Package ‘msgl’

February 20, 2015

Type Package

Title High Dimensional Multiclass Classification Using Sparse Group Lasso

Version 2.0.125.1

Date 2014-11-04

Author Martin Vincent

Maintainer Martin Vincent <vincent@math.ku.dk>

Description Sparse group lasso multiclass classification, suitable for high dimensional problems with many classes. Fast algorithm for solving the multinomial sparse group lasso convex optimization problem. This package apply template metaprogramming techniques, therefore -- when compiling the package from source -- a high level of optimization is needed to gain full speed (e.g. for the GCC compiler use -O3). Use of multiple processors for cross validation and subsampling is supported through OpenMP. The Armadillo C++ library is used as the primary linear algebra engine.

URL http://dx.doi.org/10.1016/j.csda.2013.06.004

License GPL (>= 2)

LazyLoad yes

Depends R (>= 3.0.0), Matrix, sglOptim (>= 1.0.122.0)

LinkingTo Rcpp, RcppProgress, RcppArmadillo, BH, sglOptim

Collate 'msgl_fit.R' 'msgl_cv.R' 'msgl_subsampling.R' 'msgl_predict.R' 'msgl_navigate.R' 'msgl_config.R'

NeedsCompilation yes

Repository CRAN

Date/Publication 2014-11-06 01:05:25
R topics documented:

coef.msgl ................................................................. 2
Err.msgl ................................................................. 3
features.msgl ............................................................ 5
models.msgl ............................................................... 6
msgl .............................................................. 6
msgl.algorithm.config ............................................. 8
msgl.cv ............................................................... 10
msgl.lambda.seq ...................................................... 12
msgl.standard.config ............................................... 13
msgl.subsampling ..................................................... 14
nmod.msgl ............................................................... 16
parameters.msgl ....................................................... 17
predict.msgl ............................................................ 18
print.msgl ............................................................... 19
sim.data ............................................................... 20

Index 21

---

coef.msgl  Extract nonzero coefficients

Description

Extract nonzero coefficients

Usage

```r
## 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)
x <- SimData$x
classes <- SimData$classes
lambda <- msgl.lambda.seq(x, classes, alpha = .5, d = 50, lambda.min = 0.05)
fit <- msgl(x, classes, alpha = .5, lambda = lambda)

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

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 rate or count
... ignored

Value

a vector of error rates

Author(s)

Martin Vincent
Examples

data(SimData)
  x.all <- sim.data$x
  x.1 <- sim.data$x[1:50,]
  x.2 <- sim.data$x[51:100,]
  classes.all <- sim.data$classes
  classes.1 <- sim.data$classes[1:50]
  classes.2 <- sim.data$classes[51:100]

  # Fit models using x.1
  lambda <- msgl.lambda.seq(x.1, classes.1, alpha = .5, d = 50, lambda.min = 0.05)
  fit <- msgl(x.1, classes.1, alpha = .5, lambda = lambda)

  # Training errors:
  err(fit, x.1)
  err(fit, x.1, type = "count")
  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)
  err(fit.cv)
  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)
  err(fit.sub, type = "loglike")
features.msgl

Nonzero features

Description

Extracts the nonzero features for each model.

Usage

```r
## 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 columns of the beta matrices)

Author(s)

Martin Vincent

Examples

```r
data(SimData)
x <- sim.data$x
classes <- sim.data$classes
lambda <- msgl.lambda.seq(x, classes, alpha = .5, d = 50, lambda.min = 0.05)
fit <- msgl(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)
```
models.msgl  

Extract the fitted models

Description

Returns the fitted models, that is the estimated $\beta$ matrices.

Usage

```r
## 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 $\beta$ matrices.

Author(s)

Martin Vincent

msgl  

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

```r
msgl(x, classes,
     sampleWeights = rep(1/length(classes), length(classes)),
     grouping = NULL, groupWeights = NULL,
     parameterWeights = NULL, alpha = 0.5,
     standardize = TRUE, lambda, return = 1:length(lambda),
     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 $\alpha$ 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**: the lambda sequence for the regularization path.
- **return**: the indices of lambda values for which to return a the fitted parameters.
- **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.
- **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)
x <- SimData$x
classes <- SimData$classes
lambda <- msgl.lambda.seq(x, classes, alpha = .5, d = 50, lambda.min = 0.05)
fit <- msgl(x, classes, alpha = .5, lambda = lambda)

# 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

---

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_penalized_loop = TRUE,
stepsize_opt_penalized_initial_t = 1,
stepsize_opt_penalized_a = 0.1,
stepsize_opt_penalized_b = 0.1, 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_middeel_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_penalized_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.
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)
x <- sim.data$x
classes <- sim.data$classes

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

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

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

msgl.cv

Multinomial sparse group lasso cross validation

Description

Multinomial sparse group lasso cross validation using multiple possessors.

Usage

msgl.cv(x, classes, sampleWeights = NULL,
        grouping = NULL, groupWeights = NULL,
        parameterWeights = NULL, alpha = 0.5,
        standardize = TRUE, lambda, fold = 10L,
        cv.indices = list(), intercept = TRUE,
        sparse.data = is(x, "sparseMatrix"), maxthreads = 2L,
        seed = NULL, 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 0 for the intercept weights and 1 for all other weights.
**alpha**
the $\alpha$ 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**
the lambda sequence for the regularization path.

**fold**
the fold of the cross validation, an integer larger than 1 and less than $N + 1$. Ignored if `cv.indices` != NULL. If fold<=$\max(\text{table(classes)})$ 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**
the maximal number of threads to be used

**seed**
deprecated, use `set.seed`.

**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)
x <- sim.data$x
classes <- sim.data$classes

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

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 $\lambda_{\text{max}}$ to the user inputed $\lambda_{\text{min}}$.

Usage

```r
msgl.lambda.seq(x, classes, 
    sampleWeights = rep(1/length(classes)), length(classes)), 
    grouping = NULL, groupWeights = NULL, 
    parameterWeights = NULL, alpha = 0.5, d = 100L, 
    standardize = TRUE, lambda.min, intercept = TRUE, 
    sparse.data = is(x, "sparseMatrix"), 
    algorithm.config = sgl.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 0 for the intercept weights and 1 for all other weights.
- **alpha** the $\alpha$ value 0 for group lasso, 1 for lasso, between 0 and 1 gives a sparse group lasso penalty.
**msgl.standard.config**

- **d**
  - the length of lambda sequence
- **standardize**
  - if TRUE the features are standardized 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.
- **algorithm.config**
  - the algorithm configuration to be used.

**Value**

a vector of length d containing the computed lambda sequence.

**Author(s)**

Martin Vincent

**Examples**

```r
data(simData)
x <- sim.data$x
classes <- sim.data$classes

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

---

**msgl.standard.config**  
*Standard msgl algorithm configuration*

**Description**

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

**Usage**

`msgl.standard.config`

**Format**

List of 13  
- `tolerance_penalized_main_equation_loop` : num 1e-10  
- `tolerance_penalized_inner_loop_alpha` : num 1e-04  
- `tolerance_penalized_inner_loop_beta` : num 1  
- `tolerance_penalized_middel_loop_alpha` : num 0.01  
- `tolerance_penalized_outer_loop_alpha` : num 0.01  
- `tolerance_penalized_outer_loop_beta` : num 0  
- `tolerance_penalized_outer_loop_gamma` : num 1e-05  
- `use_bound_optimization` : logi TRUE  
- `use_stepsize_optimization_in_penalized_loop` : logi TRUE  
- `stepsize_opt_penalized_initial_t` : num 1  
- `stepsize_opt_penalized_a` : num 0.1  
- `stepsize_opt_penalized_b` : num 0.1  
- `verbose` : logi TRUE
msgl.subsampling

Multinomial sparse group lasso generic subsampling procedure

Description

Multinomial sparse group lasso generic subsampling procedure using multiple possessors

Usage

```r
msgl.subsampling(x, classes,
    sampleWeights = rep(1/length(classes), length(classes)),
    grouping = NULL, groupWeights = NULL,
    parameterWeights = NULL, alpha = 0.5,
    standardize = TRUE, lambda, training, test,
    intercept = TRUE, sparse.data = is(x, "sparseMatrix"),
    collapse = FALSE, max.threads = 2L,
    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 0 for the intercept weights and 1 for all other weights.
- `alpha`  
  the $\alpha$ 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`  
  the lambda sequence for the regularization path.
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: The maximal number of threads to be used.

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 $d = \text{length}$(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)
x <- sim.data$x
classes <- sim.data$classes

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

lambda <- msgl.lambda.seq(x, classes, alpha = .5, d = 50, lambda.min = 0.05)
fit.sub <- msgl.subsampling(x, classes, 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")
```

---

**nmod.msgl**

*Returns the number of models in a msgl object*

---

**Description**

Returns the number of models in a msgl object

**Usage**

```r
## 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**

```r
data(SimData)
x <- SimData$x
classes <- SimData$classes
lambda <- msgl.lambda.seq(x, classes, alpha = .5, d = 50, lambda.min = 0.05)
fit <- msgl(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**

```r
## 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**

```r
data(SimData)
x <- SimData$x
classes <- SimData$classes
lambda <- msgl.lambda.seq(x, classes, alpha = .5, d = 50, lambda.min = 0.05)
fit <- msgl(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)
```
**predict.msgl**

**Description**

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

**Usage**

```r
## 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**

```r
data(SimData)
x.1 <- sim.data$x[1:50,]
x.2 <- sim.data$x[51:100,]
classes.1 <- sim.data$classes[1:50]
classes.2 <- sim.data$classes[51:100]
lambda <- msgl.lambda.seq(x.1, classes.1, alpha = .5, d = 50, lambda.min = 0.05)
```
print.msgl

```
fit <- msgl(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]
```

print.msgl  

**Print function for msgl**

### Description

This function will print some general information about the msgl object.

### Usage

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

### Arguments

- **x**: msgl object
- **...**: ignored

### Author(s)

Martin Vincent

### Examples

```r
data(SimData)
x <- sim.data$x
classes <- sim.data$classes

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

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

```r
lambda <- mseg.lambda.seq(x, classes, alpha = .5, d = 50, lambda.min = 0.05)
fit.sub <- mseg.subsampling(x, classes, alpha = .5, lambda = lambda, training = train, test = test)
```

# Print some information

```r
fit.sub
```

---

**sim.data**

### 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.
Index

*Topic datasets
  msgl.standard.config, 13
  sim.data, 20
*Topic data
  sim.data, 20

coeff.msgl, 2
Err.msgl, 3
features.msgl, 5
models.msgl, 6
msgl, 6
msgl.algorithm.config, 8
msgl.cv, 10
msgl.lambda.seq, 12
msgl.standard.config, 13
msgl.subsampling, 14

nmod.msgl, 16

parameters.msgl, 17
predict.msgl, 18
print.msgl, 19

sim.data, 20