Package ‘sprinter’

February 20, 2015

Type Package
Title Framework for Screening Prognostic Interactions
Version 1.1.0
Date 2014-04-11
Author Isabell Hoffmann
Maintainer Isabell Hoffmann <Isabell.Hoffmann@uni-mainz.de>
VignetteBuilder knitr
Depends R (>= 3.0.0), CoxBoost, survival
Imports GAMBoost, randomForestSRC, LogicReg
Suggests parallel, knitr
Description The main function of this package builds prognostic models that consider interactions by combining available statistical components. Furthermore, it provides a function for evaluating the relevance of the selected interactions by resampling techniques.
License GPL-2
NeedsCompilation no
Repository CRAN
Date/Publication 2014-12-18 15:56:40

R topics documented:

plot.resample.sprinter ........................................... 2
predict.sprinter .................................................. 2
resample.sprinter ............................................... 4
screen.inter ...................................................... 7
screen.main.cox .................................................. 9
screen.main.glm ................................................ 12
simul.int .......................................................... 14
sprinter ............................................................ 16
summary.resample.sprinter ................................. 22
summary.sprinter ................................................ 23

Index 24
plot.resample.sprinter

Plots coefficients generated by resample.sprinter

Description

Plots for each variable which is selected in resample.sprinter the mean coefficient over all sub-samples against its single coefficients.

Usage

```r
## S3 method for class 'resample.sprinter'
plot(x, optional.only = FALSE, threshold.vif = 0,...)
```

Arguments

- `x` resample.sprinter object from a resample.sprinter call.
- `optional.only` logical value. If true, only the coefficients of the optional variables are plotted. If false, the coefficients of the mandatory variables are plotted as well.
- `threshold.vif` value. Only the coefficients with inclusion frequencies larger than threshold.vif are plotted.
- `...` additional arguments.

Value

No value is returned, but a plot is generated.

Author(s)

Isabell Hoffmann <isabell.hoffmann@uni-mainz.de>

predict.sprinter

Predict method for objects of class sprinter

Description

Evaluates the linear predictor from a Cox proportional Hazards model fitted by sprinter.

Usage

```r
## S3 method for class 'sprinter'
predict(object, newdata=NULL,...)
```
Arguments

object  Cox proportional Hazards model from a sprinter call.
newdata  n.new * p matrix with new covariate values. If just prediction for the training data is wanted, it can be omitted.
...  additional arguments.

Value

The linear predictor, a vector of length n.new, is returned.

Author(s)

Isabell Hoffmann <isabell.hoffmann@uni-mainz.de>

Examples

```r
simulation <- simul.int(287578, n = 200, p = 500,
  beta.int = 1.0,
  beta.main = 0.9,
  censparam = 1/20,
  lambda = 1/20)
data <- simulation$data
simulation$info

set.seed(123)

## Not run:
testcb <- sprinter(x=data[,,:500],
  time = data$obs.time,
  status= data$obs.status,
  repetitions = 10,
  mandatory = c("ID1","ID2"),
  n.inter.candidates = 1000,
  screen.main = fit.CoxBoost,
  fit.final = fit.CoxBoost,
  args.screen.main = list(seed=123,stepno = 10, K = 10,
    criterion = 'pscore', nu = 0.05),
  parallel = FALSE, trace=TRUE)
summary(testcb)

# true coefficients:
# Clin.cov1  Clin.cov2  ID5:ID6  ID7:ID8
# 0.9       -0.9       1       -1

# Simulate New Data:
newSimulation <- simul.int(12345,n = 200, p = 500,
  beta.int = 1.0,
  beta.main = 0.9,
  censparam = 1/20,
  lambda = 1/20)
```
resample.sprinter

Wrapper for the function sprinter in order to judge the stability of the selected variables

Description

resample.sprinter is a wrapper, which subsamples the data and performs the function sprinter on each subsample. It evaluates the interaction terms by an improvement in prediction performance. The importance of an interaction is judged by variable inclusion frequencies and mean coefficients for each interaction term.

Usage

resample.sprinter(x, time, status, fold = 5, oob.rel = 0.632, mandatory, repetitions = 25, n.inter.candidates = 1000, screen.main, screen.inter = fit.rf, fit.final, args.screen.main = list(), args.screen.inter = list(), args.fit.final = args.screen.main, orthogonalize = TRUE, parallel = FALSE, mc.cores = detectCores(), ...)

Arguments

x n * p matrix of covariates.
time vector of length n specifying the observed times.
status censoring indicator, i.e., vector of length n with entries 0 for censored observations and 1 for uncensored observations.
fold number of subsamples. Default is 5.
oob.rel frequency of subsamples.
mandatory vector with variable names of mandatory covariates, where parameter estimation should be performed unpenalized.
resample.sprinter

repetitions number of repetitions of the interaction screening approach. Repetitions are performed by creating subsamples and applying the interaction screening approach on each subsample dataset separately.

n.inter.candidates minimal number of potential interaction candidates, which are considered in the final model building step.

screen.main function for screening potential main effects. See more in details.

screen.inter function for detecting potential interaction candidates. See more in details.

fit.final function for building final Cox proportional hazards model. Default is the function set in screen.main. See more in details.

args.screen.main list of arguments which should be used in the main effects detection step.

args.screen.inter list of arguments which should be used in the interaction screening step.

args.fit.final list of arguments which should be used in the final model building step.

orthogonalize logical value. If true all variables are made orthogonal to those that are assessed as main effects by screen.main.

parallel logical value indicating whether the interaction screening step should be performed parallel.

mc.cores the numbers of cores to use.
... additional arguments.

Details

resample.sprinter provides the possibility of evaluating the relevance of an interaction by using resampling techniques and the resultant variable inclusion frequency. This function subsamples the data and applies the function sprinter on each subsample. As a result the user can print the interactions with the largest inclusion frequencies and their mean coefficients.

Value

Returns a list of length fold of objects produced by sprinter.

Author(s)

Written by Isabell Hoffmann <isabell.hoffmann@uni-mainz.de>.

References


See Also

sprinter
Examples

# Not run:
# Create survival data with interactions:
data <- simul.int(518, n = 200, p = 500,
   beta.int = 1.0,
   beta.main = 0.9,
   censparam = 1/20,
   lambda = 1/20)[[1]]
set.seed(123)

resamcb <- resample.sprinter(  
x=data[,1:500],  
time = data$obs.time,  
status= data$obs.status,  
fold = 10,  
repetitions = 5,  
mandatory = c("ID1","ID2"),  
n.inter.candidates = 1000,  
screen.main = fit.CoxBoost,  
fit.final = fit.CoxBoost,  
args.screen.main = list(stepno = 5),  
parallel = F)

summary(resamcb)
summary(resamcb, optional = T)

## End(Not run)
# true coefficients:
# Clin.cov1    Clin.cov2    ID5:ID6    ID7:ID8
#  0.9       -0.9        1        -1

# Not run:
# Create survival data with interactions:
data <- simul.int(518, n = 200, p = 500,
screen.inter

beta.int = 1.0,  
beta.main = 0.9,  
censparam = 1/20,  
lambda = 1/20)##

resamunicox <- resample.sprinter(  
x=data[,1:500],  
time = data$obs.time,  
status= data$obs.status,  
fold = 10,  
repetitions = 10,  
mandatory = c("ID1","ID2"),  
n.inter.candidates = 1000,  
screen.main = fit.uniCox,  
fit.final = fit.uniCox,  
parallel = FALSE)

summary(resamunicox)
summary(resamunicox, optional = TRUE)

### End(Not run)

## screen.inter  Adaptive function for screening interactions

### Description

fit.logicReg and fit.rf are functions for screening interactions in high-dimensional datasets for the usage in the argument screen.inter in the function sprinter. They return a variable importance measurement for each variable.

### Usage

fit.rf(nr, data, indices, seed.interselect, ...) 

fit.rf.select(nr, data, indices, seed.interselect, n.select, ...)

fit.logicReg(nr, data, indices, seed.interselect,  
type,  
nleaves,  
ntrees, ...)

fit.logicReg.select(nr, data, indices, seed.interselect,  
type,  
nleaves,  
ntrees,  
n.select,...)
Arguments

nr: number of resample run.
data: data frame containing the y-outcome and x-variables in the model, which is orthogonalized to the clinical covariates and the main effects identified in the main effects detection step.
indices: indices to build the resample dataset.
seed.interselect: seed for random number generator.
n.select: Number of variables selected for performing random forest.
type: type of model to be fit. For survival data you can choose between (4) proportional hazards model (Cox regression), and (5) exponential survival model, or (0) your own scoring function.
n.leaves: maximum number of leaves to be fit in all trees combined.
n.trees: number of logic trees to be fit.
...: further arguments passed to methods.

Details

The functions logicReg and fit.rf are adapted for the usage in the function sprinter in order to screen interactions. Therein, variable importance measurements are evaluated for each variable, which will be used for pre-selecting relevant interactions in the function sprinter. In the function sprinter the identified interaction candidates will be combined with each other pairwise and will be provided as possible predictors for the final model.

fit.rf: This function performs a random forest for survival. It judges each variable by the permutation accuracy importance. For more information about performing the random forest see rfsrc.

fit.rf.select: This function performs a random forest for survival on a restricted data set. The number of covariables in this restricted data set can be set in n.select. The variables with the n.select smallest univariate p-values evaluated by Cox regression are selected.

fit.logicReg: For the usage of the logic regression all continuous variables are converted to binary variables at the median. Then the logic regression is fitted onto the binary data set. The variable importance measure is one, if the variable is included in the model and zero if not. In order to get the information about the variables in a multiple model, the set select = 2 is obligatory.

fit.logicReg.select: This function performs logic regression on a restricted data set. The number of covariables in this restricted data set can be set in n.select. The variables with the n.select smallest univariate p-values evaluated by Cox regression are selected.

Implementing new functions for the argument screen.inter: New functions for screening interactions can be constructed in a way that for each variable an importance measurement is returned as a vector of length p. The variable importance measurements larger than zero should be interpreted as relevant for the model.

The following arguments must be enclosed in this function:
nr  value displaying the actual resampling run.
data  data frame containing the y-outcome and x-variables in the model.
indices  indices to build the resample dataset.
seed.interselect  seed for random number generator.

With this directive other functions can be implemented and used for screening potential interaction candidates.

Value

fit.rf and fit.logicReg return a vector of length p, containing the variable importance of each variable in the data set.

fit.rf evaluates the permutation accuracy importance (PAM) as a measure for the variable importance. The function fit.logicReg returns the information whether a variable is enclosed in the model (1) or not (0).

Author(s)

Written by Isabell Hoffmann <isabell.hoffmann@uni-mainz.de>.

References


See Also

logreg, rfsrc

screen.main.cox  Adaptive function for screening main effects and performing a Cox model

Description

The adaptive function selects the most important main effects and performs a Cox proportional hazards model for the usage in the argument screen.main in the function sprinter. For time-to-event data two methods are available for the screening step: Either a Cox proportional Hazards model with variable selection by adjusted univariate p-values (fit.uniCox) can be performed or a model is build via CoxBoost (fit.CoxBoost).
Usage

\[
\text{fit uniCox}(\text{time, status, } x, \text{ unpen.index = NULL, method = 'bonferroni', sig = 0.05, ...})
\]

\[
\text{fit CoxBoost}(\text{time, status, } x, \text{ unpen.index = NULL, seed = 123, stepno = NULL, K = 10, criterion = 'pscore', nu = 0.05, maxstepno=200, standardize = T, trace = T, ...})
\]

Arguments

- **time**: vector of length n specifying the observed times.
- **status**: censoring indicator, i.e., vector of length n with entries 0 for censored observations and 1 for uncensored observations.
- **x**: n * p matrix of covariates.
- **unpen.index**: vector with indices of mandatory covariates, where parameter estimation should be performed unpenalized.
- **seed**: Seed for random number generator.
- **stepno**: number of boosting steps.
- **K**: number of folds to be used for cross-validation. If K is larger or equal to the number of non-zero elements in status, leave-one-out cross-validation is performed.
- **criterion**: indicates the criterion to be used for selection in each boosting step. "pscore" corresponds to the penalized score statistics, "score" to the un-penalized score statistics. Different results will only be seen for un-standardized covariates ("pscore" will result in preferential selection of covariates with larger covariance), or if different penalties are used for different covariates. "hpiscore" and "hpscore" correspond to "pscore" and "score". However, a heuristic is used for evaluating only a subset of covariates in each boosting step, as described in Binder et al. (2011). This can considerably speed up computation, but may lead to different results.
- **nu**: value for evaluating the penalty for the update of an individual element of the parameter vector in each boosting step (penalty = sum(status)*(1/nu-1)).
- **maxstepno**: maximum number of boosting steps to evaluate, i.e, the returned "optimal" number of boosting steps will be in the range [0,maxstepno].
- **standardize**: logical value indicating whether covariates should be standardized for estimation. This does not apply for mandatory covariates, i.e., these are not standardized.
trace logical value indicating whether progress in estimation should be indicated by printing the name of the covariate updated.

method method for adjusting p-values. A variable is selected if its adjusted p-value is less than sig.

sig selection level. A variable is selected if its adjusted p-value is less than sig

... further arguments passed to methods.

Details

fit.uniCox and fit.CoxBoost are adapted functions for the usage of screening main effects in the function sprinter for time-to-event data. Both approaches fit a multivariate Cox proportional hazards model.

fit.uniCox: fit.uniCox fits a multivariate Cox proportional hazards model after a variable selection step. Therefore univariate Cox proportional hazard models are performed for each variable. All variables with univariate adjusted p-values less than sig are selected for the multivariate model. The variables with indices unpen.index are mandatory for the multivariate Cox model.

fit.CoxBoost: fit.CoxBoost fits a Cox proportional hazards model by using CoxBoost. If the number of boosting steps is not specified in stepno the step number is evaluated by cross validation. See more information about fitting a Cox proportional hazards model by CoxBoost in CoxBoost.

Implementing new functions for the argument screen.main: New functions for screening potential main effects can be implemented by the user. Therefore, the function must be constructed in a way that a Cox proportional Hazards model is generated. This model should be returned together with the names of the variables used in the model, called xnames, and their corresponding indices, called indmain.

The following arguments must be enclosed in this function:

time vector of length n specifying the observed times.

status censoring indicator, i.e., vector of length n with entries 0 for censored observations and 1 for uncensored observations.

x n * p matrix of covariates.

unpen.index vector with indices of mandatory covariates.

With this instructions the user can create new functions for screening main effects. To set further arguments for this function create a list of arguments, which can be quoted in args.screen.main. In the next step the screened main effects will be used for orthogonalizing the data by computing residuals corresponding to the selected main effects and the mandatory covariates.

Value

The adaptive function returns the following values:

model Cox proportional Hazards model. fit.uniCox returns an object of class coxph and fit.CoxBoost returns an object of class CoxBoost.

xnames vector of length p containing the names of the covariates.
indmain vector of length p containing the indices of the selected covariates.

beta vector of length p containing the coefficients of the selected covariates.

Author(s)

Written by Isabell Hoffmann <isabell.hoffmann@uni-mainz.de>.

References


See Also

p.adjust, coxph, CoxBoost

screen.main.glm

Adaptive function for screening main effects and fitting a generalized linear model

Description

The adaptive function selects the most important main effects and fits a generalized linear model for the usage in the argument screen.main in the function sprinter. Two methods are available for this screening step: Either a generalized linear model with variable selection by adjusted univariate p-values (fit.uniglm) can be performed or a model is build via GAMBoost (fit.GAMboost).

Usage

fit.GAMboost(time, status, x, unpen.index = NULL,
seed = 123,
stepno = NULL,
penalty = 100,
maxstepno = 200,
standardize = T,
criterion = 'deviance',
family = gaussian(),
trace = T, ...)
fit.uniglm(time, status, x, unpen.index = NULL,
method = 'bonferroni',
family = gaussian(),
sig = 0.05, ...)
Arguments

- `time`: vector of length n specifying the response.
- `status`: censoring indicator. These functions are not constructed for time-to-event data. Therefore, all entities of this vector are zero.
- `x`: n * p matrix of covariates.
- `unpen.index`: vector with indices of mandatory covariates, where parameter estimation should be performed unpenalized.
- `seed`: Seed for random number generator.
- `stepno`: number of boosting steps.
- `criterion`: indicates the criterion to be used for selection in each boosting step. "pscore" corresponds to the penalized score statistics, "score" to the un-penalized score statistics. Different results will only be seen for un-standardized covariates ("pscore" will result in preferential selection of covariates with larger covariance), or if different penalties are used for different covariates. "hpscore" and "hscore" correspond to "pscore" and "score". However, a heuristic is used for evaluating only a subset of covariates in each boosting step, as described in Binder et al. (2011). This can considerably speed up computation, but may lead to different results.
- `maxstepno`: maximum number of boosting steps to evaluate, i.e, the returned "optimal" number of boosting steps will be in the range [0,maxstepno].
- `standardize`: logical value indicating whether covariates should be standardized for estimation. This does not apply for mandatory covariates, i.e., these are not standardized.
- `penalty`: penalty value for the update of an individual smooth function in each boosting step.
- `family`: a description of the error distribution. This can be a character string naming a family function, a family function or the result of a call to a family function.
- `trace`: logical value indicating whether progress in estimation should be indicated by printing the name of the covariate updated.
- `method`: method for adjusting p-values. A variable is selected if its adjusted p-value is less than `sig`.
- `sig`: selection level. A variable is selected if its adjusted p-value is less than `sig`.
- `...`: further arguments passed to methods.

Details

`fit.uniGlm` and `fit.GAMBoost` are adapted as functions for the usage of screening main effects in the function `sprinter`. Both approaches fit a generalized linear model.

**fit.uniGlm**: `fit.uniCox` fits a generalized linear regression model after a variable selection step. Therefore univariate regression models are performed for each variable. All variables with univariate adjusted p-values less than `sig` are selected for the multivariate model. The variables with indices `unpen.index` are mandatory for the multivariate regression model.
**fit.GAMBoost:** *fit.GAMBoost* fits a generalized regression model by using GAMBoost. If the number of boosting steps is not specified in `stepno` the step number is evaluated by cross validation. See more information about fitting a generalized regression model by GAMBoost in `GAMBoost`.

**Implementing new functions for the argument screen.main:** New functions for screening potential main effects can be implemented by the user. Therefore, the function must be constructed in a way that a generalized linear model is generated. This model should be returned together with the names of the variables used in the model, called `xnames`, and their corresponding indices, called `indmain`.

The following arguments must be enclosed in this function:

- `time`: vector of length n specifying the outcome.
- `status`: censoring indicator, which is always NULL in the case fitting a generalized linear models.
- `x`: n * p matrix of covariates.
- `unpen.index`: vector with indices of mandatory covariates.

With this instructions the user can create new functions for screening main effects. To set further arguments for this function create a list of arguments, which can be quoted in `args.screen.main`. In the next step the screened main effects will be used for orthogonalizing the data by computing residuals corresponding to the selected main effects and the mandatory covariates.

**Value**

The adaptive function returns the following values:

- `model`: Generalized linear model. *fit.uniglm* returns an object of class `glm` and *fit.GAMBoost* returns an object of class `GAMBoost`.
- `xnames`: vector of length p containing the names of the covariates.
- `indmain`: vector of length p containing the indices of the selected covariates.
- `beta`: vector of length p containing the coefficients of the selected covariates.

**Author(s)**

Written by Isabell Hoffmann <isabell.hoffmann@uni-mainz.de>.

**See Also**

*p.adjust, glm, GAMBoost*

---

**simul.int**  
*Function for simulating survival data with interactions*

**Description**

`simul.int` simulates survival data with exponentially distributed survival times where interactions are included. The interactions are generated by variables without effect.
simul.int

Usage

```r
simul.int(seed, n = 100, p = 1000, 
        n.main = 2, 
        n.int = 2, 
        beta.main=2, 
        beta.int = 4, 
        censparam = 1/5, 
        lambda = 1/20)
```

Arguments

- **seed**: seed for random number generator.
- **n**: number of individuals in the data set.
- **p**: number of covariates in the data set.
- **n.main**: number of main effects with effects.
- **n.int**: number of interactions with effects.
- **beta.main**: effect size of main effects.
- **beta.int**: effect size of interaction effects.
- **censparam**: value for censoring Parameter.
- **lambda**: value for baseline hazard.

Details

The function `simul.int` creates exponentially distributed survival times with baseline hazard $\lambda$. The number of covariates is $p$ and the sample size is $n$. All covariates are standard normal distributed. The first $n.main$ columns correspond to the main effects and the following $n.int$ columns correspond to the interactions. The effect sizes of the main effects are in absolute value $\beta.main$, whereupon the first $\text{floor}(n.main/2)$ variables have positive effect sizes and the rest of the main effects have effect size $-\beta.main$. The effect sizes of the interactions are in absolute value $\beta.int$, where half of them are positive and half of them are negative like for the main effects.

Value

`simul.int` returns the simulated data set and the information about the effect sizes:

- **data**: simulated dataset with $p+2$ columns and $n$ rows. The last two columns consist of exponentially distributed survival time (`obs.time`) and status (`obs.status`).
- **info**: information about the effect sizes of the main effects and of the included interactions.

Author(s)

Written by Isabell Hoffmann <isabell.hoffmann@uni-mainz.de>.
Examples

```r
# Create survival data with interactions:
simul <- simul.int(287578, n = 200, p = 1000,
  beta.int = 1.0,
  beta.main = 0.9,
  censparam = 1/20,
  lambda = 1/20)

# Show the effect sizes of the main effects and interactions of the simulated data set:
simul$info

# Extract the data set:
data <- simul$data

# Plot the Kaplan Meier:
simul.fit <- survfit(Surv(obs.time,obs.status) ~ 1, data = data)
plot(simul.fit)
```

---

**sprinter**  
*Main function for building prognostic models by considering interactions.*

**Description**

The function `sprinter` builds a prognostic model by preselecting interactions and main effects before fitting a regression model.

**Usage**

```r
sprinter(x,
  time,
  status = rep(0, nrow(x)),
  mandatory = NULL,
  repetitions = 25,
  n.inter.candidates = 1000,
  screen.main = NULL,
  screen.inter = fit.rf,
  fit.final = screen.main,
  args.screen.main = list(),
  args.screen.inter = list(),
  args.fit.final = args.screen.main,
  orthogonalize = TRUE, cutoff = 0,
  parallel = FALSE, mc.cores = detectCores(),
  ...)
```
**Arguments**

- **x**: n * p matrix of covariates.
- **time**: vector of length n specifying the observed times.
- **status**: censoring indicator, i.e., vector of length n with entries 0 for censored observations and 1 for uncensored observations. This optional argument is necessary in time-to-event data.
- **mandatory**: vector with variable names of mandatory covariates, where parameter estimation should be performed unpenalized.
- **repetitions**: number of repetitions of the interaction screening approach. Repetitions are performed by creating subsamples and applying the interaction screening approach on each subsample dataset separately.
- **n.inter.candidates**: minimal number of potential interaction candidates, which are considered in the final model building step.
- **screen.main**: function for screening potential main effects. `fit.uniCox` performs univariate Cox-regressions and selects the main effects by their adjusted p-values. `fit.CoxBoost` performs a variable selection by fitting a Cox model by likelihood based boosting. Other methods are possible to be implemented as adaptive functions for the usage in `sprinter`. For more details see `fit.uniCox`.
- **screen.inter**: function for detecting potential interaction candidates. `fit.rf` performs a random forest and `fit.logicReg` performs a logic regression. Other methods are possible to implement as adaptive functions for the use in `sprinter`. For more details see `fit.rf`.
- **fit.final**: function for building the final Cox proportional hazards model. Default is the function set in `screen.main`.
- **args.screen.main**: list of arguments which should be used in the main effects detection step.
- **args.screen.inter**: list of arguments which should be used in the interaction screening step.
- **args.fit.final**: list of arguments which should be used in the final model building step.
- **orthogonalize**: logical value. If true all variables are made orthogonal to those that are assessed as main effects by `screen.main`.
- **cutoff**: value or function to evaluate a value according to the variable importance. The cutoff is used to select variables for evaluating the pairwise inclusion frequencies.
- **parallel**: logical value indicating whether the interaction screening step should be performed parallel.
- **mc.cores**: the numbers of cores to use, if parallel = TRUE.
- **...**: additional arguments.

**Details**

A call to the `sprinter`-function fits a prognostic model to time-to-event data by combining available statistical components. The modular structure secures the simultaneously consideration of two
important elements in a model: interactions and main effects. Interactions play an important role in molecular applications, because some effects arise only if two genes are differentially expressed at the same time. Therefore, it is important to consider interaction terms when predicting a clinical outcome. The method which is used to preselect interactions is set in `screen.inter`.

The interactions are preselected with respect to potential main effects. Therefore, a main effects model is performed for extracting the existing main effects and to be able to compare the main effects model with the final model to show the benefit of considering interactions. The method which is used to perform a main effects model is set in `screen.main`.

The final model is performed by the main effects resulting from `screen.main` and the interactions resulting from `screen.inter` as covariates. The method which is used for building the final model is set in `fit.final`. As default the same method is used as in `screen.main`. In the following the three components of the framework are explained more in detail: (1) Fitting a main effects model, (2) adjusting the data and pre-selecting interaction terms and (3) building the comprehensive model including promising interactions.

For screening main effects, `sprinter` allows any approach which can handle with high-dimensional datasets with time-to-event settings. The following two established approaches have already been prepared for usage: (A) Univariate-Cox regression with adjusted p-values (`fit.uniCox`) and (B) CoxBoost (`fit.CoxBoost`). Other approaches can also be implemented (see `fit.CoxBoost`).

For screening the interaction effects, `sprinter` offers the random forest method `fit.rf` and logic regression `fit.logicReg` for pre-selecting interactions. For each variable a variable importance measurement is calculated that considers the underlying interaction structure and reflects the meaning of a variable for the forest or the logic regression, respectively. The variable importance is used to construct the relevant interactions for the model. Before pre-selecting the interactions, the data are modified so that weaker interaction effects that are originally overlaid by stronger main effects can be detected. To achieve this, the data are orthogonalized by computing residuals corresponding to the selected main effects and the mandatory covariates.

For better stabilization subsamples are created and the interaction detection approach is performed on each subsampled dataset. As this step can be computationally expensive it is possible to parallelize this step, by `parallel = TRUE`. To summarize the results of all subsamples, pairwise variable inclusion frequencies of the constructed interactions terms are computed and the `n.inter.candidates` most frequent pairs are selected as relevant interaction terms. Other approaches can also be implemented (see `fit.rf`).

For building the final model, the user can set the desired method in `fit.final`. If no method is required the same method is used as for building the main effects model. In contrast to building the main effects model, the final model is constructed by the variables selected in the main effects model together with the `n.inter.candidates` pre-selected interactions of the screening step.

### Value

An object of class (sprinter) with the following components:

- `n.inter.candidates`  
  Number of potential interaction candidates considered in the final model building.

- `inter.candidates`  
  Vector of length `n.inter.candidates` with the potential interaction candidates considered in the final model building.
Main effects model. The class depends on the function used in `screen.main`.

Final model. The class depends on the function used in `fit.final`.

Vector of the variable names used in the final model.

Author(s)

Written by Isabell Hoffmann <isabell.hoffmann@uni-mainz.de>.

References


See Also

`rfsrc`, `coxph`, `CoxBoost`

Examples

```r
## Not run:
##---------------------
## Survival analysis
##---------------------

# Fit a Cox proportional hazards model by CoxBoost by considering interactions after screening interactions by random forest
# system.time:
# user system elapsed
# 370.97 2.32 374.31
# For a faster run set repetitions down!

# Create survival data with interactions:
simulation <- simul.int(287578, n = 200, p = 500,
  beta.int = 1.0,
  beta.main = 0.9,
  censparam = 1/20,
  lambda = 1/20)

data <- simulation$data

# Showing True Effects:
simulation$info

# Perform the sprinter approach:
set.seed(123)
testcb <- sprinter(x = data[,1:500],
  time = data$obs.time,
  status = data$obs.status,
  repetitions = 10,
)```
```r
mandatory = c("ID1","ID2"),
n.inter.candidates = 1000,
screen.main = fit.CoxBoost,
fit.final = fit.CoxBoost,
args.screen.main = list(seed=123,stepno = 10, K = 10,
                      criterion = 'pscore', nu = 0.05),
parallel = FALSE)
summary(testcb)

# Fit a Cox proportional hazards model by considering
# interactions after screening interactions by random forest
# and selecting relevant effects by univariate Cox regression:
# system.time:
#  user  system elapsed
# 374.50  1.53  376.68
# For a faster run set repetitions down!

# Create survival data with interactions:
data <- simul.int(287578,n = 200, p = 500,
                 beta.int = 1.0,
                 beta.main = 0.9,
                 censparam = 1/20,
                 lambda = 1/20)[[1]]

# Perform the sprinter approach:
set.seed(123)
testunicox <- sprinter(x=data[1:500],
                        time = data$obs.time,
                        status = data$obs.status,
                        repetitions = 10,
                        mandatory = c("ID1","ID2"),
                        n.inter.candidates = 1000,
                        screen.main = fit.uniCox,
                        fit.final = fit.uniCox,
                        parallel = FALSE)
summary(testunicox)

# true coefficients:
# ID1  ID2  ID5:ID6  ID7:ID8
# 0.9 -0.9 1 -1

#-----------------------------
# Continuous outcome
#-----------------------------
```
# selection of main effects by univariate generalized linear models and pre-selections of interactions
# by random forest:
sprinter.glm.rf.con <- sprinter(x=data[,1:500],
  time = data$obs.time,
  repetitions = 10,
  mandatory = c("ID1","ID2"),
  n.inter.candidates = 1000,
  screen.main = fit.uniGlm,
  fit.final = fit.uniGlm,
  parallel = FALSE)

# selection of main effects by univariate generalized linear models and pre-selections of interactions
# by logic regression:
sprinter.glm.logicR.con <- sprinter(x=data[,1:500],
  time = data$obs.time,
  repetitions = 10,
  mandatory = c("ID1","ID2"),
  n.inter.candidates = 1000,
  screen.main = fit.uniGlm,
  screen.inter = fit.logicReg,
  fit.final = fit.uniGlm,
  args.screen.inter = list(type = 2),
  parallel = FALSE)

# selection of main effects by GAMBoost and pre-selections of interactions
# by random forest:
sprinter.gamboost.rf.con <- sprinter(x=data[,1:500],
  time = data$obs.time,
  repetitions = 10,
  mandatory = c("ID1","ID2"),
  n.inter.candidates = 1000,
  screen.main = fit.GAMBoost,
  args.screen.main = list(stepno = 10),
  fit.final = fit.GAMBoost,
  parallel = FALSE)

### Binary outcome

x <- matrix(runif(200*500,min=-1,max=1),200,500)
colnames(x) <- paste('ID', 1:500. sep = '')
eta <- -0.5 + 2*x[,1] - 2*x[,3] + 2 * x[,3]*x[,4]
y <- rbinom(200,1,binomial(linkinv(eta))

# selection of main effects by univariate generalized linear models and pre-selections of interactions
# by random forest:
Generates a summary for objects of class `resample.sprinter`

**Description**

Shows the interactions with the largest variable inclusion frequencies together with its mean coefficients and plots the mean coefficient of each variable against its single coefficients.
**Usage**

```r
## S3 method for class 'resample.sprinter'
summary(object, print = TRUE, plot = TRUE, optional.only = FALSE, threshold.vif = 0, ...)
```

**Arguments**

- **object**: Object of class `resample.sprinter`
- **print**: logical value. If true, a list of interactions with the largest inclusion frequencies together with its mean coefficients is shown.
- **plot**: logical value. If true, the mean coefficients for each selected covariate are plotted against its single coefficients.
- **optional.only**: logical value. If true, only the coefficients of the optional variables are plotted. If false, the coefficients of the mandatory variables are plotted as well.
- **threshold.vif**: only the coefficients with inclusion frequencies larger than `threshold.vif` are plotted.
- **...**: additional arguments.

**Author(s)**

Isabell Hoffmann <isabell.hoffmann@uni-mainz.de>

**See Also**

- `resample.sprinter`
- `summary.sprinter`  

---

**summary.sprinter**  
*Generates a summary for objects of class sprinter*

---

**Description**

Shows the main candidates with coefficients of the main effects model together with the interaction candidates with largest inclusion frequencies and prints the final model.

**Usage**

```r
## S3 method for class 'sprinter'
summary(object, ...)
```

**Arguments**

- **object**: Object of class `sprinter`
- **...**: additional arguments.

**Author(s)**

Isabell Hoffmann <isabell.hoffmann@uni-mainz.de>
Index

*Topic \textasciitilde kwd1
  simul.int, 14
*Topic \textasciitilde kwd2
  simul.int, 14

CoxBoost, 11, 12, 19
coxph, 11, 12, 19

fit.CoxBoost, 18
fit.CoxBoost(screen.main.cox), 9
fit.GAMBoost(screen.main.glm), 12
fit.logicReg(screen.inter), 7
fit.rf, 17, 18
fit.rf(screen.inter), 7
fit.uniCox, 17
fit.uniCox(screen.main.cox), 9
fit.uniGlm(screen.main.glm), 12

GAMBoost, 14
glm, 14

logreg, 9

p.adjust, 12, 14
plot.resample.sprinter, 2
predict.sprinter, 2

resample.sprinter, 2, 4, 23
rfsrc, 8, 9, 19

screen.inter, 7
screen.main.cox, 9
screen.main.glm, 12
simul.int, 14
sprinter, 2, 3, 5, 8, 11, 13, 16, 23
summary.resample.sprinter, 22
summary.sprinter, 23