scale.
`lambda.min`	the smallest lambda value in the computed sequence.
`intercept`	should the model include intercept parameters
`sparse.data`	if TRUE `x` will be treated as sparse, if `x` is a sparse matrix it will be treated as sparse by default.
`lambda.min.rel`	is lambda.min relative to lambda.max ? (i.e. actual lambda min used is `lambda.min*lambda.max`, with `lambda.max` the computed maximal lambda value)
`algorithm.config`	the algorithm configuration to be used.

Value

a vector of length `d` containing the computed lambda sequence.

Author(s)

Martin Vincent

Examples

```
data(SimData)

# A quick look at the data
dim(x)
table(classes)

lambda <- msgl::lambda(x, classes, alpha = .5, d = 100, lambda.min = 0.01)
```

`models.msgl` *Extract the fitted models*

Description

Returns the fitted models, that is the estimated β matrices.

Usage

```
## S3 method for class 'msgl'
models(object, index = 1:nmod(object), ...)
```

Arguments

`object`	a msgl object
`index`	indices of the models to be returned
`...`	ignored

Value

a list of β matrices.

Author(s)

Martin Vincent

`msgl.algorithm.config` *Create a new algorithm configuration*

Description

With the exception of `verbose` it is not recommended to change any of the default values.

Usage

```
msgl.algorithm.config(tolerance_penalized_main_equation_loop = 1e-10,
  tolerance_penalized_inner_loop_alpha = 1e-04,
  tolerance_penalized_inner_loop_beta = 1,
  tolerance_penalized_middel_loop_alpha = 0.01,
  tolerance_penalized_outer_loop_alpha = 0.01,
  tolerance_penalized_outer_loop_beta = 0,
  tolerance_penalized_outer_loop_gamma = 1e-05,
  use_bound_optimization = TRUE,
  use_stepsize_optimization_in_penalizeed_loop = TRUE,
  stepsize_opt_penalized_initial_t = 1, stepsize_opt_penalized_a = 0.1,
  stepsize_opt_penalized_b = 0.1, max_iterations_outer = 1e+05,
  inner_loop_convergence_limit = 1e+05, verbose = TRUE)
```

Arguments

`tolerance_penalized_main_equation_loop`
: tolerance threshold.

`tolerance_penalized_inner_loop_alpha`
: tolerance threshold.

`tolerance_penalized_inner_loop_beta`
: tolerance threshold.

`tolerance_penalized_middel_loop_alpha`
: tolerance threshold.

`tolerance_penalized_outer_loop_alpha`
: tolerance threshold.

`tolerance_penalized_outer_loop_beta`
: tolerance threshold.

`tolerance_penalized_outer_loop_gamma`
: tolerance threshold.

`use_bound_optimization`
: if `TRUE` hessian bound check will be used.

`use_stepsize_optimization_in_penalizeed_loop`
: if `TRUE` step-size optimization will be used.

`stepsize_opt_penalized_initial_t`
: initial step-size.

`stepsize_opt_penalized_a`
: step-size optimization parameter.

`stepsize_opt_penalized_b`
: step-size optimization parameter.

`max_iterations_outer`
: max iteration of outer loop

`inner_loop_convergence_limit`
: inner loop convergence limit.

`verbose`
: If `TRUE` some information, regarding the status of the algorithm, will be printed in the R terminal.

Value

A configuration.

Author(s)

Martin Vincent

Examples

```
data(SimData)

# A quick look at the data
dim(x)
table(classes)

# Create configuration
config <- msgl.algorithm.config(verbose = FALSE)

lambda <- msgl::lambda(x, classes, alpha = .5, d = 50,
 lambda.min = 0.05, algorithm.config = config)

fit <- msgl::fit(x, classes, alpha = .5, lambda = lambda,
 algorithm.config = config)
```

`msgl.c.config` *Featch information about the C side configuration of the package*

Description

Featch information about the C side configuration of the package

Usage

```
msgl.c.config()
```

Value

list

Author(s)

Martin Vincent

---

msgl.standard.config *Standard msgl algorithm configuration*

Description

```
msgl.standard.config <- msgl.algorithm.config()
```

Usage

```
msgl.standard.config
```

Format

An object of class `list` of length 15.

Author(s)

Martin Vincent

---

nmod.msgl *Number of models used for fitting*

Description

Returns the number of models used for fitting. Note that cv and subsampling objects does not containing any models even though nmod returns a positive number.

Usage

```
## S3 method for class 'msgl'
nmod(object, ...)
```

Arguments

`object`	a msgl object
`...`	not used

Value

the number of models in `object`

Author(s)

Martin Vincent

Examples

```
data(SimData)


lambda <- msgl::lambda(x, classes, alpha = .5, d = 50, lambda.min = 0.05)
fit <- msgl::fit(x, classes, alpha = .5, lambda = lambda)

# the number of models
nmod(fit)
```

---

parameters.msgl *Nonzero parameters*

Description

Extracts the nonzero parameters for each model.

Usage

```
## S3 method for class 'msgl'
parameters(object, ...)
```

Arguments

object	a msgl object
...	ignored

Value

a list of length `nmod(x)` containing the nonzero parameters of the models.

Author(s)

Martin Vincent

Examples

```
data(SimData)


lambda <- msgl::lambda(x, classes, alpha = .5, d = 50, lambda.min = 0.05)
fit <- msgl::fit(x, classes, alpha = .5, lambda = lambda)
```

```
# the nonzero parameters of model 1, 10 and 25
parameters(fit)[c(1,10,25)]

# count the number of nonzero parameters in each model
sapply(parameters(fit), sum)
```

`parameters_stat.msgl` *Extracting parameter statistics*

Description

Extracts the number of nonzero parameters in each model.

Usage

```
## S3 method for class 'msgl'
parameters_stat(object, ...)
```

Arguments

`object`	a msgl object
`...`	ignored

Value

a vector of length `nmod(x)` or a matrix containing the number of nonzero parameters of the models.

Author(s)

Martin Vincent

`predict.msgl` *Predict*

Description

Computes the linear predictors, the estimated probabilities and the estimated classes for a new data set.

Usage

```
## S3 method for class 'msgl'
predict(object, x, sparse.data = is(x, "sparseMatrix"),
  ...)
```

Arguments

`object`	an object of class msgl, produced with `msgl`.
`x`	a data matrix of size $N_{\text{new}} \times p$.
`sparse.data`	if TRUE `x` will be treated as sparse, if `x` is a sparse matrix it will be treated as sparse by default.
`...`	ignored.

Value

`link`	the linear predictors – a list of length `length(fit$beta)` one item for each model, with each item a matrix of size $K \times N_{\text{new}}$ containing the linear predictors.
`response`	the estimated probabilities – a list of length `length(fit$beta)` one item for each model, with each item a matrix of size $K \times N_{\text{new}}$ containing the probabilities.
`classes`	the estimated classes – a matrix of size $N_{\text{new}} \times d$ with $d =$`length(fit$beta)`.

Author(s)

Martin Vincent

Examples

```
data(SimData)

x.1 <- x[1:50,]
x.2 <- x[51:100,]

classes.1 <- classes[1:50]
classes.2 <- classes[51:100]

lambda <- msgl::lambda(x.1, classes.1, alpha = .5, d = 50, lambda.min = 0.05)
fit <- msgl::fit(x.1, classes.1, alpha = .5, lambda = lambda)

# Predict classes of new data set x.2
res <- predict(fit, x.2)

# The error rates of the models
Err(res, classes = classes.2)

# The predicted classes for model 20
res$classes[,20]
```

PrimaryCancers *Primary cancer samples.*

Description

Data set consisting of microRNA normalized expression measurements of primary cancer samples.

Format

A design matrix and a class vector

x design matrix

classes class vector

References

http://www.ncbi.nlm.nih.gov/pubmed/24463184

print.msgl *Print function for msgl*

Description

This function will print some general information about the msgl object

Usage

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

Arguments

x	msgl object
...	ignored

Author(s)

Martin Vincent

Examples

```
data(SimData)

### Estimation
lambda <- msgl::lambda(x, classes, alpha = .5, d = 25, lambda.min = 0.075)
fit <- msgl::fit(x, classes, alpha = .5, lambda = lambda)

# Print some information about the estimated models
fit

### Cross validation
fit.cv <- msgl::cv(x, classes, alpha = .5, lambda = lambda)

# Print some information
fit.cv

### Subsampling
test <- list(1:20, 21:40)
train <- lapply(test, function(s) (1:length(classes))[-s])

lambda <- msgl::lambda(x, classes, alpha = .5, d = 50, lambda.min = 0.05)
fit.sub <- msgl::subsampling(x, classes, alpha = .5, lambda = lambda, training = train, test = test)

# Print some information
fit.sub
```

SimData *Simulated data set*

Description

The use of this data set is only intended for testing and examples. The data set contains 100 simulated samples grouped into 10 classes. For each sample 400 features have been simulated.

Format

A design matrix and a class vector

x design matrix

classes class vector ...

subsampling *Multinomial sparse group lasso generic subsampling procedure*

Description

Multinomial sparse group lasso generic subsampling procedure using multiple possessors

Usage

```
subsampling(x, classes, sampleWeights = NULL, grouping = NULL,
  groupWeights = NULL, parameterWeights = NULL, alpha = 0.5,
  standardize = TRUE, lambda, d = 100, training, test,
  intercept = TRUE, sparse.data = is(x, "sparseMatrix"),
  collapse = FALSE, max.threads = NULL, use_parallel = FALSE,
  algorithm.config = msgl.standard.config)
```

Arguments

x	design matrix, matrix of size $N \times p$.
classes	classes, factor of length N.
sampleWeights	sample weights, a vector of length N.
grouping	grouping of features (covariates), a vector of length p. Each element of the vector specifying the group of the feature.
groupWeights	the group weights, a vector of length m (the number of groups). If `groupWeights = NULL` default weights will be used. Default weights are 0 for the intercept and $$\sqrt{K \cdot \text{number of features in the group}}$$ for all other weights.
parameterWeights	a matrix of size $K \times p$. If `parameterWeights = NULL` default weights will be used. Default weights are is 0 for the intercept weights and 1 for all other weights.
alpha	the α value 0 for group lasso, 1 for lasso, between 0 and 1 gives a sparse group lasso penalty.
standardize	if TRUE the features are standardize before fitting the model. The model parameters are returned in the original scale.
lambda	lambda.min relative to lambda.max or the lambda sequence for the regularization path (that is a vector or a list of vectors with the lambda sequence for the subsamples).
d	length of lambda sequence (ignored if `length(lambda) > 1`)
training	a list of training samples, each item of the list corresponding to a subsample. Each item in the list must be a vector with the indices of the training samples for the corresponding subsample. The length of the list must equal the length of the `test` list.

test	a list of test samples, each item of the list corresponding to a subsample. Each item in the list must be vector with the indices of the test samples for the corresponding subsample. The length of the list must equal the length of the training list.
intercept	should the model include intercept parameters
sparse.data	if TRUE x will be treated as sparse, if x is a sparse matrix it will be treated as sparse by default.
collapse	if TRUE the results for each subsample will be collapse into one result (this is useful if the subsamples are not overlapping)
max.threads	Deprecated (will be removed in 2018), instead use use_parallel = TRUE and registre parallel backend (see package 'doParallel'). The maximal number of threads to be used.
use_parallel	If TRUE the foreach loop will use %dopar%. The user must registre the parallel backend.
algorithm.config	the algorithm configuration to be used.

Value

link	the linear predictors – a list of length length(test) with each element of the list another list of length length(lambda) one item for each lambda value, with each item a matrix of size $K \times N$ containing the linear predictors.
response	the estimated probabilities – a list of length length(test) with each element of the list another list of length length(lambda) one item for each lambda value, with each item a matrix of size $K \times N$ containing the probabilities.
classes	the estimated classes – a list of length length(test) with each element of the list a matrix of size $N \times d$ with $d =$length(lambda).
features	number of features used in the models.
parameters	number of parameters used in the models.
classes.true	a list of length length(training), containing the true classes used for estimation

Author(s)

Martin Vincent

Examples

```
data(SimData)

# A quick look at the data
dim(x)
table(classes)

test <- list(1:20, 21:40)
train <- lapply(test, function(s) (1:length(classes))[-s])
```

```
# Run subsampling
# Using a lambda sequence ranging from the maximal lambda to 0.5 * maximal lambda
fit.sub <- msgl::subsampling(x, classes, alpha = 0.5, lambda = 0.5, training = train, test = test)

# Print some information
fit.sub

# Mean misclassification error of the tests
Err(fit.sub)

# Negative log likelihood error
Err(fit.sub, type="loglike")
```

---

x *Design matrix*

Description

Design matrix

Index