Package 'sox'

May 29, 2026

Type Package

Title Structured Learning in Time-Dependent Cox Models

Version 1.2.3

Date 2026-05-29

Description Efficient procedures for fitting and cross-validating the structurally-regularized time-dependent Cox models.

License GPL (>= 3)

Encoding UTF-8

Depends R (>= 3.5.0), survival, glmnet

Imports Rcpp (>= 1.0.10)

LinkingTo Rcpp

Suggests testthat (>= 3.0.0)

Config/testthat/edition 3

RoxygenNote 7.3.3

LazyData true

Copyright file inst/COPYRIGHTS

NeedsCompilation yes

Author Yi Lian [aut, cre],
Guanbo Wang [aut],
Archer Y. Yang [aut],
Mireille E. Schnitzer [aut],
Robert W. Platt [aut],
Rui Wang [aut],
Marc Dorais [aut],
Sylvie Perreault [aut],
Julien Mairal [ctb],
Yuansi Chen [ctb]

Maintainer Yi Lian <yi.lian@mail.mcgill.ca>

Repository CRAN

Date/Publication 2026-05-29 15:20:01 UTC

Contents

`nested_structure` *Automatically generate objects used to describe the structure of the nested group lasso penalty.*

Description

Automatically generate objects used to describe the structure of the nested group lasso penalty. The output is then used by `sox()` and `sox_cv()`.

Usage

```
nested_structure(group_list)
```

Arguments

`group_list`	A list containing the indices of the group members.

Value

A list of objects describing the group structure.

`groups`	Required by `sox()` and `sox_cv()` to describe the relationship between the G `overlapping` groups. A $G * G$ integer matrix whose (i, j) entry is `1` if and only if $i \neq j$ and g_i is a child group (subset) of g_j, and is `0` otherwise.
`own_variables`	Required by `sox()` and `sox_cv()` to describe the relationship between the G `overlapping` groups and the p variables. The entries are the smallest variable indices in the groups (to achieve this, `group` is sorted. For any two groups i and j, if i is the parent group of j, then i is before j and vice versa, otherwise, the one with the smallest variable index is before the other.
`N_own_variables`	Required by `sox()` and `sox_cv()` to describe the relationship between the G `overlapping` groups and the p variables. An integer vector of length G indicating the number of variables that are in each group but not in any of its child groups.
`group_weights`	Required by `sox()` and `sox_cv()` to specify the group-specific penalty weights. The weight is generated in a way such that, the penalty weights of all the groups that contain a given variable sum to 1 for all variables.

Examples

```
# p = 9 Variables:
## 1: A1
## 2: A2
## 3: C1
## 4: C2
## 5: B
## 6: A1B
## 7: A2B
## 8: C1B
## 9: C2B

# G = 12 Nested groups (misspecified, for the demonstration of the software only.)
## g1: A1, A2, C1, C2, B, A1B, A2B, C1B, C2B
## g2: A1B, A2B, A1B, A2B
## g3: C1, C2, C1B, C2B
## g4: 1
## g5: 2
## ...
## G12: 9

nested.groups <- list(1:9,
                      c(1, 2, 6, 7),
                      c(3, 4, 8, 9),
                      1, 2, 3, 4, 5, 6, 7, 8, 9)

pars.nested <- nested_structure(nested.groups)

str(pars.nested)
```

`overlap_structure` *Automatically generate objects used to describe the structure of the overlapping group lasso penalty*

Description

Automatically generate objects used to describe the structure of the overlapping group lasso penalty The output is then used by `sox()` and `sox_cv()`.

Usage

```
overlap_structure(group_list)
```

Arguments

`group_list`	A list containing the indices of the group members.

Value

A list of objects describing the group structure.

`groups`	Required by `sox()` and `sox_cv()` to describe the relationship between the G overlapping groups. A $G * G$ integer matrix whose (i, j) entry is `1` if and only if $i \neq j$ and g_i is a child group (subset) of g_j, and is `0` otherwise.
`groups_var`	Required by `sox()` and `sox_cv()` to describe the relationship between the G overlapping groups and the p variables. A $p * G$ integer matrix whose (i, j) entry is `1` if and only if variable i is in group g_j, but not in any child group of g_j, and is `0` otherwise.
`group_weights`	Required by `sox()` and `sox_cv()` to specify the group-specific penalty weights. The penalty weight for each group is equal to the square root of the group size.

Examples

```
# p = 9 Variables:
## 1: A1
## 2: A2
## 3: C1
## 4: C2
## 5: B
## 6: A1B
## 7: A2B
## 8: C1B
## 9: C2B

# G = 5 Overlapping groups:
## g1: A1, A2, A1B, A2B
## g2: B, A1B, A2B, C1B, C2B
## g3: A1B, A2B
## g4: C1, C2, C1B, C2B
## g5: C1B, C2B

overlapping.groups <- list(c(1, 2, 6, 7),
                           c(5, 6, 7, 8, 9),
                           c(6, 7),
                           c(3, 4, 8, 9),
                           c(8, 9))

pars.overlapping <- overlap_structure(overlapping.groups)

str(pars.overlapping)
```

`plot.sox` *Solution path plot for* `sox()`

Description

Plot the solution path generated by sox().

Usage

```
## S3 method for class 'sox'
plot(x, type = "l", log = "x", ...)
```

Arguments

x	Fitted sox model.
type	Graphical argument to be passed to matplot(), a character string (length 1 vector) or vector of 1-character strings indicating the type of plot for each column of y, see plot.default for all possible types. Default is "l" for lines.
log	Graphical argument to be passed to matplot(), a character string which contains "x" if the x axis is to be logarithmic, "y" if the y axis is to be logarithmic, "" if neither, "xy" or "yx" if both axes are to be logarithmic. Default is "x".
...	Further arguments of matplot() and ultimately of plot.default() for some.

Value

Produces a coefficient profile plot of the coefficient paths for a fitted sox model.

See Also

sox, sox_cv.

Examples

```
x <- as.matrix(sim[, c("A1","A2","C1","C2","B","A1B","A2B","C1B","C2B")])
lam.seq <- exp(seq(log(1e0), log(1e-3), length.out = 20))

overlapping.groups <- list(c(1, 2, 6, 7),
                           c(5, 6, 7, 8, 9),
                           c(6, 7),
                           c(3, 4, 8, 9),
                           c(8, 9))

pars.overlapping <- overlap_structure(overlapping.groups)

fit.overlapping <- sox(
  x = x,
  ID = sim$Id,
  time = sim$Start,
  time2 = sim$Stop,
  event = sim$Event,
  penalty = "overlapping",
  lambda = lam.seq,
  group = pars.overlapping$groups,
  group_variable = pars.overlapping$groups_var,
```

```
  penalty_weights = pars.overlapping$group_weights,
  tol = 1e-4,
  maxit = 1e3,
  verbose = FALSE
)

plot(fit.overlapping)

cv.overlapping <- sox_cv(
  x = x,
  ID = sim$Id,
  time = sim$Start,
  time2 = sim$Stop,
  event = sim$Event,
  penalty = "overlapping",
  lambda = lam.seq,
  group = pars.overlapping$groups,
  group_variable = pars.overlapping$groups_var,
  penalty_weights = pars.overlapping$group_weights,
  nfolds = 5,
  tol = 1e-4,
  maxit = 1e3,
  verbose = FALSE
)

plot(cv.overlapping$sox.fit)
```

`plot.sox_cv` *Plots for* `sox_cv`

Description

Plot the solution path or cross-validation curves produced by `sox_cv()`.

Usage

```
## S3 method for class 'sox_cv'
plot(x, type = "cv-curve", ...)
```

Arguments

`x`	The `sox_cv` object.
`type`	Character string, `"solution-path"` to generate a solution path with marks at `lambda.min` and `lambda.1se`; `"cv-curve"` to generate a cross-validation curve.
`...`	Other graphical parameters to plot

Value

The "solution-path" plot produces a coefficient profile plot of the coefficient paths for a fitted sox model. The "cv-curve" plot is the cvm (red dot) for each lambda with its standard error (vertical bar). The two vertical dashed lines corresponds to the lambda.min and lambda.1se

See Also

sox, sox_cv.

Examples

```
x <- as.matrix(sim[, c("A1","A2","C1","C2","B","A1B","A2B","C1B","C2B")])
lam.seq <- exp(seq(log(1e0), log(1e-3), length.out = 20))

overlapping.groups <- list(c(1, 2, 6, 7),
                           c(5, 6, 7, 8, 9),
                           c(6, 7),
                           c(3, 4, 8, 9),
                           c(8, 9))

pars.overlapping <- overlap_structure(overlapping.groups)

cv.overlapping <- sox_cv(
  x = x,
  ID = sim$Id,
  time = sim$Start,
  time2 = sim$Stop,
  event = sim$Event,
  penalty = "overlapping",
  lambda = lam.seq,
  group = pars.overlapping$groups,
  group_variable = pars.overlapping$groups_var,
  penalty_weights = pars.overlapping$group_weights,
  nfolds = 5,
  tol = 1e-4,
  maxit = 1e3,
  verbose = FALSE
)

plot(cv.overlapping)
plot(cv.overlapping, type = "solution-path")
```

---

sim *A simulated demo dataset* sim

Description

A simulated demo dataset sim

Usage

```
data(sim)
```

Format

A simulated data frame that is used to illustrate the use of the sox package. The max follow-up time for each subject is set to be 5. The total number of subject is 50.

Id The ID of each subject.

Event During the time from `Start` to `Stop`, if the subject experience the event. We use the function `permalgorithm` in the `R` package `PermAlgo` to generate the Event.

Start Start time.

Stop Stop time.

Fup The total follow-up time for the subject.

Covariates A1, A2, C1, C2, B, A1B, A2B, C1B, C2B. The dataset contains 5 variables (9 columns after one-hot encoding). Variable A is a e 3-level categorical variable, which results in 2 binary variables (A1 and A2), the same with the variable C. B is a continuous variable. The interaction term AB and CB are also two 3-level categorical variables. The code for generating the covariates is given below.

See Also

`PermAlgo`

Examples

```
# generate B
gen_con=function(m){
 X=rnorm(m/5)
 XX=NULL
 for (i in 1:length(X)) {
   if (length(XX)<m){
   X.rep=rep(X[i],round(runif(1,5,10),0))
   XX=c(XX,X.rep)
   }
 }
 return(XX[1:m])
}
# generate A and C
gen_cat=function(m){
 X=sample.int(3, m/5,replace = TRUE)
 XX=NULL
 for (i in 1:length(X)) {
   if (length(XX)<m){
     X.rep=rep(X[i],round(runif(1,5,10),0))
     XX=c(XX,X.rep)
   }
 }
 return(XX[1:m])
```

```
}

# generate covariate for one subject
gen_X=function(m){
 A=gen_cat(m);B=gen_con(m);C=gen_cat(m)
 A1=ifelse(A==1,1,0);A2=ifelse(A==2,1,0)
 C1=ifelse(C==1,1,0);C2=ifelse(C==2,1,0)
 A1B=A1*B;A2B=A2*B
 C1B=C1*B;C2B=C2*B
 return(as.matrix(cbind(A1,A2,C1,C2,B,A1B,A2B,C1B,C2B)))
}

# generate covariate for all subject
gen_X_n=function(m,n){
 Xn=NULL
 for (i in 1:n) {
   X=gen_X(m)
   Xn=rbind(Xn,X)
 }
 return(Xn)
}

 n=50;m=5
 covariates=gen_X_n(m,n)
 # generate outcomes
 # library(PermAlgo)
 # data <- permalgorithm(n, m, covariates,
 #                       XmatNames = c("A1","A2","C1","C2","B","A1B","A2B","C1B","C2B"),
 #                       #change according to scenario 1/2
 #                     betas = c(rep(log(3),2),rep(0,2), log(4), rep(log(3),2),rep(0,2)),
 #                       groupByD=FALSE )
 # fit.original = coxph(Surv(Start, Stop, Event) ~ . ,data[,-c(1,3)])
```

---

sox *(Time-dependent) Cox model with structured variable selection*

Description

Fit a (time-dependent) Cox model with overlapping (including nested) group lasso penalty. The regularization path is computed at a grid of values for the regularization parameter lambda.

Usage

```
sox(
  x,
  ID,
  time,
  time2,
  event,
  penalty,
```

```
  lambda,
  group,
  group_variable,
  own_variable,
  no_own_variable,
  penalty_weights,
  par_init,
  stepsize_init = 1,
  stepsize_shrink = 0.8,
  tol = 1e-05,
  maxit = 1000L,
  verbose = FALSE
)
```

Arguments

x	Predictor matrix with dimension $nm*p$, where n is the number of subjects, m is the maximum observation time, and p is the number of predictors. See Details.
ID	The ID of each subjects, each subject has one ID (multiple rows in x can share one ID).
time	Represents the start of each time interval.
time2	Represents the stop of each time interval.
event	Indicator of event. event = 1 when event occurs and event = 0 otherwise.
penalty	Character string, indicating whether "overlapping" or "nested" group lasso penalty is imposed.
lambda	Sequence of regularization coefficients λ's.
group	A $G * G$ integer matrix required to describe the structure of the overlapping and nested groups. We recommend that the users generate it automatically using overlap_structure() and nested_structure(). See Examples and Details.
group_variable	A $p * G$ integer matrix required to describe the structure of the overlapping groups. We recommend that the users generate it automatically using overlap_structure(). See Examples and Details.
own_variable	A non-decreasing integer vector of length G required to describe the structure of the nested groups. We recommend that the users generate it automatically using nested_structure(). See Examples and Details.
no_own_variable	An integer vector of length G required to describe the structure of the nested groups. We recommend that the users generate it automatically using nested_structure(). See Examples and Details
penalty_weights	Optional, vector of length G specifying the group-specific penalty weights. We recommend that the users generate it automatically using overlap_structure() or nested_structure(). If not specified, $\mathbf{1}_G$ is used.
par_init	Optional, vector of initial values of the optimization algorithm. Default initial value is zero for all p variables.

stepsize_init	Initial value of the stepsize of the optimization algorithm. Default is 1.0.
stepsize_shrink	Factor in $(0, 1)$ by which the stepsize shrinks in the backtracking linesearch. Default is 0.8.
tol	Convergence criterion. Algorithm stops when the l_2 norm of the difference between two consecutive updates is smaller than tol.
maxit	Maximum number of iterations allowed.
verbose	Logical, whether progress is printed.

Details

The predictor matrix should be of dimension $nm * p$. Each row records the values of covariates for one subject at one time, for example, the values at the day from time (Start) to time2 (Stop). An example dataset sim is provided. The dataset has the format produced by the R package **PermAlgo**. The specification of the arguments group, group_variable, own_variable and no_own_variable for the grouping structure can be found in https://thoth.inrialpes.fr/people/mairal/spams/doc-R/html/doc_spams006.html#sec26 and https://thoth.inrialpes.fr/people/mairal/spams/doc-R/html/doc_spams006.html#sec27.

In the Examples below, $p = 9, G = 5$, the group structure is:

$$g_1 = \{A_1, A_2, A_1B, A_2B\},$$

$$g_2 = \{B, A_1B, A_2B, C_1B, C_2B\},$$

$$g_3 = \{A_1B, A_2B\},$$

$$g_4 = \{C_1, C_2, C_1B, C_2B\},$$

$$g_5 = \{C_1B, C_2B\}.$$

where g_3 is a subset of g_1 and g_2, and g_5 is a subset of g_2 and g_4.

Value

A list with the following three elements.

lambdas	The user-specified regularization coefficients lambda sorted in decreasing order.
estimates	A matrix, with each column corresponding to the coefficient estimates at each λ in lambdas.
iterations	A vector of number of iterations it takes to converge at each λ in lambdas.

Examples

```
x <- as.matrix(sim[, c("A1","A2","C1","C2","B","A1B","A2B","C1B","C2B")])
lam.seq <- exp(seq(log(1e0), log(1e-3), length.out = 20))

# Variables:
## 1: A1
## 2: A2
## 3: C1
```

```
## 4: C2
## 5: B
## 6: A1B
## 7: A2B
## 8: C1B
## 9: C2B

# Overlapping groups:
## g1: A1, A2, A1B, A2B
## g2: B, A1B, A2B, C1B, C2B
## g3: A1B, A2B
## g4: C1, C2, C1B, C2B
## g5: C1B, C2B

overlapping.groups <- list(c(1, 2, 6, 7),
                           c(5, 6, 7, 8, 9),
                           c(6, 7),
                           c(3, 4, 8, 9),
                           c(8, 9))

pars.overlapping <- overlap_structure(overlapping.groups)

fit.overlapping <- sox(
  x = x,
  ID = sim$Id,
  time = sim$Start,
  time2 = sim$Stop,
  event = sim$Event,
  penalty = "overlapping",
  lambda = lam.seq,
  group = pars.overlapping$groups,
  group_variable = pars.overlapping$groups_var,
  penalty_weights = pars.overlapping$group_weights,
  tol = 1e-4,
  maxit = 1e3,
  verbose = FALSE
)

str(fit.overlapping)

# Nested groups (misspecified, for the demonstration of the software only.)
## g1: A1, A2, C1, C2, B, A1B, A2B, C1B, C2B
## g2: A1B, A2B, A1B, A2B
## g3: C1, C2, C1B, C2B
## g4: 1
## g5: 2
## ...
## G12: 9

nested.groups <- list(1:9,
                      c(1, 2, 6, 7),
                      c(3, 4, 8, 9),
                      1, 2, 3, 4, 5, 6, 7, 8, 9)
```

```
pars.nested <- nested_structure(nested.groups)

fit.nested <- sox(
  x = x,
  ID = sim$Id,
  time = sim$Start,
  time2 = sim$Stop,
  event = sim$Event,
  penalty = "nested",
  lambda = lam.seq,
  group = pars.nested$groups,
  own_variable = pars.nested$own_variables,
  no_own_variable = pars.nested$N_own_variables,
  penalty_weights = pars.nested$group_weights,
  tol = 1e-4,
  maxit = 1e3,
  verbose = FALSE
)

str(fit.nested)
```

---

sox_cv *cross-validation for* sox

Description

Conduct cross-validation (cv) for sox.

Usage

```
sox_cv(
  x,
  ID,
  time,
  time2,
  event,
  penalty,
  lambda,
  group,
  group_variable,
  own_variable,
  no_own_variable,
  penalty_weights,
  par_init,
  nfolds = 10,
  foldid = NULL,
  stepsize_init = 1,
```

```
  stepsize_shrink = 0.8,
  tol = 1e-05,
  maxit = 1000L,
  verbose = FALSE
)
```

Arguments

x	Predictor matrix with dimension $nm * p$, where n is the number of subjects, m is the maximum observation time, and p is the number of predictors. See Details.
ID	The ID of each subjects, each subject has one ID (multiple rows in x can share one ID).
time	Represents the start of each time interval.
time2	Represents the stop of each time interval.
event	Indicator of event. event = 1 when event occurs and event = 0 otherwise.
penalty	Character string, indicating whether "overlapping" or "nested" group lasso penalty is imposed.
lambda	Sequence of regularization coefficients λ's.
group	A $G * G$ integer matrix required to describe the structure of the overlapping and nested groups. We recommend that the users generate it automatically using overlap_structure() and nested_structure(). See Examples and Details.
group_variable	A $p * G$ integer matrix required to describe the structure of the overlapping groups. We recommend that the users generate it automatically using overlap_structure(). See Examples and Details.
own_variable	A non-decreasing integer vector of length G required to describe the structure of the nested groups. We recommend that the users generate it automatically using nested_structure(). See Examples and Details.
no_own_variable	An integer vector of length G required to describe the structure of the nested groups. We recommend that the users generate it automatically using nested_structure(). See Examples and Details
penalty_weights	Optional, vector of length G specifying the group-specific penalty weights. We recommend that the users generate it automatically using overlap_structure() or nested_structure(). If not specified, $\mathbf{1}_G$ is used.
par_init	Optional, vector of initial values of the optimization algorithm. Default initial value is zero for all p variables.
nfolds	Optional, the folds of cross-validation. Default is 10.
foldid	Optional, user-specified vector indicating the cross-validation fold in which each observation should be included. Values in this vector should range from 1 to nfolds. If left unspecified, sox will randomly assign observations to folds
stepsize_init	Initial value of the stepsize of the optimization algorithm. Default is 1.

`stepsize_shrink`
: Factor in $(0, 1)$ by which the stepsize shrinks in the backtracking linesearch. Default is 0.8.

`tol`
: Convergence criterion. Algorithm stops when the l_2 norm of the difference between two consecutive updates is smaller than `tol`.

`maxit`
: Maximum number of iterations allowed.

`verbose`
: Logical, whether progress is printed.

Details

For each lambda, 10 folds cross-validation (by default) is performed. The cv error is defined as follows. Suppose we perform K-fold cross-validation, denote $\hat{\beta}^{-k}$ by the estimate obtained from the rest of $K - 1$ folds (training set). The error of the k-th fold (test set) is defined as $2(P - Q)$ divided by R, where P is the log partial likelihood evaluated at $\hat{\beta}^{-k}$ using the entire dataset, Q is the log partial likelihood evaluated at $\hat{\beta}^{-k}$ using the training set, and R is the number of events in the test set. We do not use the negative log partial likelihood evaluated at $\hat{\beta}^{-k}$ using the test set because the former definition can efficiently use the risk set, and thus it is more stable when the number of events in each test set is small (think of leave-one-out). The cv error is used in parameter tuning. To account for balance in outcomes among the randomly formed test set, we divide the deviance $2(P - Q)$ by R. To get the estimated coefficients that has the minimum cv error, use `sox_cv()$Estimates[, sox_cv$index["min",]]`. To apply the 1-se rule, use `sox_cv()$Estimates[, sox_cv$index["1se",]]`.

Value

A list.

`lambdas`
: A vector of lambda used for each cross-validation.

`cvm`
: The cv error averaged across all folds for each lambda.

`cvsd`
: The standard error of the cv error for each lambda.

`cvup`
: The cv error plus its standard error for each lambda.

`cvlo`
: The cv error minus its standard error for each lambda.

`nzero`
: The number of non-zero coefficients at each lambda.

`sox.fit`
: A fitted model for the full data at all lambdas of class "`sox`".

`lambda.min`
: The lambda such that the `cvm` reach its minimum.

`lambda.1se`
: The maximum of lambda such that the `cvm` is less than the minimum the `cvup` (the minmum of `cvm` plus its standard error).

`foldid`
: The fold assignments used.

`index`
: A one column matrix with the indices of `lambda.min` and `lambda.1se`.

`iterations`
: A vector of number of iterations it takes to converge at each λ in `lambdas`.

See Also

`sox`, `plot.sox_cv`.

Examples

```
x <- as.matrix(sim[, c("A1","A2","C1","C2","B","A1B","A2B","C1B","C2B")])
lam.seq <- exp(seq(log(1e0), log(1e-3), length.out = 20))

# Variables:
## 1: A1
## 2: A2
## 3: C1
## 4: C2
## 5: B
## 6: A1B
## 7: A2B
## 8: C1B
## 9: C2B

# Overlapping groups:
## g1: A1, A2, A1B, A2B
## g2: B, A1B, A2B, C1B, C2B
## g3: A1B, A2B
## g4: C1, C2, C1B, C2B
## g5: C1B, C2B

overlapping.groups <- list(c(1, 2, 6, 7),
                           c(5, 6, 7, 8, 9),
                           c(6, 7),
                           c(3, 4, 8, 9),
                           c(8, 9))

pars.overlapping <- overlap_structure(overlapping.groups)

cv.overlapping <- sox_cv(
  x = x,
  ID = sim$Id,
  time = sim$Start,
  time2 = sim$Stop,
  event = sim$Event,
  penalty = "overlapping",
  lambda = lam.seq,
  group = pars.overlapping$groups,
  group_variable = pars.overlapping$groups_var,
  penalty_weights = pars.overlapping$group_weights,
  nfolds = 5,
  tol = 1e-4,
  maxit = 1e3,
  verbose = FALSE
)

str(cv.overlapping)

# Nested groups (misspecified, for the demonstration of the software only.)
## g1: A1, A2, C1, C2, B, A1B, A2B, C1B, C2B
## g2: A1B, A2B, A1B, A2B
```

```
## g3: C1, C2, C1B, C2B
## g4: 1
## g5: 2
## ...
## G12: 9

nested.groups <- list(1:9,
                      c(1, 2, 6, 7),
                      c(3, 4, 8, 9),
                      1, 2, 3, 4, 5, 6, 7, 8, 9)

pars.nested <- nested_structure(nested.groups)

cv.nested <- sox_cv(
  x = x,
  ID = sim$Id,
  time = sim$Start,
  time2 = sim$Stop,
  event = sim$Event,
  penalty = "nested",
  lambda = lam.seq,
  group = pars.nested$groups,
  own_variable = pars.nested$own_variables,
  no_own_variable = pars.nested$N_own_variables,
  penalty_weights = pars.nested$group_weights,
  nfolds = 5,
  tol = 1e-4,
  maxit = 1e3,
  verbose = FALSE
)

str(cv.nested)
```

Index