Package ‘riskRegression’

September 7, 2023

Type Package

Title Risk Regression Models and Prediction Scores for Survival Analysis with Competing Risks

Version 2023.09.08

Depends R (>= 3.5.0),
Imports cmprsk, data.table (>= 1.12.2), doParallel, foreach, ggplot2 (>= 3.1.0), graphics, lattice, lava (>= 1.6.5), mets, mvtnorm, parallel, plotrix, prodlim (>= 2018.4.18), Publish, ranger, Rcpp, rms (>= 5.1.3), stats, survival (>= 2.44.1), timereg (>= 1.9.3)

LinkingTo Rcpp, RcppArmadillo

Suggests boot, smcfc, casebase, glmnet, gbm, flexsurv, grpreg, hal9001, mgcv, mstate, nnls, numDeriv, party, pec, penalized, pROC, randomForest, randomForestSRC, rpart, scam, SuperLearner, testthat, R.rsp

Maintainer Thomas Alexander Gerds <tag@biostat.ku.dk>

Description Implementation of the following methods for event history analysis.
   Risk regression models for survival endpoints also in the presence of competing
   risks are fitted using binomial regression based on a time sequence of binary
   event status variables. A formula interface for the Fine-Gray regression model
   and an interface for the combination of cause-specific Cox regression models.
   A toolbox for assessing and comparing performance of risk predictions (risk
   markers and risk prediction models). Prediction performance is measured by the
   Brier score and the area under the ROC curve for binary possibly time-dependent
   outcome. Inverse probability of censoring weighting and pseudo values are used
   to deal with right censored data. Lists of risk markers and lists of risk models
   are assessed simultaneously. Cross-validation repeatedly splits the data, trains
   the risk prediction models on one part of each split and then summarizes and
   compares the performance across splits.

License GPL (>= 2)

RoxygenNote 7.2.3

VignetteBuilder R.rsp

Encoding UTF-8
URL: https://github.com/tagteam/riskRegression

Bug Reports: https://github.com/tagteam/riskRegression/issues

NeedsCompilation: yes

Author: Thomas Alexander Gerds [aut, cre],
        Johan Sebastian Ohlendorff [aut],
        Paul Blanche [ctb],
        Rikke Mortensen [ctb],
        Marvin Wright [ctb],
        Nikolaj Tollenaar [ctb],
        John Muschelli [ctb],
        Ulla Brasch Mogensen [ctb],
        Brice Ozenne [aut] (<https://orcid.org/0000-0001-9694-2956>)

Repository: CRAN

Date/Publication: 2023-09-07 09:20:02 UTC

R topics documented:

anova.ate ................................................................. 4
as.data.table.ate .................................................. 7
as.data.table.influenceTest ......................................... 8
as.data.table.predictCox ............................................ 8
as.data.table.predictCSC ............................................ 9
ate ................................................................. 9
autoplot.ate ..................................................... 15
autoplot.predictCox ................................................ 18
autoplot.predictCSC ............................................... 21
autoplot.Score ...................................................... 23
baseHaz_cpp ....................................................... 24
boot2pvalue .......................................................... 25
boxplot.Score ....................................................... 26
calcSeCox ........................................................... 29
calcSeCSC ........................................................... 31
Cforest ............................................................. 32
coeff.CauseSpecificCox ............................................. 33
coeff.riskRegression ............................................... 34
colCenter_cpp ....................................................... 34
colCumSum .......................................................... 35
colMultiply_cpp ..................................................... 35
colScale_cpp ........................................................ 36
confint.ate ....................................................... 37
confint.influenceTest .............................................. 38
confint.predictCox ............................................... 40
confint.predictCSC ............................................... 41
coxBaseEstimator .................................................. 42
coxCenter .......................................................... 43
coxFormula .......................................................... 45
R topics documented:

- coxLP
- coxModelFrame
- coxN
- coxSpecial
- coxStrata
- coxStrataLevel
- coxVarCov
- coxVariableName
- CSC
- Ctree
- discreteRoot
- FGR
- getSplitMethod
- GLMnet
- Hal9001
- iid.wglm
- iidCox
- influenceTest
- information.wglm
- IPA
- ipcw
- Melanoma
- model.matrix.cph
- model.matrix.phreg
- Paquid
- penalizedS3
- plot.riskRegression
- plotAUC
- plotBrier
- plotCalibration
- plotEffects
- plotPredictRisk
- plotRisk
- plotROC
- predict.CauseSpecificCox
- predict.FGR
- predict.riskRegression
- predictCox
- predictCoxPL
- predictRisk
- print.ate
- print.CauseSpecificCox
- print.FGR
- print.influenceTest
- print.IPA
- print.predictCox
- print.predictCSC
- print.riskRegression
Risk Comparison Over Time

Description

Comparison of risk differences or risk ratios over all timepoints.
Usage

```r
## S3 method for class 'ate'
anova(
  object,
  allContrast = NULL,
  type = "diff",
  estimator = object$estimator[1],
  test = "CvM",
  transform = NULL,
  alternative = "two.sided",
  n.sim = 10000,
  print = TRUE,
  ...
)
```

Arguments

- `object`: A `ate` object, i.e. output of the `ate` function.
- `allContrast`: [matrix] contrast for which the risks should be compared. Matrix with two rows, the first being the sequence of reference treatments and the second the sequence of alternative treatments.
- `type`: [character vector] the functionnal used to compare the risks: "diffRisk" or "ratioRisk".
- `estimator`: [character] The type of estimator relative to which the comparison should be performed.
- `test`: [character] The type of statistic used to compare the risks over times: "KM" (extremum risk), "CvM" (sum of squares of the risk), or "sum" (sum of the risks).
- `transform`: [character] Should a transformation be used, e.g. the test is performed after log-transformation of the estimate, standard error, and influence function.
- `alternative`: [character] a character string specifying the alternative hypothesis, must be one of "two.sided", "greater" or "less".
- `n.sim`: [integer, >0] the number of simulations used to compute the p-values.
- `print`: [logical] should the results be displayed?
- `...`: Not used.

Details

Experimental!!!
set.seed(12)
n <- 200
dtS <- sampleData(n,outcome="survival")
dtS$X12 <- LETTERS[as.numeric(as.factor(paste0(dtS$X1,dtS$X2)))]
dtS <- dtS[dtS$X12!='D']

## model fit
fit <- cph(formula = Surv(time,event)~ X1+X6,data=dtS,y=TRUE,x=TRUE)
seqTime <- 1:10
ateFit <- ate(fit, data = dtS, treatment = "X1", contrasts = NULL,
times = seqTime, B = 0, iid = TRUE, se = TRUE, verbose = TRUE, band = TRUE)

## display
autoplot(ateFit)

## inference (two sided)
statistic <- ateFit$diffRisk$estimate/ateFit$diffRisk$se
confint(ateFit, p.value = TRUE, method.band = "bonferroni")$diffRisk
confint(ateFit, p.value = TRUE, method.band = "maxT-simulation")$diffRisk
anova(ateFit, test = "KS")
anova(ateFit, test = "CvM")
anova(ateFit, test = "sum")

## manual calculation (one sided)
n.sim <- 1e4
statistic <- ateFit$diffRisk[, estimate/se]
IID.norm <- scale(ateFit$iid$GFORMULA["1"]-ateFit$iid$GFORMULA["0"],
                  scale = ateFit$diffRisk$se)

ls.out <- lapply(1:n.sim, function(iSim){
iG <- rnorm(NROW(IID.norm))
iCurve <- t(IID.norm) %*% iG
data.table(max = max(iCurve), L2 = sum(iCurve^2), sum = sum(iCurve),
           maxC = max(iCurve) - max(statistic),
           L2C = sum(iCurve^2) - sum(statistic^2),
           sumC = sum(iCurve) - sum(statistic),
           sim = iSim)
})
dt.out <- do.call(rbind,ls.out)
dt.out[,.(max = mean(.SD$maxC>=0),
        L2 = mean(.SD$L2C>=0),
        sum = mean(.SD=sumC>=0))]

## permutation
n.sim <- 250
stats.perm <- vector(mode = "list", length = n.sim)
progressBar <- txtProgressBar(max = n.sim, style=3)
treatVar <- ateFit$variables["treatment"]

for(iSim in 1:n.sim){
  iData <- data.table::copy(dtS)
  stats.perm[[iSim]] <- 

iIndex <- sample.int(NROW(iData), replace = FALSE)
iData[, c(treatVar) := .SD[[treatVar]][iIndex]]

iFit <- update(fit, data = iData)
iAteSim <- ate(iFit, data = iData, treatment = treatVar,
times = seqTime, verbose = FALSE)
iStatistic <- iAteSim$diffRisk[,.(max = max(iStatistic),
                         L2 = sum(iStatistic^2),
                         sum = sum(iStatistic))]

stats.perm[[iSim]] <- cbind(iAteSim$diffRisk[,.(max = max(iStatistic),
                           L2 = sum(iStatistic^2),
                           sum = sum(iStatistic))],
                           sim = iSim)
stats.perm[[iSim]]$maxC <- stats.perm[[iSim]]$max - max(statistic)
stats.perm[[iSim]]$L2C <- stats.perm[[iSim]]$L2 - sum(statistic^2)
stats.perm[[iSim]]$sumC <- stats.perm[[iSim]]$sum - sum(statistic)

setTxtProgressBar(pb, iSim)
}
dtstats.perm <- do.call(rbind,stats.perm)
dtstats.perm[,.(max = mean(.SD$maxC>=0),
                       L2 = mean(.SD$L2C>=0),
                       sum = mean(.SD$sumC>=0))]

## End(Not run)

---

### as.data.table.ate

**Turn ate Object Into a data.table**

**Description**

Turn ate object into a data.table.

**Usage**

```r
## S3 method for class 'ate'
as.data.table(
  x,
  keep.rownames = FALSE,
  estimator = x$estimator,
  type = c("meanRisk", "diffRisk", "ratioRisk"),
  ...
)
```

**Arguments**

- `x`: object obtained with function `ate`
- `keep.rownames`: Not used.
- `estimator`: [character] The type of estimator relative to which the estimates should be output.
type [character vector] The type of risk to export. Can be "meanRisk" to export the risks specific to each treatment group, "diffRisk" to export the difference in risks between treatment groups, or "ratioRisk" to export the ratio of risks between treatment groups.
...
Not used.
as.data.table.predictCSC

Turn predictCSC Object Into a data.table

Description

Turn predictCSC object into a data.table.

Usage

## S3 method for class 'predictCSC'
as.data.table(x, keep.rownames = FALSE, se = TRUE, ...)

Arguments

x          object obtained with function predictCSC
keep.rownames  not used
se          should standard errors/quantile for confidence bands be displayed?
...        not used

ate

Average Treatment Effects Computation

Description

Use the g-formula or the IPW or the double robust estimator to estimate the average treatment effect (absolute risk difference or ratio) based on Cox regression with or without competing risks.

Usage

ate(
    event,
    treatment,
    censor = NULL,
    data,
    data.index = NULL,
    formula = NULL,
    estimator = NULL,
    strata = NULL,
    contrasts = NULL,
    allContrasts = NULL,
    times,
    cause = NA,
    landmark = NULL,
se = TRUE,
iid = (B == 0) && (se || band),
known.nuisance = FALSE,
band = FALSE,
B = 0,
seed,
handler = "foreach",
mc.cores = 1,
cl = NULL,
verbose = TRUE,
...
)

Arguments

event Outcome model which describes how the probability of experiencing a terminal event depends on treatment and covariates. The object carry its own call and have a predictRisk method. See examples.

treatment Treatment model which describes how the probability of being allocated to a treatment group depends on covariates. The object must be a glm object (logistic regression) or the name of the treatment variable. See examples.

censor Censoring model which describes how the probability of being censored depends on treatment and covariates. The object must be a coxph or cph object. See examples.

data [data.frame or data.table] Data set in which to evaluate risk predictions based on the outcome model

data.index [numeric vector] Position of the observation in argument data relative to the dataset used to obtain the argument event, treatment, censor. Only necessary for the standard errors when computing the Average Treatment Effects on a subset of the data set.

formula For analyses with time-dependent covariates, the response formula. See examples.

estimator [character] The type of estimator used to compute the average treatment effect. Can be "G-formula", "IPTW", or "AIPTW". When using estimator="G-formula", a model for the outcome should be provided (argument event). When using estimator="IPTW", a model for the treatment should be provided (argument treatment), as well as for the censoring (if any, argument censor). When using estimator="AIPTW" (double robust estimator), a model for the outcome and the treatment should be provided (argument event and treatment), as well as for the censoring (if any, argument censor).

strata [character] Strata variable on which to compute the average risk. Incompatible with treatment. Experimental.

contrasts [character vector] levels of the treatment variable for which the risks should be assessed and compared. Default is to consider all levels.

allContrasts [2-row character matrix] levels of the treatment variable to be compared. Default is to consider all pairwise comparisons.
times
cause
landmark
se
iid
known.nuisance
band
B
seed
handler
mc.cores
cl
verbose

... passed to predictRisk

Author(s)

Brice Ozenne <broz@sund.ku.dk> and Thomas Alexander Gerds <tag@biostat.ku.dk>

References

See Also

as.data.table to extract the estimates in a data.table object. autoplot.ate for a graphical representation the standardized risks. confint.ate to compute (pointwise/simultaneous) confidence intervals and (unadjusted/adjusted) p-values, possibly using a transformation. summary.ate for a table containing the standardized risks over time and treatment/strata.

Examples

library(survival)
library(rms)
library(prodlim)
library(data.table)
set.seed(10)

### Survival settings ###
### ATE with Cox model ###

## generate data
n <- 100
dtS <- sampleData(n, outcome="survival")
dtS$time <- round(dtS$time,1)
dtS$X1 <- factor(rbinom(n, prob = c(0.3,0.4) , size = 2), labels = paste0("T",0:2))

## estimate the Cox model
fit <- cph(formula = Surv(time,event)~ X1+X2,data=dtS,y=TRUE,x=TRUE)

## compute the ATE at times 5, 6, 7, and 8 using X1 as the treatment variable
## standard error computed using the influence function
## confidence intervals / p-values based on asymptotic results
ateFit1a <- ate(fit, data = dtS, treatment = "X1", times = 5:8)
summary(ateFit1a)
summary(ateFit1a, short = TRUE, type = "meanRisk")
summary(ateFit1a, short = TRUE, type = "diffRisk")
summary(ateFit1a, short = TRUE, type = "ratioRisk")

## Not run:
## same as before with in addition the confidence bands / adjusted p-values
## (argument band = TRUE)
ateFit1b <- ate(fit, data = dtS, treatment = "X1", times = 5:8,
               band = TRUE)
summary(ateFit1b)

## by default bands/adjuste p-values computed separately for each treatment modality
summary(ateFit1b, band = 1,
       se = FALSE, type = "diffRisk", short = TRUE, quantile = TRUE)
## adjustment over treatment and time using the band argument of confint
summary(ateFit1b, band = 2,
       se = FALSE, type = "diffRisk", short = TRUE, quantile = TRUE)

## confidence intervals / p-values computed using 1000 bootstrap samples
## (argument se = TRUE and B = 1000)
ateFit1c <- ate(fit, data = dtS, treatment = "X1",
               se = TRUE, B = 1000)
ateFit1d <- ate(fit, data = dtS, treatment = "X1",
    times = 5:8, se = TRUE, B = 50, handler = "mclapply")
## NOTE: for real applications 50 bootstrap samples is not enough

## same but using 2 cpus for generating and analyzing the bootstrap samples
## (parallel computation, argument mc.cores = 2)
ateFit1d <- ate(fit, data = dtS, treatment = "X1",
    times = 5:8, se = TRUE, B = 50, mc.cores = 2)

## manually defining the cluster to be used
## useful when specific packages need to be loaded in each cluster
fit <- cph(formula = Surv(time,event)~ X1+X2+rcs(X6),data=dtS,y=TRUE,x=TRUE)
cl <- parallel::makeCluster(2)
parallel::clusterEvalQ(cl, library(rms))
ateFit1e <- ate(fit, data = dtS, treatment = "X1",
    times = 5:8, se = TRUE, B = 50,
    handler = "foreach", cl = cl)
## End(Not run)

#### Survival settings without censoring ####
#### ATE with glm ####
## generate data
n <- 100
dtB <- sampleData(n, outcome="binary")
dtB[, X2 := as.numeric(X2)]

## estimate a logistic regression model
fit <- glm(formula = Y ~ X1+X2, data=dtB, family = "binomial")

## compute the ATE using X1 as the treatment variable
## only point estimate (argument se = FALSE)
ateFit1a <- ate(fit, data = dtB, treatment = "X1", se = FALSE)
ateFit1a

## Not run:
## with confidence intervals
ateFit1b <- ate(fit, data = dtB, treatment = "X1",
    times = 5) ## just for having a nice output not used in computations
summary(ateFit1b, short = TRUE)

## using the lava package
library(lava)
ateLava <- estimate(fit, function(p, data){
a <- p["(Intercept)"] ; b <- p["X11"] ; c <- p["X2"] ;
R.X11 <- expit(a + b + c * data["X2"])
R.X10 <- expit(a + c * data["X2"])
list(risk0=R.X10,risk1=R.X11,riskdiff=R.X11-R.X10),
    average=TRUE)
ateLava
## End(Not run)

## see wglm for handling right-censoring with glm

#### Competing risks settings ####

#### ATE with cause specific Cox regression ####

## generate data

```r
n <- 500
set.seed(10)
dt <- sampleData(n, outcome="competing.risks")
dt$X1 <- factor(rbinom(n, prob = c(0.2,0.3) , size = 2), labels = paste0("T",0:2))
```

## estimate cause specific Cox model

```r
fitCR <- CSC(Hist(time,event)~ X1+X8,data=dt,cause=1)
```

## compute the ATE at times 1, 5, 10 using X1 as the treatment variable

```r
ateFit2a <- ate(fitCR, data = dt, treatment = "X1", times = c(1,5,10),
                cause = 1, se = TRUE, band = TRUE)
```

```r
summary(ateFit2a)
as.data.table(ateFit2a)
```

#### Double robust estimator ####

Not run:

## generate data

```r
n <- 500
set.seed(10)
dt <- sampleData(n, outcome="competing.risks")
dt$time <- round(dt$time,1)
dt$X1 <- factor(rbinom(n, prob = c(0.4) , size = 1), labels = paste0("T",0:1))
```

## working models

```r
m.event <- CSC(Hist(time,event)~ X1+X2+X3+X5+X8,data=dt)
m.censor <- coxph(Surv(time,event==0)~ X1+X2+X3+X5+X8,data=dt, x = TRUE, y = TRUE)
m.treatment <- glm(X1~X2+X3+X5+X8,data=dt,family=binomial(link="logit"))
```

## prediction + average

```r
ateRobust <- ate(event = m.event,
                 treatment = m.treatment,
                 censor = m.censor,
                 data = dt, times = 5:10,
                 cause = 1, band = TRUE)
```

## compare various estimators

```r
ateRobust3 <- ate(event = m.event,
                 treatment = m.treatment,
                 censor = m.censor,
                 estimator = c("GFORMULA","IPTW","AIPTW"),
                 data = dt, times = c(5:10),
                 cause = 1, se = TRUE)
```

print(setkeyv(as.data.table(ateRobust3, type = "meanRisk"),"time"))
print(setkeyv(as.data.table(ateRobust3, type = "diffRisk"),"time"))
## End(Not run)

#### time-dependent covariates ####

Not run:
```r
library(survival)
fit <- coxph(Surv(time, status) ~ celltype+karno + age + trt, veteran)
vet2 <- survSplit(Surv(time, status) ~ ., veteran,
                 cut=c(60, 120), episode="timegroup")
fitTD <- coxph(Surv(tstart, time, status) ~ celltype+karno + age + trt,
               data= vet2,x=1)
set.seed(16)
resVet <- ate(fitTD,formula=Hist(entry=tstart,time=time,event=status)-1,
data = vet2, treatment = "celltype",
times=5,verbose=1,
landmark = c(0,30,60,90), cause = 1, B = 50, se = 1,
band = FALSE, mc.cores=1)
summary(resVet)
```

Not run:
```r
d=sampleDataTD(127)
library(survival)
d[,status:=1*(event==1)]
d[,X3:=as.factor(X3)]
## ignore competing risks
cox1TD <- coxph(Surv(start,time, status,type="counting") ~ X3+X5+X6+X8,
data=d, x = TRUE)
restD1 <- ate(cox1TD,formula=Hist(entry=start,time=time,event=status)-1,
data = d, treatment = "X3", contrasts = NULL,
times=.5,verbose=1,
landmark = c(0,0.5,1), B = 20, se = 1,
band = FALSE, mc.cores=1)
restD1
## account for competing risks
cscTD <- CSC(Hist(time=time, event=event,entry=start) ~ X3+X5+X6+X8, data=d)
set.seed(16)
restD <- ate(cscTD,formula=Hist(entry=start,time=time,event=event)-1,
data = d, treatment = "X3", contrasts = NULL,
times=.5,verbose=1,
landmark = c(0,0.5,1), cause = 1, B = 20, se = 1,
band = FALSE, mc.cores=1)
restD
```

## End(Not run)
Description

Plot average risks.

Usage

```r
## S3 method for class 'ate'
autoplot(
  object,
  type = "meanRisk",
  first.derivative = FALSE,
  estimator = object$estimator[1],
  ci = object$inference$ci,
  band = object$inference$band,
  plot.type = "1",
  plot = TRUE,
  smooth = FALSE,
  digits = 2,
  alpha = NA,
  ylab = NULL,
  ...
)
```

Arguments

- **object**: Object obtained with the function `ate`.
- **type**: [character vector] what to displayed. Can be "meanRisk" to display the risks specific to each treatment group, "diffRisk" to display the difference in risks between treatment groups, or "ratioRisk" to display the ratio of risks between treatment groups.
- **first.derivative**: [logical] If TRUE, display the first derivative over time of the risks/risk differences/risk ratios. (confidence intervals are obtained via simulation).
- **estimator**: [character] The type of estimator relative to which the risks should be displayed.
- **ci**: [logical] If TRUE display the confidence intervals for the average risks.
- **band**: [logical] If TRUE display the confidence bands for the average risks.
- **plot.type**: [character] Type of plot to be used. plot.type="2" is useful when looking simulateneous at all eventtimes. Otherwise use plot.type="1".
- **plot**: [logical] Should the graphic be plotted.
- **smooth**: [logical] Should a smooth version of the risk function be plotted instead of a simple function?
- **digits**: [integer, >0] Number of decimal places.
- **alpha**: [numeric, 0-1] Transparency of the confidence bands. Argument passed to ggplot2::geom_ribbon.
- **ylab**: [character] Label for the y axis.
- **...**: Additional parameters to customize the display.
Value

Invisible. A list containing:

- plot: the ggplot object.
- data: the data used to create the plot.

See Also

ate to compute average risks.

Examples

```r
library(survival)
library(rms)
library(ggplot2)

#### simulate data ####
n <- 1e2
set.seed(10)
dtS <- sampleData(n, outcome="survival")
seqTimes <- c(0, sort(dtS$time[dtS$event==1]), max(dtS$time))

#### Cox model ####
fit <- cph(formula = Surv(time,event)~ X1+X2,data=dtS,y=TRUE,x=TRUE)

#### plot.type = 1: for few timepoints ####
ateFit <- ate(fit, data = dtS, treatment = "X1",
times = c(1,2,5,10), se = TRUE, band = TRUE)
ggplot2::autoplot(ateFit)

#### plot.type = 2: when looking at all jump times ####
ateFit <- ate(fit, data = dtS, treatment = "X1",
times = seqTimes, se = TRUE, band = TRUE)
ggplot2::autoplot(ateFit, plot.type = "2")

#### customize plot
outGG <- ggplot2::autoplot(ateFit, plot.type = "2", alpha = 0.25)
outGG$plot + facet_wrap(~X1, labeller = label_both)

#### Looking at the difference after smoothing
outGGS <- ggplot2::autoplot(ateFit, plot.type = "2", alpha = NA, smooth = TRUE)
outGGS$plot + facet_wrap(~X1, labeller = label_both)
```
## first derivative
## (computation of the confidence intervals takes time)
## (based on simulation - n.sim parameter)
ggplot2::autoplot(ateFit, plot.type = "2", smooth = TRUE, band = FALSE, type = "diffRisk")
ggplot2::autoplot(ateFit, plot.type = "2", smooth = TRUE, first.derivative = TRUE, band = FALSE, type = "diffRisk")

## End(Not run)

autoplot.predictCox  

Plot Predictions From a Cox Model

Description
Plot predictions from a Cox model.

Usage
## S3 method for class 'predictCox'
autoplot(
  object,
  type = NULL,
  ci = object$se,
  band = object$band,
  plot = TRUE,
  smooth = NULL,
  digits = 2,
  alpha = NA,
  group.by = "row",
  reduce.data = FALSE,
  ylab = NULL,
  first.derivative = FALSE,
  ...
)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>Object obtained with the function <code>predictCox</code>.</td>
</tr>
<tr>
<td>type</td>
<td>[character] The type of predicted value to display. Choices are: &quot;hazard&quot; the hazard function, &quot;cumhazard&quot; the cumulative hazard function, or &quot;survival&quot; the survival function.</td>
</tr>
<tr>
<td>ci</td>
<td>[logical] If TRUE display the confidence intervals for the predictions.</td>
</tr>
<tr>
<td>band</td>
<td>[logical] If TRUE display the confidence bands for the predictions.</td>
</tr>
<tr>
<td>plot</td>
<td>[logical] Should the graphic be plotted.</td>
</tr>
<tr>
<td>smooth</td>
<td>[logical] Should a smooth version of the risk function be plotted instead of a simple function?</td>
</tr>
</tbody>
</table>
autoplot.predictCox

digits [integer] Number of decimal places when displaying the values of the covariates in the caption.
alpha [numeric, 0-1] Transparency of the confidence bands. Argument passed to ggplot2::geom_ribbon.
group.by [character] The grouping factor used to color the prediction curves. Can be "row", "strata", or "covariates".
reduce.data [logical] If TRUE only the covariates that does take identical values for all observations are displayed.

... Additional parameters to customize the display.

Value

Invisible. A list containing:

- plot: the ggplot object.
- data: the data used to create the plot.

See Also

predictCox to compute cumulative hazard and survival based on a Cox model.

Examples

library(survival)
library(ggplot2)

#### simulate data ####
set.seed(10)
d <- sampleData(1e2, outcome = "survival")
seqTau <- c(0, sort(unique(d$time[d$event==1])), max(d$time))

#### Cox model ####
m.cox <- coxph(Surv(time,event)~ X1 + X2 + X3,
data = d, x = TRUE, y = TRUE)

## display baseline hazard
e.basehaz <- predictCox(m.cox)
autoplot(e.basehaz, type = "cumhazard")
## Not run:
autoplot(e.basehaz, type = "cumhazard", size.point = 0) ## without points
autoplot(e.basehaz, type = "cumhazard", smooth = TRUE)
autoplot(e.basehaz, type = "cumhazard", smooth = TRUE, first.derivative = TRUE)

## End(Not run)
## display baseline hazard with type of event
## Not run:
```r
e.basehaz <- predictCox(m.cox, keep.newdata = TRUE)
autoplot(e.basehaz, type = "cumhazard")
autoplot(e.basehaz, type = "cumhazard", shape.point = c(3,NA))
```
## End(Not run)

## display predicted survival
## Not run:
pred.cox <- predictCox(m.cox, newdata = d[1:2,],
  times = seqTau, type = "survival", keep.newdata = TRUE)
autoplot(pred.cox)
autoplot(pred.cox, smooth = TRUE)
autoplot(pred.cox, group.by = "covariates")
autoplot(pred.cox, group.by = "covariates", reduce.data = TRUE)
autoplot(pred.cox, group.by = "X1", reduce.data = TRUE)
## End(Not run)

## predictions with confidence interval/bands
## Not run:
pred.cox <- predictCox(m.cox, newdata = d[1:2,,drop=FALSE],
  times = seqTau, type = "survival", band = TRUE, se = TRUE, keep.newdata = TRUE)
res <- autoplot(pred.cox, ci = TRUE, band = TRUE, plot = FALSE)
res$plot + facet_wrap(~row)
res2 <- autoplot(pred.cox, ci = TRUE, band = TRUE, alpha = 0.1, plot = FALSE)
res2$plot + facet_wrap(~row)
## End(Not run)

#### Stratified Cox model ####
## Not run:
m.cox.strata <- coxph(Surv(time,event)~ strata(X1) + strata(X2) + X3 + X4,
data = d, x = TRUE, y = TRUE)

## baseline hazard
pred.baseline <- predictCox(m.cox.strata, keep.newdata = TRUE, type = "survival")
res <- autoplot(pred.baseline)
res$plot + facet_wrap(~strata, labeller = label_both)

## predictions
pred.cox.strata <- predictCox(m.cox.strata, newdata = d[1:3,,drop=FALSE],
  time = seqTau, keep.newdata = TRUE, se = TRUE)
```
```
res2 <- autoplot(pred.cox.strata, type = "survival", group.by = "strata", plot = FALSE)
res2$plot + facet_wrap(~strata, labeller = label_both) + theme(legend.position="bottom")
```

## smooth version
autoplot(pred.cox.strata, type = "survival", group.by = "strata", smooth = TRUE, ci = FALSE)
## End(Not run)
Plot Predictions From a Cause-specific Cox Proportional Hazard Regression

Description

Plot predictions from a Cause-specific Cox proportional hazard regression.

Usage

```r
## S3 method for class 'predictCSC'
autoplot(
  object,
  ci = object$se,
  band = object$band,
  plot = TRUE,
  smooth = FALSE,
  digits = 2,
  alpha = NA,
  group.by = "row",
  reduce.data = FALSE,
  ...
)
```

Arguments

- **object**: Object obtained with the function `predictCox`.
- **ci**: [logical] If TRUE display the confidence intervals for the predictions.
autoplot.predictCSC

band     [logical] If TRUE display the confidence bands for the predictions.
plot     [logical] Should the graphic be plotted.
smooth   [logical] Should a smooth version of the risk function be plotted instead of a simple function?
digits   [integer] Number of decimal places.
alphas   [numeric, 0-1] Transparency of the confidence bands. Argument passed to ggplot2::geom_ribbon.
group.by [character] The grouping factor used to color the prediction curves. Can be "row", "strata", or "covariates".
reduce.data [logical] If TRUE only the covariates that does take indentical values for all observations are displayed.
... Additional parameters to customize the display.

Value

Invisible. A list containing:

- plot: the ggplot object.
- data: the data used to create the plot.

See Also

predict.CauseSpecificCox to compute risks based on a CSC model.

Examples

library(survival)
library(rms)
library(ggplot2)
library(prodlim)

#### simulate data ####
set.seed(10)
d <- sampleData(1e2, outcome = "competing.risks")
seqTau <- c(0, unique(sort(d[d$event==1,time])), max(d$time))

#### CSC model ####
m.CSC <- CSC(Hist(time,event)~ X1 + X2 + X6, data = d)
pred.CSC <- predict(m.CSC, newdata = d[1:2,], time = seqTau, cause = 1, band = TRUE)
autoplot(pred.CSC, alpha = 0.2)

#### stratified CSC model ####
m.SCSC <- CSC(Hist(time,event)~ strata(X1) + strata(X2) + X6,
data = d)
pred.SCSC <- predict(m.SCSC, time = seqTau, newdata = d[1:4,],
cause = 1, keep.newdata = TRUE, keep.strata = TRUE)
autoplot(pred.SCSC, group.by = "strata")
### Description

ggplot AUC curves

### Usage

```r
## S3 method for class 'Score'
autoplot(
  object,
  models,
  type = "score",
  lwd = 2,
  xlim,
  ylim,
  axes = TRUE,
  conf.int = FALSE,
  ...
)
```

### Arguments

- `object` Object obtained with `Score.list`
- `models` Choice of models to plot
- `type` Character. Either "score" to show AUC or "contrasts" to show differences between AUC.
- `lwd` Line width
- `xlim` Limits for x-axis
- `ylim` Limits for y-axis
- `axes` Logical. If TRUE draw axes.
- `conf.int` Logical. If TRUE draw confidence shadows.
- `...` Not yet used

### Examples

```r
library(survival)
library(ggplot2)
set.seed(10)
d=sampleData(100,outcome="survival")
nd=sampleData(100,outcome="survival")
f1=coxph(Surv(time,event)~X1+X6+X8,data=d,x=TRUE,y=TRUE)
f2=coxph(Surv(time,event)~X2+X5+X9,data=d,x=TRUE,y=TRUE)
xx=Score(list(f1,f2), formula=Surv(time,event)-1,
```
```r
data = nd, metrics = "auc", null.model = FALSE, times = seq(3:10))
g <- autoplot(xx)
print(g)
aucgraph <- plotAUC(xx)
plotAUC(xx, conf.int = TRUE)
plotAUC(xx, which = "contrasts")
plotAUC(xx, which = "contrasts", conf.int = TRUE)
```

---

### baseHaz_cpp

**C++ Fast Baseline Hazard Estimation**

#### Description

C++ function to estimate the baseline hazard from a Cox Model

#### Usage

```r
baseHaz_cpp(
  starttimes,
  stoptimes,
  status,
  eXb,
  strata,
  predtimes,
  emaxtimes,
  nPatients,
  nStrata,
  cause,
  Efron
)
```

#### Arguments

- `starttimes`: a vector of times (begin at risk period).
- `stoptimes`: a vector of times (end at risk period).
- `status`: a vector indicating censoring or event.
- `eXb`: a numeric vector (exponential of the linear predictor).
- `strata`: a vector of integers (index of the strata for each observation).
- `predtimes`: a vector of times (time at which to evaluate the hazard). Must be sorted.
- `emaxtimes`: another vector of times, one per strata (last observation time in each strata).
- `nPatients`: number of observations.
- `nStrata`: number of strata
- `cause`: the status value corresponding to event.
- `Efron`: whether Efron or Breslow estimator should be used in presence of ties.
Details

WARNING stoptimes status eXb and strata must be sorted by strata, stoptimes, and status

boot2pvalue

Compute the p.value from the distribution under H1

Description

Compute the p.value associated with the estimated statistic using a bootstrap sample of its distribution under H1.

Usage

boot2pvalue(
  x,
  null,
  estimate = NULL,
  alternative = "two.sided",
  FUN.ci = quantileCI,
  tol = .Machine$double.eps^0.5
)

Arguments

x [numeric vector] a vector of bootstrap estimates of the statistic.
null [numeric] value of the statistic under the null hypothesis.
estimate [numeric] the estimated statistic.
alternative [character] a character string specifying the alternative hypothesis, must be one of "two.sided" (default), "greater" or "less".
FUN.ci [function] the function used to compute the confidence interval. Must take x, alternative, conf.level and sign.estimate as arguments and only return the relevant limit (either upper or lower) of the confidence interval.
tol [numeric] the absolute convergence tolerance.

Details

For test statistic close to 0, this function returns 1.

For positive test statistic, this function search the quantile alpha such that:

- quantile(x, probs = alpha)=0 when the argument alternative is set to "greater".
- quantile(x, probs = 0.5*alpha)=0 when the argument alternative is set to "two.sided".
If the argument alternative is set to "less", it returns 1.

For negative test statistic, this function search the quantile alpha such that:

- quantile(x, probs = 1-alpha=0 when the argument alternative is set to "less".
- quantile(x, probs = 1-0.5*alpha=0 when the argument alternative is set to "two.sided".

If the argument alternative is set to "greater", it returns 1.

Examples

```r
set.seed(10)

#### no effect ####
 x <- rnorm(1e3)
boot2pvalue(x, null = 0, estimate = mean(x), alternative = "two.sided")
## expected value of 1
boot2pvalue(x, null = 0, estimate = mean(x), alternative = "greater")
## expected value of 0.5
boot2pvalue(x, null = 0, estimate = mean(x), alternative = "less")
## expected value of 0.5

#### positive effect ####
 x <- rnorm(1e3, mean = 1)
boot2pvalue(x, null = 0, estimate = 1, alternative = "two.sided")
## expected value of 0.32 = 2*pnorm(q = 0, mean = -1) = 2*mean(x<=0)
boot2pvalue(x, null = 0, estimate = 1, alternative = "greater")
## expected value of 0.16 = pnorm(q = 0, mean = 1) = mean(x<=0)
boot2pvalue(x, null = 0, estimate = 1, alternative = "less")
## expected value of 0.84 = 1-pnorm(q = 0, mean = 1) = mean(x>=0)

#### negative effect ####
 x <- rnorm(1e3, mean = -1)
boot2pvalue(x, null = 0, estimate = -1, alternative = "two.sided")
## expected value of 0.32 = 2*(1-pnorm(q = 0, mean = -1)) = 2*mean(x>=0)
boot2pvalue(x, null = 0, estimate = -1, alternative = "greater")
## expected value of 0.84 = pnorm(q = 0, mean = -1) = mean(x<=0)
boot2pvalue(x, null = 0, estimate = -1, alternative = "less") # pnorm(q = 0, mean = -1)
## expected value of 0.16 = 1-pnorm(q = 0, mean = -1) = mean(x>=0)
```

boxplot.Score

Boxplot risk quantiles

Description

Retrospective boxplots of risk quantiles conditional on outcome
Usage

```r
## S3 method for class 'Score'
boxplot(
  x,
  model,
  reference,
  type = "risk",
  timepoint,
  overall = 1L,
  lwd = 3,
  xlim,
  xlab = "",
  main,
  outcome.label,
  outcome.label.offset = 0,
  event.labels,
  refline = (type != "risk"),
  add = FALSE,
  ...
)
```

Arguments

- **x**: Score object obtained by calling function `Score`.
- **model**: Choice of risk prediction model.
- **reference**: Choice of reference risk prediction model for calculation of risk differences.
- **type**: Either "risk" for predicted risks or "diff" for differences between predicted risks.
- **timepoint**: Time point specifying the prediction horizon.
- **overall**: Logical. Tag to be documented.
- **lwd**: Line width.
- **xlim**: X-axis limits.
- **xlab**: X-axis label.
- **main**: Title of plot.
- **outcome.label**: Title label for column which shows the outcome status.
- **outcome.label.offset**: Vertical offset for outcome.label.
- **event.labels**: Labels for the different events (causes).
- **refline**: Logical, for type="diff" only. If TRUE draw a red vertical line at 0.
- **add**: Logical. Tag to be documented.
- **...**: Not used.
Examples

# binary outcome
library(data.table)
library(prodlim)
set.seed(10)
db=sampleData(40,outcome="binary")
fitconv=glm(Y~X3+X5,data=db,family=binomial)
fitnew=glm(Y~X1+X3+X5+X6+X7,data=db,family=binomial)
x=Score(list(new=fitnew,conv=fitconv),
       formula=Y~1,contrasts=list(c(2,1)),
data=db,plots="box",null.model=FALSE)
boxplot(x)

# survival outcome
library(survival)
ds=sampleData(40,outcome="survival")
fit=coxph(Surv(time,event)~X6+X9,data=ds,x=TRUE,y=TRUE)
## Not run:
scoreobj=Score(list("Cox"=fit),
              formula=Hist(time,event)~1, data=ds,
              metrics=NULL, plots="box",
              times=c(1,5),null.model=FALSE)
boxplot(scoreobj,timepoint=5)
boxplot(scoreobj,timepoint=1)
## End(Not run)

# competing risks outcome
library(survival)
data(Melanoma, package = "riskRegression")
fit = CSC(Hist(time,event,cens.code="censored")~invasion+age+sex,data=Melanoma)
scoreobj=Score(list("CSC"=fit),
              formula=Hist(time,event,cens.code="censored")~1,
data=Melanoma,plots="box",times=5*365.25,null.model=FALSE)
par(mar=c(4,12,4,4))
boxplot(scoreobj,timepoint=5*365.25)

# more than 2 competing risks
m=lava::lvm(~X1+X2+X3)
lava::distribution(m, "eventtime1") <- lava::coxWeibull.lvm(scale = 1/100)
lava::distribution(m, "eventtime2") <- lava::coxWeibull.lvm(scale = 1/100)
lava::distribution(m, "eventtime3") <- lava::coxWeibull.lvm(scale = 1/100)
lava::distribution(m, "censtime") <- lava::coxWeibull.lvm(scale = 1/100)
lava::regression(m,eventtime2~X3)=1.3
m <- lava::eventTime(m,
              time ~ min(eventtime1 = 1, eventtime2 = 2, eventtime3 = 3, censtime = 0), "event")
set.seed(101)
dcr=as.data.table(lava::sim(m,101))
fit = CSC(Hist(time,event)~X1+X2+X3,data=dcr)
scoreobj=Score(list("my model"=fit),
              formula=Hist(time,event)~1,
calcSeCox

```r
data=dcr, plots="box", times=5, null.model=FALSE)
boxplot(scoreobj)
```

calcSeCox  Computation of standard errors for predictions

Description

Compute the standard error associated to the predictions from Cox regression model using a first order von Mises expansion of the functional (cumulative hazard or survival).

Usage

```r
calcSeCox(
  object,
  times,
  nTimes,
  type,
  diag,
  Lambda0,
  object.n,
  object.time,
  object.eXb,
  object.strata,
  nStrata,
  new.n,
  new.eXb,
  new.LPdata,
  new.strata,
  new.survival,
  nVar.lp,
  export,
  store.iid
)
```

Arguments

- **object**: The fitted Cox regression model object either obtained with coxph (survival package) or cph (rms package).
- **times**: Vector of times at which to return the estimated hazard/survival.
- **nTimes**: the length of the argument times.
- **type**: One or several strings that match (either in lower or upper case or mixtures) one or several of the strings "hazard", "cumhazard", "survival".
diag  [logical] when FALSE the hazard/cumulative hazard/survival for all observations at all times is computed, otherwise it is only computed for the i-th observation at the i-th time.

Lambda0  the baseline hazard estimate returned by BaseHazStrata_cpp.

object.n  the number of observations in the dataset used to estimate the object.

object.time  the time to event of the observations used to estimate the object.

object.eXb  the exponential of the linear predictor relative to the observations used to estimate the object.

object.strata  the strata index of the observations used to estimate the object.

nStrata  the number of strata.

new.n  the number of observations for which the prediction was performed.

new.eXb  the linear predictor evaluated for the new observations.

new(LP)data  the variables involved in the linear predictor for the new observations.

new.strata  the strata indicator for the new observations.

new.survival  the survival evaluated for the new observations.

nVar.lp  the number of variables that form the linear predictor.

export  can be "iid" to return the value of the influence function for each observation. "se" to return the standard error for a given timepoint. "average.iid" to return the value of the average influence function over the observations for which the prediction was performed.

store.iid  Implementation used to estimate the influence function and the standard error. Can be "full" or "minimal". See the details section.

Details

store.iid="full" compute the influence function for each observation at each time in the argument times before computing the standard error / influence functions. store.iid="minimal" recompute for each subject specific prediction the influence function for the baseline hazard. This avoid to store all the influence functions but may lead to repeated evaluation of the influence function. This solution is therefore more efficient in memory usage but may not be in terms of computation time.

Value

A list optionally containing the standard error for the survival, cumulative hazard and hazard.

Author(s)

Brice Ozenne broz@sund.ku.dk, Thomas A. Gerds tag@biostat.ku.dk
calcSeCSC  

Standard error of the absolute risk predicted from cause-specific Cox models

Description

Standard error of the absolute risk predicted from cause-specific Cox models using a first order von Mises expansion of the absolute risk functional.

Usage

calcSeCSC(
  object,
  cif,
  hazard,
  cumhazard,
  survival,
  object.time,
  object.maxtime,
  eXb,
  new.LPdata,
  new.strata,
  times,
  surv.type,
  ls.infoVar,
  new.n,
  cause,
  nCause,
  nVar.lp,
  export,
  store.iid,
  diag
)

Arguments

object  The fitted cause specific Cox model
cif     the cumulative incidence function at each prediction time for each individual.
hazard  list containing the baseline hazard for each cause in a matrix form. Columns correspond to the strata.
cumhazard list containing the cumulative baseline hazard for each cause in a matrix form. Columns correspond to the strata.
survival list containing the (all cause) survival in a matrix form at t-. Columns correspond to event times.
object.time a vector containing all the events regardless to the cause.
object.maxtime  a matrix containing the latest event in the strata of the observation for each cause.
eXb       a matrix containing the exponential of the linear predictor evaluated for the new
observations (rows) for each cause (columns)
new.LPdata  a list of design matrices for the new observations for each cause.
new.strata  a matrix containing the strata indicator for each observation and each cause.
times       the time points at which to evaluate the predictions.
surv.type   see the surv.type argument of CSC.
ls.infoVar  A list containing the output of coxVariableName for each Cox model.
new.n        the number of new observations.
cause        the cause of interest.
nCause       the number of causes.
nVar.lp      the number of variables that form the linear predictor in each Cox model
export       can be "iid" to return the value of the influence function for each observation
"se" to return the standard error for a given timepoint
store.iid    the method used to compute the influence function and the standard error. Can
be "full" or "minimal". See the details section.
diag         [logical] when FALSE the absolute risk/survival for all observations at all times
is computed, otherwise it is only computed for the i-th observation at the i-th

Details
Can also return the empirical influence function of the functionals cumulative hazard or survival or
the sum over the observations of the empirical influence function.

store.iid="full" compute the influence function for each observation at each time in the arg-
ument times before computing the standard error/influence functions. store.iid="minimal"
recompute for each subject specific prediction the influence function for the baseline hazard. This
avoid to store all the influence functions but may lead to repeated evaluation of the influence func-
tion. This solution is therefore efficient more efficient in memory usage but may not be in term of
computation time.

Cforest  

S3-wrapper function for cforest from the party package

Description
S3-wrapper function for cforest from the party package

Usage
Cforest(formula, data, ...)
Arguments

- `formula` Passed on as is. See `cforest` of the party package
- `data` Passed on as is. See `cforest` of the party package
- `...` Passed on as they are. See `cforest` of the party package

Details

See `cforest` of the party package.

Value

list with two elements: `cforest` and `call`

References


---

calc.CauseSpecificCox  Extract coefficients from a Cause-Specific Cox regression model

Description

Extract coefficients from a Cause-Specific Cox regression model

Usage

```r
## S3 method for class 'CauseSpecificCox'
coef(object, ...)
```

Arguments

- `object` Object obtained with CSC
- `...` not used
### coef.riskRegression

*Extract coefficients from riskRegression model*

**Description**

Extract coefficients from riskRegression model

**Usage**

```r
## S3 method for class 'riskRegression'
coef(object, digits = 3, eps = 10^-4, ...)
```

**Arguments**

- `object`: Object obtained with ARR or LRR or riskRegression
- `digits`: Number of digits
- `eps`: P-values below this number are shown as <eps
- `...`: not used

### colCenter_cpp

*Apply - by column*

**Description**

Fast computation of sweep(X, MARGIN = 1, FUN = ",-", STATS = center)

**Usage**

```r
colCenter_cpp(X, center)
```

**Arguments**

- `X`: A matrix.
- `center`: a numeric vector of length equal to the number of rows of `X`

**Value**

A matrix of same size as `X`.

**Author(s)**

Brice Ozenne <broz@sund.ku.dk>

**Examples**

```r
x <- matrix(1,6,5)
sweep(x, MARGIN = 1, FUN = ",-", STATS = 1:6)
colCenter_cpp(x, 1:6 )
```
colCumSum

Apply cumsum in each column

Description
Fast computation of apply(x,2,cumsum)

Usage
colCumSum(x)

Arguments
x A matrix.

Value
A matrix of same size as x.

Author(s)
Thomas Alexander Gerds <tag@biostat.ku.dk>

Examples
x <- matrix(1:8,ncol=2)
colCumSum(x)

colMultiply_cpp

Apply * by column

Description
Fast computation of sweep(X, MARGIN = 1, FUN = ", STATS = scale)

Usage
colMultiply_cpp(X, scale)

Arguments
X A matrix.
scale a numeric vector of length equal to the number of rows of x

Value
A matrix of same size as X.
Author(s)

Brice Ozenne <broz@sund.ku.dk>

Examples

```R
x <- matrix(1, 6, 5)
sweep(x, MARGIN = 1, FUN = "+", STATS = 1:6)
colMultiply_cpp(x, 1:6)
```

**Description**

Fast computation of sweep(X, MARGIN = 1, FUN = "/", STATS = scale)

**Usage**

```R
colScale_cpp(X, scale)
```

**Arguments**

- `X` A matrix.
- `scale` a numeric vector of length equal to the number of rows of `x`

**Value**

A matrix of same size as `X`.

Author(s)

Brice Ozenne <broz@sund.ku.dk>

Examples

```R
x <- matrix(1, 6, 5)
sweep(x, MARGIN = 1, FUN = "/", STATS = 1:6)
colScale_cpp(x, 1:6)
```
null
meanRisk.transform
[character] the transformation used to improve coverage of the confidence intervals for the mean risk in small samples. Can be "none", "log", "loglog", "cloglog".

diffRisk.transform
[character] the transformation used to improve coverage of the confidence intervals for the risk difference in small samples. Can be "none", "atanh".

ratioRisk.transform
[character] the transformation used to improve coverage of the confidence intervals for the risk ratio in small samples. Can be "none", "log".

seed
[integer, >0] seed number set when performing simulation for the confidence bands. If not given or NA no seed is set.

ci
[logical] should the confidence intervals be computed?

band
[logical] should the confidence bands be computed?

p.value
[logical] should the p-values/adjusted p-values be computed? Requires argument ci and/or band to be TRUE.

method.band
[character] method used to adjust for multiple comparisons. Can be any element of p.adjust.methods (e.g. "holm"), "maxT-integration", or "maxT-simulation".

alternative
[character] a character string specifying the alternative hypothesis, must be one of "two.sided" (default), "greater" or "less".

bootci.method
[character] Method for constructing bootstrap confidence intervals. Either "perc" (the default), "norm", "basic", "stud", or "bca".

... not used.

Details
Argument ci, band, p.value, method.band, alternative, meanRisk.transform, diffRisk.transform, ratioRisk.transform are only active when the ate object contains the influence function. Argument bootci.method is only active when the ate object contains bootstrap samples.

Influence function: confidence bands and confidence intervals computed via the influence function are automatically restricted to the interval of definition of the parameter (e.g. [0;1] for the average risk). Single step max adjustment for multiple comparisons, i.e. accounting for the correlation between the test statistics but not for the ordering of the tests, can be performed setting the argument method.band to "maxT-integration" or "maxT-simulation". The former uses numerical integration (pmvnorm and qmvnorm to perform the adjustment while the latter using simulation. Both assume that the test statistics are jointly normally distributed.

Bootstrap: confidence intervals obtained via bootstrap are computed using the boot.ci function of the boot package. p-value are obtained using test inversion method (finding the smallest confidence level such that the interval contain the null hypothesis).

Author(s)
Brice Ozenne
Examples

```r
library(survival)
library(data.table)

## ## generate data ####
set.seed(10)
d <- sampleData(70, outcome="survival")
d[, X1 := paste0("T", rbinom(.N, size = 2, prob = c(0.51)))]
## table(d$X1)

#### stratified Cox model ####
fit <- coxph(Surv(time, event) ~ X1 + strata(X2) + X6, 
data=d, ties="breslow", x = TRUE, y = TRUE)

#### average treatment effect ####
fit.ate <- ate(fit, treatment = "X1", times = 1:3, data = d, 
se = TRUE, iid = TRUE, band = TRUE)
summary(fit.ate)
dt.ate <- as.data.table(fit.ate)

## manual calculation of se
dd <- data.table::copy(d)
dd$X1 <- rep(factor("T0", levels = paste0("T",0:2)), NROW(dd))
out <- predictCox(fit, newdata = dd, se = TRUE, times = 1:3, average.iid = TRUE)
term1 <- -out$survival.average.iid
term2 <- sweep(1-out$survival, MARGIN = 2, FUN = 
FUN = ", STATS = colMeans(1-out$survival))
sqrt(colSums((term1 + term2/NROW(d))^2))
## fit.ate$meanRisk[treatment="T0",se]

## note
out2 <- predictCox(fit, newdata = dd, se = TRUE, times = 1:3, iid = TRUE)
mean(out2$survival.iid[1,1])
out$survival.average.iid[1,1]

## check confidence intervals (no transformation)
dt.ate[, .(lower = pmax(0, estimate + qnorm(0.025) * se), 
upper = estimate + qnorm(0.975) * se),
]

## add confidence intervals computed on the log-log scale
## and backtransformed
outCI <- confint(fit.ate, 
meanRisk.transform = "loglog", diffRisk.transform = "atanh",
ratioRisk.transform = "log")
summary(outCI, type = "risk", short = TRUE)
dt.ate[type == "meanRisk", newse := se/(estimate*log(estimate))]
dt.ate[type == "meanRisk", .(lower = exp(-exp(log(-log(estimate)) - 1.96 * newse)),
upper = exp(-exp(log(-log(estimate)) + 1.96 * newse)))]
```
Description

Confidence intervals and confidence Bands for the difference between two estimates.

Usage

```r
## S3 method for class 'influenceTest'
confint(
  object,
  parm = NULL,
  level = 0.95,
  n.sim = 10000,
  transform = "none",
  seed = NA,
  ...
)
```

Arguments

- **object**: A `influenceTest` object, i.e. output of the `influenceTest` function.
- **parm**: not used. For compatibility with the generic method.
- **level**: [numeric, 0-1] Level of confidence.
- **n.sim**: [integer, >0] the number of simulations used to compute the quantiles for the confidence bands.
- **transform**: [character] the transformation used to improve coverage of the confidence intervals. Can be "none" or "atanh".
- **seed**: [integer, >0] seed number set before performing simulations for the confidence bands. If not given or NA no seed is set.
- **...**: not used.

Details

Except for the cumulative hazard, the confidence bands and confidence intervals are automatically restricted to the interval [−1;1].

Author(s)

Brice Ozenne
confint.predictCox  Confidence Intervals and Confidence Bands for the predicted Survival/Cumulative Hazard

Description

Confidence intervals and confidence Bands for the predicted survival/cumulative Hazard.

Usage

```r
## S3 method for class 'predictCox'
confint(
  object,
  parm = NULL,
  level = 0.95,
  n.sim = 10000,
  cumhazard.transform = "log",
  survival.transform = "loglog",
  seed = NA,
  ...
)
```

Arguments

- **object**: A `predictCox` object, i.e. output of the `predictCox` function.
- **parm**: [character] the type of predicted value for which the confidence intervals should be output. Can be "survival" or "cumhazard".
- **level**: [numeric, 0-1] Level of confidence.
- **n.sim**: [integer, >0] the number of simulations used to compute the quantiles for the confidence bands.
- **cumhazard.transform**: [character] the transformation used to improve coverage of the confidence intervals for the cumulative hazard in small samples. Can be "none", "log".
- **survival.transform**: [character] the transformation used to improve coverage of the confidence intervals for the survival in small samples. Can be "none", "log", "loglog", "cloglog".
- **seed**: [integer, >0] seed number set before performing simulations for the confidence bands. If not given or NA no seed is set.
- **...**: not used.

Details

The confidence bands and confidence intervals are automatically restricted to the interval of definition of the statistic, i.e. a confidence interval for the survival of [0.5;1.2] will become [0.5;1].
**Author(s)**

Brice Ozenne

**Examples**

```r
library(survival)

#### generate data ####
set.seed(10)
d <- sampleData(40, outcome="survival")

#### estimate a stratified Cox model ####
fit <- coxph(Surv(time,event)~X1 + strata(X2) + X6,
data=d, ties="breslow", x = TRUE, y = TRUE)

#### compute individual specific survival probabilities
fit.pred <- predictCox(fit, newdata=d[1:3], times=c(3,8), type = "survival",
se = TRUE, iid = TRUE, band = TRUE)
fit.pred

## check standard error
sqrt(rowSums(fit.pred$survival.iid[,,1]^2)) ## se for individual 1

## check confidence interval
newse <- fit.pred$survival.se/(-fit.pred$survival*log(fit.pred$survival))
cbind(lower = as.double(exp(-exp(log(-log(fit.pred$survival)) + 1.96 * newse))),
    upper = as.double(exp(-exp(log(-log(fit.pred$survival)) - 1.96 * newse)))
)

#### compute confidence intervals without transformation
confint(fit.pred, survival.transform = "none")
cbind(lower = as.double(fit.pred$survival - 1.96 * fit.pred$survival.se),
    upper = as.double(fit.pred$survival + 1.96 * fit.pred$survival.se)
)
```

---

**confint.predictCSC**

Confidence Intervals and Confidence Bands for the Predicted Absolute Risk (Cumulative Incidence Function)

**Description**

Confidence intervals and confidence Bands for the predicted absolute risk (cumulative incidence function).

**Usage**

```r
## S3 method for class 'predictCSC'
confint(
```
object,  
parm = NULL,  
level = 0.95,  
n.sim = 10000,  
absRisk.transform = "loglog",  
seed = NA,  
...  
)

Arguments

object  A predictCSC object, i.e. output of the predictCSC function.
parm  not used. For compatibility with the generic method.
level  \([\text{numeric, }0-1]\) Level of confidence.
n.sim  \([\text{integer, }>0]\) the number of simulations used to compute the quantiles for the confidence bands.
absRisk.transform  \([\text{character}]\) the transformation used to improve coverage of the confidence intervals for the predicted absolute risk in small samples. Can be "none", "log", "loglog", "cloglog".
seed  \([\text{integer, }>0]\) seed number set before performing simulations for the confidence bands. If not given or NA no seed is set.
...  not used.

Details

The confidence bands and confidence intervals are automatically restricted to the interval \([0;1]\).

Author(s)

Brice Ozenne

Examples

library(survival)
library(prodlim)

#### generate data ####
s.set.seed(10)
d <- sampleData(100)

#### estimate a stratified CSC model ####
fit <- CSC(Hist(time, event) ~ X1 + strata(X2) + X6, data=d)

#### compute individual specific risks ####
fit.pred <- predict(fit, newdata=d[1:3], times=c(3, 8), cause = 1,
          se = TRUE, iid = TRUE, band = TRUE)
fit.pred

# check confidence intervals
newse <- fit.pred$absRisk.se/(-fit.pred$absRisk*log(fit.pred$absRisk))
cbind(lower = as.double(exp(-exp(log(-log(fit.pred$absRisk)) + 1.96 * newse))),
       upper = as.double(exp(-exp(log(-log(fit.pred$absRisk)) - 1.96 * newse)))
)

### compute confidence intervals without transformation
confint(fit.pred, absRisk.transform = "none")
cbind(lower = as.double(fit.pred$absRisk - 1.96 * fit.pred$absRisk.se),
       upper = as.double(fit.pred$absRisk + 1.96 * fit.pred$absRisk.se)
)

---

**coxBaseEstimator**  
*Extract the type of estimator for the baseline hazard*

**Description**

Extract the type of estimator for the baseline hazard

**Usage**

```r
coxBaseEstimator(object)
```

```r
## S3 method for class 'coxph'
coxBaseEstimator(object)
```

```r
## S3 method for class 'phreg'
coxBaseEstimator(object)
```

**Arguments**

- `object`  
The fitted Cox regression model object either obtained with coxph (survival package), cph (rms package), or phreg (mets package).

**Author(s)**

Brice Ozenne broz@sund.ku.dk
**coxCenter**

*Extract the mean value of the covariates*

**Description**

Extract the mean value of the covariates

**Usage**

```r
coxCenter(object)
```

## S3 method for class 'cph'
coxCenter(object)

## S3 method for class 'coxph'
coxCenter(object)

## S3 method for class 'phreg'
coxCenter(object)

**Arguments**

- `object`: The fitted Cox regression model object either obtained with `coxph` (survival package), `cph` (rms package), or `phreg` (mets package).

**Author(s)**

Brice Ozenne broz@sund.ku.dk

---

**coxFormula**

*Extract the formula from a Cox model*

**Description**

Extract the formula from a Cox model

**Usage**

```r
coxFormula(object)
```

## S3 method for class 'cph'
coxFormula(object)

## S3 method for class 'coxph'
coxFormula(object)

## S3 method for class 'phreg'
coxFormula(object)
## S3 method for class 'phreg'
coxFormula(object)

## S3 method for class 'glm'
coxFormula(object)

### Arguments

- **object**: The fitted Cox regression model object either obtained with `coxph` (survival package), `cph` (rms package), or `phreg` (mets package).

### Author(s)

Brice Ozenne broz@sund.ku.dk

---

**coxLP**  
*Compute the linear predictor of a Cox model*

### Description

Compute the linear predictor of a Cox model

### Usage

`coxLP(object, data, center)`

### Arguments

- **object**: The fitted Cox regression model object either obtained with `coxph` (survival package), `cph` (rms package), or `phreg` (mets package).
- **data**: a `data.frame` or a `data.table`
- **center**: should the linear predictor be computed after centering the covariates

### Details

In case of empty linear predictor returns a vector of 0 with the same length as the number of rows of the dataset
**coxModelFrame**

**Description**
Extract the design matrix used to train a Cox model. Should contain the time of event, the type of event, the variable for the linear predictor, the strata variables and the date of entry (in case of delayed entry).

**Usage**

```r
coxModelFrame(object, center)
```

## S3 method for class 'coxph'

```r
coxModelFrame(object, center = FALSE)
```

## S3 method for class 'cph'

```r
coxModelFrame(object, center = FALSE)
```

## S3 method for class 'phreg'

```r
coxModelFrame(object, center = FALSE)
```

**Arguments**

- `object` : The fitted Cox regression model object either obtained with `coxph` (survival package), `cph` (rms package), or `phreg` (mets package).
- `center` : [logical] Should the variables of the linear predictor be added?

**Author(s)**
Brice Ozenne broz@sund.ku.dk

---

**coxN**

**Description**
Extract the number of observations from a Cox model

---

**Description**
Extract the number of observations from a Cox model
Usage

```r
coxN(object)
```

## S3 method for class 'coxph'

```r
coxN(object)
```

## S3 method for class 'cph'

```r
coxN(object)
```

## S3 method for class 'phreg'

```r
coxN(object)
```

## S3 method for class 'CauseSpecificCox'

```r
coxN(object)
```

## S3 method for class 'glm'

```r
coxN(object)
```

Arguments

- `object` The fitted Cox regression model object either obtained with `coxph` (survival package), `cph` (rms package), or `phreg` (mets package).

Author(s)

- Brice Ozenne broz@sund.ku.dk

Description

Return the special character(s) of the Cox model, e.g. used to indicate the strata variables.

Usage

```r
coxSpecial(object)
```

## S3 method for class 'coxph'

```r
ccoxSpecial(object)
```

## S3 method for class 'cph'

```r
ccoxSpecial(object)
```

## S3 method for class 'phreg'

```r
ccoxSpecial(object)
```


**coxStrata**

**Arguments**

- **object**: The fitted Cox regression model object either obtained with `coxph` (survival package), `cph` (rms package), or `phreg` (mets package).

**Details**

Must return a list with at least one element `strata` indicating the character in the formula marking the variable(s) defining the strata.

**Author(s)**

Brice Ozenne broz@sund.ku.dk

---

**coxStrata**

*Define the strata for a new dataset*

**Description**

Define the strata in a dataset to match those of a stratified Cox model

**Usage**

```r
coxStrata(object, data, sterms, strata.vars, strata.levels)
```

## S3 method for class `cph`
```r
coxStrata(object, data, sterms, strata.vars, strata.levels)
```

## S3 method for class `coxph`
```r
coxStrata(object, data, sterms, strata.vars, strata.levels)
```

## S3 method for class `phreg`
```r
coxStrata(object, data, sterms, strata.vars, strata.levels)
```

**Arguments**

- **object**: The fitted Cox regression model object either obtained with `coxph` (survival package), `cph` (rms package), or `phreg` (mets package).
- **data**: a data.frame or a data.table
- **sterms**: terms in the formula corresponding to the strata variables
- **strata.vars**: the name of the variables used to define the strata
- **strata.levels**: a named list containing for each variable used to form the strata all its possible levels
- **levels**: the strata levels that have been used to fit the Cox model
Details

if no strata variables returns a vector of "1" (factor).

Author(s)

Brice Ozenne broz@sund.ku.dk

---

coxStrataLevel

*Returns the name of the strata in Cox model*

Description

Return the name of the strata in Cox model

Usage

```r
coxStrataLevel(object)
```

## S3 method for class 'coxph'
coxStrataLevel(object)

## S3 method for class 'cph'
coxStrataLevel(object)

## S3 method for class 'phreg'
coxStrataLevel(object)

Arguments

- `object` The fitted Cox regression model object either obtained with coxph (survival package), cph (rms package), or phreg (mets package).

Author(s)

Brice Ozenne broz@sund.ku.dk
**Description**

Extract the variance covariance matrix of the beta from a Cox model

**Usage**

```r
coxVarCov(object)

## S3 method for class 'cph'
coxVarCov(object)

## S3 method for class 'coxph'
coxVarCov(object)

## S3 method for class 'phreg'
coxVarCov(object)
```

**Arguments**

- `object` The fitted Cox regression model object either obtained with `coxph` (survival package), `cph` (rms package), or `phreg` (mets package).

**Details**

Should return `NULL` if the Cox model has no covariate. The rows and columns of the variance covariance matrix must be named with the names used in the design matrix.

**Author(s)**

Brice Ozenne `broz@sund.ku.dk`

---

**Description**

Extract variable names from a model

**Usage**

```r
coxVariableName(object, model.frame)
```

**Details**

Extract the name of the variables belonging to the linear predictor or used to form the strata.
Arguments

object The fitted Cox regression model object either obtained with coxph (survival package) or cph (rms package).

c_model.frame [data.frame] dataset containing all the relevant variables (entry, time to event, type of event, variables in the linear predictor, strata). Output from coxModelFrame.

Author(s)

Brice Ozenne broz@sund.ku.dk

Description

Interface for fitting cause-specific Cox proportional hazard regression models in competing risk.

Usage

CSC(formula, data, cause, surv.type = "hazard", fitter = "coxph", ...)

Arguments

formula Either a single Hist formula or a list of formulas. If it is a list it must contain as many Hist formulas as there are causes when surv.type="hazard" and exactly two formulas when surv.type="survival". If it is a list the first formula is used for the cause of interest specific Cox regression and the other formula(s) either for the other cause specific Cox regression(s) or for the Cox regression of the combined event where each cause counts as event. Note that when only one formula is given the covariates enter in exactly the same way into all Cox regression analyses.

data A data in which to fit the models.

cause The cause of interest. Defaults to the first cause (see Details).

surv.type Either "hazard" (the default) or "survival". If "hazard" fit cause-specific Cox regression models for all causes. If "survival" fit one cause-specific Cox regression model for the cause of interest and also a Cox regression model for event-free survival.

fitter Routine to fit the Cox regression models. If coxph use survival::coxph else use rms::cph.

... Arguments given to fitter, e.g., coxph.

Details

The causes and their order are determined by prodlim::getStates() applied to the Hist object.
Value

- **models**: a list with the fitted (cause-specific) Cox regression objects
- **response**: the event history response
- **eventTimes**: the sorted (unique) event times
- **surv.type**: the value of surv.type
- **theCause**: the cause of interest. see cause
- **causes**: the other causes

Author(s)

Thomas A. Gerds <tag@biostat.ku.dk> and Ulla B. Mogensen

References


See Also

- coxph

Examples

```r
library(prodlim)
library(survival)
data(Melanoma)
## fit two cause-specific Cox models
## different formula for the two causes
fit1 <- CSC(list(Hist(time,status)~sex+age,Hist(time,status)~invasion+epicel+log(thick)),
            data=Melanoma)
print(fit1)
## Not run:
library(Publish)
publish(fit1)
## End(Not run)

## model hazard of all cause mortality instead of hazard of type 2
fit1a <- CSC(list(Hist(time,status)~sex+age,Hist(time,status)~invasion+epicel+log(thick)),
             data=Melanoma,
             surv.type="surv")

## the predicted probabilities are similar
```
plot(predictRisk(fit1, times=500, cause=1, newdata=Melanoma),
       predictRisk(fit1a, times=500, cause=1, newdata=Melanoma))

## special case where cause 2 has no covariates
fit1b <- CSC(list(Hist(time,status)~sex+age, Hist(time,status)~1),
              data=Melanoma)
print(fit1b)
predict(fit1b, cause=1, times=100, newdata=Melanoma)

## same formula for both causes
fit2 <- CSC(Hist(time,status)~invasion+epicel+age,
           data=Melanoma)
print(fit2)

## combine a cause-specific Cox regression model for cause 2
## and a Cox regression model for the event-free survival:
## different formula for cause 2 and event-free survival
fit3 <- CSC(list(Hist(time,status)~sex+invasion+epicel+age,
                Hist(time,status)~invasion+epicel+age),
            surv.type="surv",
            data=Melanoma)
print(fit3)

## same formula for both causes
fit4 <- CSC(Hist(time,status)~invasion+epicel+age,
           data=Melanoma,
           surv.type="surv")
print(fit4)

## strata
fit5 <- CSC(Hist(time,status)~invasion+epicel+age+strata(sex),
            data=Melanoma,
            surv.type="surv")
print(fit5)

## sanity checks
cox1 <- coxph(Surv(time,status==1)~invasion+epicel+age+strata(sex), data=Melanoma)
cox2 <- coxph(Surv(time,status!=0)~invasion+epicel+age+strata(sex), data=Melanoma)
all.equal(coef(cox1), coef(fit5$models[[1]]))
all.equal(coef(cox2), coef(fit5$models[[2]]))

## predictions
## surv.type = "hazard": predictions for both causes can be extracted
## from the same fit
fit2 <- CSC(Hist(time,status)~invasion+epicel+age, data=Melanoma)
predict(fit2, cause=1, newdata=Melanoma[c(17,99,108),], times=c(100,1000,10000))
predictRisk(fit2, cause=1, newdata=Melanoma[c(17,99,108),], times=c(100,1000,10000))
predictRisk(fit2, cause=2, newdata=Melanoma[c(17,99,108),], times=c(100,1000,10000))
predict(fit2, cause=1, newdata=Melanoma[c(17,99,108),], times=c(100,1000,10000))
predict(fit2, cause=2, newdata=Melanoma[c(17,99,108),], times=c(100,1000,10000))
## surv.type = "surv" we need to change the cause of interest
library(survival)
fit5.2 <- CSC(Hist(time,status)~invasion+epicel+age+strata(sex),
data=Melanoma,
surv.type="surv",cause=2)
## now this does not work
try(predictRisk(fit5.2,cause=1,newdata=Melanoma,times=4))

## but this does
predictRisk(fit5.2,cause=2,newdata=Melanoma,times=100)
predict(fit5.2,cause=2,newdata=Melanoma,times=100)
predict(fit5.2,cause=2,newdata=Melanoma[4,],times=100)

---

**Ctree**

S3-Wrapper for ctree.

### Description
The call is added to an ctree object

### Usage
Ctree(...)

### Arguments
... passed to ctree

### Value
list with two elements: ctree and call

### Author(s)
Thomas A. Gerds <tag@biostat.ku.dk>

### See Also
Cforest

### Examples
```r
if (require("party", quietly=TRUE)){
library(prodlim)
library(party)
library(survival)
set.seed(50)
d <- SimSurv(50)
```
```r
nd <- data.frame(X1=c(0,1,0),X2=c(-1,0,1))
f <- Ctree(Surv(time,status)~X1+X2,data=d)
predictRisk(f,newdata=nd,times=c(3,8))
}
```

---

**discreteRoot**

*Dichotomic search for monotone function*

**Description**

Find the root of a monotone function on a discrete grid of value using dichotomic search

**Usage**

```r
discreteRoot(
  fn,
  grid,
  increasing = TRUE,
  check = TRUE,
  tol = .Machine$double.eps^0.5
)
```

**Arguments**

- **fn** [function] objective function to minimize in absolute value.
- **grid** [vector] possible minimizers.
- **increasing** [logical] is the function fn increasing?
- **check** [logical] should the program check that fn takes a different sign for the first vs. the last value of the grid?
- **tol** [numeric] the absolute convergence tolerance.

---

**FGR**

*Formula wrapper for crr from cmprsk*

**Description**

Formula interface for Fine-Gray regression competing risk models.

**Usage**

```r
FGR(formula, data, cause = 1, y = TRUE, ...)
```
Arguments

- **formula**: A formula whose left hand side is a `Hist` object – see `Hist`. The right hand side specifies (a linear combination of) the covariates. See examples below.
- **data**: A data.frame in which all the variables of `formula` can be interpreted.
- **cause**: The failure type of interest. Defaults to 1.
- **y**: logical value: if TRUE, the response vector is returned in component `response`.
- **...**: ...

Details

Formula interface for the function `crr` from the `cmprsk` package.

The function `crr` allows to multiply some covariates by time before they enter the linear predictor. This can be achieved with the formula interface, however, the code becomes a little cumbersome. See the examples. Note that FGR does not allow for delayed entry (left-truncation). The assumed value for indicating censored observations in the event variable is 0. The function `Hist` has an argument `cens.code` which can change this (if you do not want to change the event variable).

Value

See `crr`.

Author(s)

Thomas Alexander Gerds <tag@biostat.ku.dk>

References

Gerds, TA and Scheike, T and Andersen, PK (2011) Absolute risk regression for competing risks: interpretation, link functions and prediction Research report 11/7. Department of Biostatistics, University of Copenhagen

See Also

- `riskRegression`

Examples

```r
library(prodlim)
library(survival)
library(cmprsk)
library(lava)
d <- prodlim::SimCompRisk(100)
f1 <- FGR(Hist(time,cause)~X1+X2,data=d)
print(f1)

## crr allows that some covariates are multiplied by
## a function of time (see argument tf of crr)
## by FGR uses the identity matrix
```
f2 <- FGR(Hist(time,cause)-cov2(X1)+X2,data=d)
print(f2)

## same thing, but more explicit:
f3 <- FGR(Hist(time,cause)-cov2(X1)+cov1(X2),data=d)
print(f3)

## both variables can enter cov2:
f4 <- FGR(Hist(time,cause)-cov2(X1)+cov2(X2),data=d)
print(f4)

## change the function of time
qFun <- function(x)(x^2)
noFun <- function(x){x}
qFun <- function(x)x^0.5

## multiply X1 by time^2 and X2 by time:
f5 <- FGR(Hist(time,cause)-cov2(X1,tf=qFun)+cov2(X2),data=d)
print(f5)
print(f5$crrFit)

## same results as crr
with(d,crr(ftime=time,
    fstatus=cause,
    cov2=d[,c("X1","X2")],
    tf=function(time){cbind(qFun(time),time)}))

## still same result, but more explicit
f5a <- FGR(Hist(time,cause)-cov2(X1,tf=qFun)+cov2(X2,tf=noFun),data=d)
f5a$crrFit

## multiply X1 by time^2 and X2 by sqrt(time)
f5b <- FGR(Hist(time,cause)-cov2(X1,tf=qFun)+cov2(X2,tf=sqFun),data=d,cause=1)

## additional arguments for crr
f6<- FGR(Hist(time,cause)-X1+X2,data=d, cause=1,gtol=1e-5)
f6
f6a<- FGR(Hist(time,cause)-X1+X2,data=d, cause=1,gtol=0.1)
f6a

---

gSplitMethod  

Input for data splitting algorithms

Description

Parse hyperparameters for data splitting algorithm

Usage

gSplitMethod(split.method, B, N, M, seed)
**getSplitMethod**

**Arguments**

split.method  A character string specifying the algorithm for data splitting:

- "loob" leave one out bootstrap
- "bootcv" bootstrap cross validation
- "cv5" 5-fold cross validation
- "loocv" leave one out cross validation aka N-1 fold cross validation
- "632plus" Efron’s .632+ bootstrap

B  Number of repetitions of bootstrap or k-fold cross-validation

N  Sample size

M  Subsample size. Default is N (no subsampling).

seed  Integer passed to set.seed. If not given or NA no seed is set.

**Value**

A list with the following elements:

- split.methodName: the print name of the algorithm
- split.method: the internal name of the algorithm
- index: the index for data splitting. For bootstrap splitting this is a matrix with B columns and M rows identifying the in-bag subjects. For k-fold cross-validation this is a matrix with B columns identifying the membership to the k groups.
- k: the k of k-fold cross-validation
- N: the sample size
- M: the subsample size

**Author(s)**

Thomas A. Gerds <tag@biostat.ku.dk>

**See Also**

Score

**Examples**

# 3-fold crossvalidation
getSplitMethod("cv3",B=4,N=37)

# bootstrap with replacement
getSplitMethod("loob",B=4,N=37)

# bootstrap without replacement
getSplitMethod("loob",B=4,N=37,M=20)
GLMnet

Fitting GLMnet for use with predictRisk

Description

Fit GLMnet models via a formula and a data set for use with predictRisk.

Usage

GLMnet(
    formula, 
    data, 
    lambda = NULL, 
    cv = TRUE, 
    alpha = 1, 
    nfolds = 10, 
    type.measure = "deviance", 
    family, 
    ...
)

Arguments

formula A formula.
data The data on which to fit the model.
lambda The tuning parameters for GLMnet. If set to NULL, then it the parameters are chosen for you.
cv Whether to use cross-validation or not. Default is TRUE.
alpha The elasticnet mixing parameter. See the ?glmnet for more details.
nfolds Number of folds for cross-validation. Default is 10.
type.measure loss to use for cross-validation. Default is deviance.
family passed to glmnet. Defaults for binary outcome to “binomial” and for survival to “cox”.
... Additional arguments that are passed on to the glmnet.
Fitting HAL for use with `predictRisk`

**Description**

Fit HAL models via a formula and a data set for use with `predictRisk`.

**Usage**

```r
Hal9001(formula, data, lambda = NULL, ...)
```

**Arguments**

- `formula` A formula.
- `data` The data on which to fit the model.
- `lambda` The tuning parameters for HAL. If set to NULL, then it the parameters are chosen for you.
- `...` Additional arguments that are passed on to the `hal_fit`.

---

**IID for IPCW Logistic Regressions**

**Description**

Compute the decomposition in iid elements of the ML estimor of IPCW logistic regressions.

**Usage**

```r
## S3 method for class 'wglm'
iid(x, times = NULL, simplifies = TRUE, ...)
```

**Arguments**

- `x` a `wglm` object.
- `times` [numeric vector] time points at which the iid should be output.
- `simplifies` [logical] should the output be converted to a matrix when only one timepoint is requested. Otherwise will always return a list.
- `...` Not used.
iidCox

Extract iid decomposition from a Cox model

Description

Compute the influence function for each observation used to estimate the model

Usage

iidCox(
  object,
  newdata,
  baseline.iid,
  tau.hazard,
  tau.max,
  store.iid,
  keep.times,
  return.object
)

## S3 method for class 'coxph'

iidCox(
  object,
  newdata = NULL,
  baseline.iid = TRUE,
  tau.hazard = NULL,
  tau.max = NULL,
  store.iid = "full",
  keep.times = TRUE,
  return.object = TRUE
)

## S3 method for class 'cph'

iidCox(
  object,
  newdata = NULL,
  baseline.iid = TRUE,
  tau.hazard = NULL,
  tau.max = NULL,
  store.iid = "full",
  keep.times = TRUE,
  return.object = TRUE
)

## S3 method for class 'phreg'

iidCox(
  object,
newdata = NULL,
baseline.iid = TRUE,
tau.hazard = NULL,
tau.max = NULL,
store.iid = "full",
keep.times = TRUE,
return.object = TRUE
)

## S3 method for class 'CauseSpecificCox'
iidCox(
  object,
  newdata = NULL,
  baseline.iid = TRUE,
  tau.hazard = NULL,
  tau.max = NULL,
  store.iid = "full",
  keep.times = TRUE,
  return.object = TRUE
)

Arguments

object  object The fitted Cox regression model object either obtained with coxph (survival package) or cph (rms package).
newdata [data.frame] Optional new data at which to do iid decomposition
baseline.iid [logical] Should the influence function for the baseline hazard be computed.
tau.hazard [numeric vector] the vector of times at which the i.i.d decomposition of the baseline hazard will be computed
tau.max [numeric] latest time at which the i.i.d decomposition of the baseline hazard will be computed. Alternative to tau.hazard.
store.iid [character] the method used to compute the influence function and the standard error. Can be "full" or "minimal". See the details section.
keep.times [logical] If TRUE add the evaluation times to the output.
return.object [logical] If TRUE return the object where the iid decomposition has been added. Otherwise return a list (see the return section)

Details

This function implements the first three formula (no number,10,11) of the subsection "Empirical estimates" in Ozenne et al. (2017). If there is no event in a strata, the influence function for the baseline hazard is set to 0.

Argument store.iid: If n denotes the sample size, J the number of jump times, and p the number of coefficients:

- store.iid="full" exports the influence function for the coefficients and the baseline hazard at each event time.
- `store.iid="minimal"` exports the influence function for the coefficients. For the baseline hazard it only computes the quantities necessary to compute the influence function in order to save memory.

More details can be found in appendix B of Ozenne et al. (2017).

Value

For Cox models, it returns the object with an additional iid slot (i.e. `object$ iid`). It is a list containing:

- `IFbetaInfluence` function for the regression coefficient.
- `IFhazardTime` differential of the influence function of the hazard.
- `IFcumhazardInfluence` function of the cumulative hazard.
- `calcIFhazardElements` used to compute the influence function at a given time.
- `timeTimes` at which the influence function has been evaluated.
- `etime.minTime` of first event (i.e. jump) in each strata.
- `etime.maxLast` last observation time (i.e. jump or censoring) in each strata.
- `indexObsIndex` of the observation in the original dataset.

For Cause-Specific Cox models, it returns the object with an additional iid slot for each model (e.g. `object$ models[[1]]iid`).

References


Examples

```r
library(survival)
library(data.table)
library(prodlim)
set.seed(10)
d <- sampleData(100, outcome = "survival")[,.(eventtime,event,X1,X6)]
setkey(d, eventtime)

m.cox <- coxph(Surv(eventtime, event) ~ X1+X6, data = d, y = TRUE, x = TRUE)

system.time(IF.cox <- iidCox(m.cox))

IF.cox.all <- iidCox(m.cox, tau.hazard = sort(unique(c(7,d$eventtime))))
IF.cox.beta <- iidCox(m.cox, baseline.iid = FALSE)
```
influenceTest  

**Influence test [Experimental!!]**

**Description**

Compare two estimates using their influence function

**Usage**

```r
influenceTest(object, ...)  
## S3 method for class 'list'
influenceTest(  
  object,  
  newdata,  
  times,  
  type,  
  cause,  
  keep.newdata = TRUE,  
  keep.strata = FALSE,  
  ...  
)  
## Default S3 method:
influenceTest(object, object2, band = TRUE, ...)
```

**Arguments**

- **object**: either a list of models or an object of class predictCox or predictCSC.
- **...**: additional arguments to be passed to lower level functions.
- **newdata**: [data.frame or data.table] Contain the values of the predictor variables defining subject specific predictions.
- **times**: [numeric vector] Time points at which to return the estimated absolute risk.
- **type**: [character] the type of predicted value.
- **cause**: [integer/character] Identifies the cause of interest among the competing events.
- **keep.newdata**: [logical] If TRUE add the value of the covariates used to make the prediction in the output.
- **keep.strata**: [logical] If TRUE add the value of the strata used to make the prediction in the output.
- **object2**: same as predict1 but for another model.
- **band**: [logical] If TRUE add the influence function to the output such that `confint` will be able to compute the confidence bands.
Examples

library(lava)
library(survival)
library(prodlim)
library(data.table)
n <- 100

#### Under H1
set.seed(1)
newdata <- data.frame(X1=0:1)

## simulate non proportional hazard using lava
m <- lvmlv()
regression(m) <- y ~ 1
regression(m) <- s ~ exp(-2*X1)
distribution(m,~X1) <- binomial.lvm()
distribution(m,~cens) <- coxWeibull.lvm(scale=1)
distribution(m,~y) <- coxWeibull.lvm(scale=1,shape=-s)
eventTime(m) <- eventtime ~ min(y=1,cens=0)
d <- as.data.table(sim(m,n))
setkey(d, eventtime)

## fit cox models
m.cox <- coxph(Surv(eventtime, status) ~ X1,
data = d, y = TRUE, x = TRUE)
mStrata.cox <- coxph(Surv(eventtime, status) ~ strata(X1),
data = d, y = TRUE, x = TRUE)

## compare models
# one time point
outIF <- influenceTest(list(m.cox, mStrata.cox),
    type = "survival", newdata = newdata, times = 0.5)
confint(outIF)

# several time points
outIF <- influenceTest(list(m.cox, mStrata.cox),
    type = "survival", newdata = newdata, times = c(0.5,1,1.5))
confint(outIF)

#### Under H0 (Cox) ####
set.seed(1)

## simulate proportional hazard using lava
m <- lvmlv()
regression(m) <- y ~ 1
distribution(m,~X1) <- binomial.lvm()
distribution(m,~cens) <- coxWeibull.lvm()
distribution(m,~y) <- coxWeibull.lvm()
eventTime(m) <- eventtime ~ min(y=1,cens=0)
d <- as.data.table(sim(m,n))
setkey(d, eventtime)
## fit cox models

```
Utime <- sort(unique(d$eventtime))
m.cox <- coxph(Surv(eventtime, status) ~ X1, 
data = d, y = TRUE, x = TRUE)

mStrata.cox <- coxph(Surv(eventtime, status) ~ strata(X1), 
data = d, y = TRUE, x = TRUE)
```

```
p.cox <- predictCox(m.cox, newdata = newdata, time = Utime, type = "survival")
p.coxStrata <- predictCox(mStrata.cox, newdata = newdata, time = Utime, type = "survival")
```

## display

```
library(ggplot2)
autoplot(p.cox)
autoplot(p.coxStrata)
```

## compare models

```
outIF <- influenceTest(list(m.cox, mStrata.cox), 
type = "survival", newdata = newdata, times = Utime[1:6])
confint(outIF)
```

#### Under H0 (CSC) ####

```
set.seed(1)
ff <- ~ f(X1,2) + f(X2,-0.033)
ff <- update(ff, ~ .+ f(X3,0) + f(X4,0) + f(X5,0))
ff <- update(ff, ~ .+ f(X6,0) + f(X7,0) + f(X8,0) + f(X9,0))
d <- sampleData(n, outcome = "competing.risk", formula = ff)
d[,X1:=as.numeric(as.character(X1))]
d[,X2:=as.numeric(as.character(X2))]
d[,X3:=as.numeric(as.character(X3))]
d[,X4:=as.numeric(as.character(X4))]
d[,X5:=as.numeric(as.character(X5))]
setkey(d, time)

Utime <- sort(unique(d$time))
```

## fit cox models

```
m.CSC <- CSC(Hist(time, event) ~ X1 + X2, data = d)
mStrata.CSC <- CSC(Hist(time, event) ~ strata(X1) + X2 + X3, data = d)
```

## display

```
library(ggplot2)
autoplot(p.cox)
autoplot(p.coxStrata)
```

## compare models

```
outIF <- influenceTest(list(m.CSC, mStrata.CSC), 
cause = 1, newdata = unique(d[,.(X1,X2,X3)]), times = Utime[1:5])
confint(outIF)
```
Description

Compute the information (i.e. opposite of the expectation of the second derivative of the log-likelihood) for IPCW logistic regressions.

Usage

```r
## S3 method for class 'wglm'
information(x, times = NULL, simplifies = TRUE, ...)
```

Arguments

- `x`: a `wglm` object.
- `times`: [numeric vector] time points at which the score should be output.
- `simplifies`: [logical] should the output be converted to a matrix when only one timepoint is requested. Otherwise will always return a list.
- `...`: Not used.

IPA

**Explained variation for settings with binary, survival and competing risk outcome**

Description

Index of Prediction Accuracy: General R^2 for binary outcome and right censored time to event (survival) outcome also with competing risks

Usage

```r
rsquared(object,...)
IPA(object,...)
```

```r
## Default S3 method:
rsquared(object,formula,newdata,times,cause,...)
## S3 method for class 'glm'
rsquared(object,formula,newdata,...)
## S3 method for class 'coxph'
rsquared(object,formula,newdata,times,...)
## S3 method for class 'CauseSpecificCox'
rsquared(object,formula,newdata,times,cause,...)
## Default S3 method:
IPA(object,formula,newdata,times,cause,...)
## S3 method for class 'glm'
IPA(object,formula,newdata,...)
## S3 method for class 'coxph'
IPA(object,formula,newdata,times,...)
## S3 method for class 'CauseSpecificCox'
IPA(object,formula,newdata,times,cause,...)
```
IPA

Arguments

- **object**: Model for which we want IPA.
- **...** passed to `riskRegression::Score`
- **newdata**: Optional validation data set in which to compute IPA
- **formula**: Formula passed to `Score`. If not provided, try to use the formula of the call of `object`, if any.
- **cause**: For competing risk models the event of interest
- **times**: Vector of time points used as prediction horizon for the computation of Brier scores.

Details

IPA ($R^2$) is calculated based on the model's predicted risks. The Brier score of the model is compared to the Brier score of the null model.

Value

Data frame with explained variation values for the full model.

Author(s)

Thomas A. Gerds <tag@biostat.ku.dk>

See Also

Score

Examples

```r
library(prodlim)
library(data.table)
# binary outcome
library(lava)
set.seed(18)
learndat <- sampleData(48, outcome="binary")
lr1 = glm(Y~X1+X2+X7+X9, data=learndat, family=binomial)
IPA(lr1)

## validation data
valdat=sampleData(94, outcome="binary")
IPA(lr1,newdata=valdat)

## predicted risks externally given
p1=predictRisk(lr1,newdata=valdat)
IPA(p1, formula=Y~1, valdat)

# survival
library(survival)
data(pbc)
```
setup code:

```r
pbc = na.omit(pbc)
pbctest = !(1:NROW(pbc)) %in% sample(1:NROW(pbc), size = .632 * NROW(pbc))
pbclearn = pbc[pbctest,]
cox1 = coxph(Surv(time, status != 0) ~ age + sex + log(bili) + log(albumin) + log(protime),
             data = pbclearn, x = TRUE)

## same data
IPA(cox1, formula = Surv(time, status != 0) - 1, times = 1000)

## validation data
pbcval = pbc[!pbctest,]
IPA(cox1, formula = Surv(time, status != 0) - 1, newdata = pbcval, times = 1000)

## predicted risks externally given
p2 = predictRisk(cox1, newdata = pbcval, times = 1000)
IPA(cox1, formula = Surv(time, status != 0) - 1, newdata = pbcval, times = 1000)

# competing risks
data(Melanoma)
Melanomatest = !(1:NROW(Melanoma)) %in% sample(1:NROW(Melanoma), size = .632 * NROW(Melanoma))
Melanomalearn = Melanoma[Melanomatest,]
fit1 <- CSC(list(Hist(time, status) ~ sex,
                Hist(time, status) ~ invasion + epicel + age),
            data = Melanoma)
IPA(fit1, times = 1000, cause = 2)

## validation data
Melanomaval = Melanoma[!Melanomatest,]
IPA(fit1, formula = Hist(time, status) - 1, newdata = Melanomaval, times = 1000)

## predicted risks externally given
p3 = predictRisk(fit1, cause = 1, newdata = Melanomaval, times = 1000)
IPA(p3, formula = Hist(time, status) - 1, cause = 1, newdata = Melanomaval, times = 1000)
```

---

**ipcw**

*Estimation of censoring probabilities*

**Description**

This function is used internally to obtain inverse of the probability of censoring weights.

**Usage**

```r
ipcw(formula, data, method, args, times,

```
```r
subject.times,  
lag = 1,  
what,  
keep = NULL
)

Arguments

formula A survival formula like, \( \text{Surv}(\text{time},\text{status}) \sim 1 \), where as usual status=0 means censored. The status variable is internally reversed for estimation of censoring rather than survival probabilities. Some of the available models (see argument model) will use predictors on the right hand side of the formula.
data The data used for fitting the censoring model
method Censoring model used for estimation of the (conditional) censoring distribution.
args A list of arguments which is passed to method
times For what="IPCW.times" a vector of times at which to compute the probabilities of not being censored.
subject.times For what="IPCW.subject.times" a vector of individual times at which the probabilities of not being censored are computed.
lag If equal to 1 then obtain \( G(T_i-X_i) \), if equal to 0 estimate the conditional censoring distribution at the subject.times, i.e. \( G(T_i|X_i) \).
what Decide about what to do: If equal to "IPCW.times" then weights are estimated at given times. If equal to "IPCW.subject.times" then weights are estimated at individual subject.times. If missing then produce both.
keep Which elements to add to the output. Any subset of the vector c("times","fit","call").

Details

Inverse of the probability of censoring weights (IPCW) usually refer to the probabilities of not being censored at certain time points. These probabilities are also the values of the conditional survival function of the censoring time given covariates. The function ipcw estimates the conditional survival function of the censoring times and derives the weights.

IMPORTANT: the data set should be ordered, order(time,-status) in order to get the values IPCW.subject.times in the right order for some choices of method.

Value

A list with elements depending on argument keep.
times The times at which weights are estimated
IPCW.times Estimated weights at times
IPCW.subject.times Estimated weights at individual time values subject.times
fit The fitted censoring model
method The method for modelling the censoring distribution
call The call
Author(s)

Thomas A. Gerds <tag@biostat.ku.dk>

Examples

library(prodlim)
library(rms)
dat=SimSurv(30)

dat <- dat[order(dat$time),]

# using the marginal Kaplan-Meier for the censoring times
WKMM=ipcw(Hist(time,status)~X2,
data=dat,
   method="marginal",
times=sort(unique(dat$time)),
   subject.times=dat$time,keep=c("fit"))
plot(WKMM$fit)

# using the Cox model for the censoring times given X2
library(survival)
WCox=ipcw(Hist(time=time,event=status)~X2,
data=dat,
   method="cox",
times=sort(unique(dat$time)),
   subject.times=dat$time,keep=c("fit"))
WCox$fit

plot(WCox$fit)
lines(sort(unique(dat$time)),
   1-WCox$IPCW.times[,1],
   type="l",
   col=2,
   lty=3,
   lwd=3)
lines(sort(unique(dat$time)),
   1-WCox$IPCW.times[,5],
   type="l",
   col=3,
   lty=3,
   lwd=3)

# using the stratified Kaplan-Meier
# for the censoring times given X2
WKMM2=ipcw(Hist(time,status)~X2,
data=dat,
   method="nonpar",
times=sort(unique(dat$time)),
   subject.times=dat$time,keep=c("fit"))
WKMM2$fit
Melanoma

73

plot(WKM2$fit, add=FALSE)

Melanoma

Malignant melanoma data

Description

In the period 1962-77, 205 patients with malignant melanoma (cancer of the skin) had a radical operation performed at Odense University Hospital, Denmark. All patients were followed until the end of 1977 by which time 134 were still alive while 71 had died (of out whom 57 had died from cancer and 14 from other causes).

Format

A data frame with 205 observations on the following 12 variables.

time  time in days from operation
status a numeric with values 0=censored 1=death.malignant.melanoma 2=death.other.causes
event a factor with levels censored death.malignant.melanoma death.other.causes
invasion a factor with levels level.0, level.1, level.2
ici inflammatory cell infiltration (IFI): 0, 1, 2 or 3
epicel a factor with levels not present present
ulcer a factor with levels not present present
thick tumour thickness (in 1/100 mm)
sex a factor with levels Female Male
age age at operation (years)
logthick tumour thickness on log-scale

Details

The object of the study was to assess the effect of risk factors on survival. Among such risk factors were the sex and age of the patients and the histological variables tumor thickness and ulceration (absent vs. present).

References

Regression with linear predictors (2010)
Andersen, P.K. and Skovgaard, L.T.
Springer Verlag

Examples

data(Melanoma)
model.matrix.cph  
**Extract design matrix for cph objects**

**Description**

Extract design matrix for cph objects

**Usage**

```r
## S3 method for class 'cph'
model.matrix(object, data, ...)
```

**Arguments**

- `object`: a cph object.
- `data`: a dataset.
- `...`: not used

model.matrix.phreg  
**Extract design matrix for phreg objects**

**Description**

Extract design matrix for phreg objects

**Usage**

```r
## S3 method for class 'phreg'
model.matrix(object, data, ...)
```

**Arguments**

- `object`: a phreg object.
- `data`: a dataset.
- `...`: not used

**Details**

mainly a copy paste of the beginning of the phreg function.
Description

PAQUID is a prospective cohort study initiated in 1988 in South Western France to explore functional and cerebral ageing. This sample includes n=2561 subjects. Data contains a time-to-event, a type of event and two cognitive scores measured at baseline.

Format

A data frame with 2561 observations on the following 4 variables.

- **time**  the time-to-event (in years).
- **status** the type of event 0 = censored, 1 = dementia onset and 2 = death without dementia.
- **DSST**  score at the Digit Symbol Substitution Score Test. This test explores attention and psychomotor speed.
- **MMSE**  score at the Mini Mental State Examination. This test is often used as an index of global cognitive performance.

Source

The data have been first made publicly available via the package timeROC.

References


Examples

```
data(Paquid)
```
penalizedS3  

S3-wrapper for S4 function penalized

Description

S3-wrapper for S4 function penalized

Usage

penalizedS3(formula, data, type = "elastic.net", lambda1, lambda2, fold, ...)

Arguments

- **formula**: Communicated outcome and explanatory variables. See examples.
- **data**: Data set in which formula is to be interpreted
- **type**: String specifying the type of penalization. Should match one of the following values: "ridge", "lasso", "elastic.net".
- **lambda1**: Lasso penalty
- **lambda2**: Ridge penalty
- **fold**: Passed to penalized::profL1
- ... Arguments passed to penalized

Examples

library(prodlim)

## Not run:
## too slow
if (require("penalized", quietly=TRUE)){
  library(penalized)
  set.seed(8)
  d <- sampleData(200, outcome="binary")
  newd <- sampleData(80, outcome="binary")
  fitridge <- penalizedS3(Y~X1+X2+pen(7:8), data=d, type="ridge",
                          standardize=TRUE, model="logistic", trace=FALSE)
  fitlasso <- penalizedS3(Y~X1+X2+pen(7:8), data=d, type="lasso",
                          standardize=TRUE, model="logistic", trace=FALSE)
  # fitnet <- penalizedS3(Y~X1+X2+pen(7:8), data=d, type="elastic.net",
  #                       standardize=TRUE, model="logistic", trace=FALSE)
  predictRisk(fitr ridge,newdata=newd)
  predictRisk(fitlasso,newdata=newd)
  # predictRisk(fitnet,newdata=newd)
  Score(list(fitr ridge),data=newd,formula=Y~1)
  # Score(list(fitr ridge),data=newd,formula=Y~1,split.method="bootcv",B=2)
  data(nki70)  ## S4 fit
  fitS4 <- penalized(Surv(time, event), penalized = nki70[,8:77],
                     unpenalized = ~ER+Age+Diam+N+Grade, data = nki70, lambda1 = 1)
fitS3 <- penalizedS3(Surv(time,event)~ER+Age+Diam+pen(8:77)+N+Grade, data=nki70, lambda1=1)
## or
penS3 <- penalizedS3(Surv(time,event)~ER+pen(TSPYL5,Contig63649_RC)+pen(10:77)+N+Grade, data=nki70, lambda1=1)
## also this works
penS3 <- penalizedS3(Surv(time,event)~ER+Age+pen(8:33)+Diam+pen(34:77)+N+Grade, data=nki70, lambda1=1)
}
## End(Not run)

---

**plot.riskRegression**  
*Plotting predicted risk*

**Description**

Show predicted risk obtained by a risk prediction model as a function of time.

**Usage**

```r
## S3 method for class 'riskRegression'
plot(x,  
cause,  
newdata,  
xlab,  
ylab,  
xlim,  
ylim,  
lwd,  
col,  
lty,  
axes=TRUE,  
percent=TRUE,  
legend=TRUE,  
add=FALSE,  
...)```

**Arguments**

- **x**: Fitted object obtained with one of `ARR`, `LRR`, `riskRegression`.
- **cause**: For `CauseSpecificCox` models the cause of interest.
- **newdata**: A data frame containing predictor variable combinations for which to compute predicted risk.
- **xlab**: See `plot`
- **ylab**: See `plot`
- **xlim**: See `plot`
plotAUC

A vector of line thicknesses for the regression coefficients.

col
A vector of colors for the regression coefficients.

lty
A vector of line types for the regression coefficients.

axes
Logical. If FALSE then do not draw axes.

percent
If true the y-axis is labeled in percent.

legend
If true draw a legend.

add
Logical. If TRUE then add lines to an existing plot.

... Used for transclusion of smart arguments for plot, lines, axis and background.

See function SmartControl from prodlim.

Author(s)

Thomas Alexander Gerds <tag@biostat.ku.dk>

Examples

library(survival)
library(prodlim)
data(Melanoma)
fit.arr <- ARR(Hist(time,status)~invasion+age+strata(sex),data=Melanoma,cause=1)
plot(fit.arr,xlim=c(500,3000))

plotAUC

Plot of time-dependent AUC curves

Description

Plot of time-dependent AUC curves

Usage

plotAUC(
  x,
  models,
  which = "score",
  xlim,
  ylim,
  xlab,
  ylab,
  col,
  lwd,
  lty = 1,
cex = 1,
pch = 1,
type = "l",
axes = TRUE,
percent = TRUE,
conf.int = FALSE,
legend = TRUE,
...
)

Arguments

- **x**: Object obtained with `Score.list`
- **models**: Choice of models to plot
- **which**: Character. Either "score" to show AUC or "contrasts" to show differences between AUC.
- **xlim**: Limits for x-axis
- **ylim**: Limits for y-axis
- **xlab**: Label for x-axis
- **ylab**: Label for y-axis
- **col**: line color
- **lwd**: line width
- **lty**: line style
- **cex**: point size
- **pch**: point style
- **type**: line type
- **axes**: Logical. If TRUE draw axes.
- **percent**: Logical. If TRUE scale y-axis in percent.
- **conf.int**: Logical. If TRUE draw confidence shadows.
- **legend**: Logical. If TRUE draw legend.
- **...**: Used for additional control of the subroutines: plot,

Examples

```r
set.seed(9)
library(survival)
library(prodlim)
set.seed(10)
d = sampleData(100, outcome="survival")
nd = sampleData(100, outcome="survival")
f1 = coxph(Surv(time, event) ~ X1 + X6 + X8, data=d, x=TRUE, y=TRUE)
f2 = coxph(Surv(time, event) ~ X2 + X5 + X9, data=d, x=TRUE, y=TRUE)
x = Score(list("X1+X6+X8"=f1, "X2+X5+X9"=f2), formula=Surv(time, event) ~ 1,
data=nd, metrics="auc", null.model=FALSE, times=seq(3:10))

aucgraph <- plotAUC(x)
```
plotBrier

Plot Brier curve

Description
Plot Brier score curves

Usage
plotBrier(
x, models, which = "score", xlim, ylim, xlab, ylab, col, lwd, lty = 1, cex = 1, pch = 1, type = "l", axes = 1L, percent = 1L, conf.int = 0L, legend = 1L, ...
)

Arguments
x Object obtained with Score
models Choice of models to plot
which Character. Either "score" to show AUC or "contrasts" to show differences between AUC.
xlim Limits for x-axis
ylim Limits for y-axis
xlab Label for x-axis
ylab Label for y-axis
plotCalibration

Description

Plot Calibration curves for risk prediction models

Usage

plotCalibration(x, models, times, method = "nne", cens.method, round = TRUE, bandwidth = NULL, q = 10, bars = FALSE, col, lwd, lty, cex, pch, type, axes, percent, conf.int, legend, ...)

Examples

# survival
library(survival)
library(prodlim)
set.seed(7)
ds1=sampleData(40,outcome="survival")
ds2=sampleData(40,outcome="survival")
f1 <- coxph(Surv(time,event)~X1+X3+X5+X7+X9,data=ds1,x=TRUE)
f2 <- coxph(Surv(time,event)~X2+X4+X6+X8+X10,data=ds1,x=TRUE)
xscore <- Score(list(f1,f2),formula=Hist(time,event)~1,data=ds2,times=0:12,metrics="brier")
plotBrier(xscore)
plotCalibration

```
plotCalibration (
  hanging = FALSE,
  names = "quantiles",
  pseudo = FALSE,
  rug,
  show.frequencies = FALSE,
  plot = TRUE,
  add = FALSE,
  diag = !add,
  legend = !add,
  auc.in.legend,
  brier.in.legend,
  axes = !add,
  xlim = c(0, 1),
  ylim = c(0, 1),
  xlab = ifelse(bars, "Risk groups", "Predicted risk"),
  ylab,
  col,
  lwd,
  lty,
  pch,
  type,
  percent = TRUE,
  na.action = na.fail,
  cex = 1,
  ...)
```

**Arguments**

- **x** Object obtained with function `Score` models Choice of models to plot times Time point specifying the prediction horizon. method The method for estimating the calibration curve(s):
  - "quantile" The observed proportion at predicted risk value ‘p’ is obtained in groups defined by quantiles of the predicted event probabilities of all subjects. The number of groups is controlled by argument `q`. `nne`: The observed proportion at predicted risk value ‘p’ is obtained based on the subjects whose predicted risk is inside a nearest neighborhood around the value ‘p’. The larger the bandwidth the more subjects are included in the current neighborhood.

cens.method For right censored data only. How observed proportions are calculated. Either "jackknife" or "local":
  - "jackknife" Compute a running mean of the jackknife pseudovalues across neighborhoods/groups of the predicted risks. Here we rely on the assumption that censoring is independent of the event time and the covariates, see References.
Compute the Kaplan-Meier estimator in absence of competing risks and the Aalen-Johansen estimator in presence of competing risks locally like a running mean in neighborhoods of the predicted risks. The widths of the neighborhoods are defined according to method.

- **round**: If TRUE predicted probabilities are rounded to two digits before smoothing. This may have a considerable effect on computing efficiency in large data sets.
- **bandwidth**: The bandwidth for method=\"nne\".
- **q**: The number of quantiles for method=\"quantile\" and bars=TRUE.
- **bars**: If TRUE, use barplots to show calibration.
- **hanging**: Barplots only. If TRUE, hang bars corresponding to observed frequencies (estimated actual risk) at the value of the corresponding prediction.
- **names**: Barplots only. Names argument passed to names.arg of barplot.
- **pseudo**: If TRUE show pseudo values (only for right censored data).
- **rug**: If TRUE show rug plot at the predictions.
- **show.frequencies**: Barplots only. If TRUE, show frequencies above the bars.
- **plot**: If FALSE, do not plot the results, just return a plottable object.
- **add**: If TRUE the line(s) are added to an existing plot.
- **diag**: If FALSE no diagonal line is drawn.
- **legend**: Logical. If TRUE draw legend.
- **auc.in.legend**: Logical. If TRUE add AUC to legend.
- **brier.in.legend**: Logical. If TRUE add Brier score to legend.
- **axes**: If FALSE no axes are drawn.
- **xlim**: Limits of x-axis.
- **ylim**: Limits of y-axis.
- **xlab**: Label for x-axis.
- **ylab**: Label for y-axis.
- **col**: Vector with colors, one for each element of object. Passed to *lines*.
- **lwd**: Vector with line widths, one for each element of object. Passed to *lines*.
- **lty**: Vector with line style, one for each element of object. Passed to *lines*.
- **pch**: Passed to *lines*.
- **type**: Passed to *lines*.
- **percent**: If TRUE axes labels are multiplied by 100 and thus interpretable on a percent scale.
- **na.action**: what to do with NA values. Passed to *model.frame*
- **cex**: Default cex used for legend and labels.
- **...**: Used to control the subroutines: plot, axis, lines, barplot, legend, addtable2plot, points (pseudo values), rug. See *SmartControl*. 

See SmartControl.
Details

In uncensored data, the observed frequency of the outcome event is calculated locally at the predicted risk. In right censored data with and without competing risks, the actual risk is calculated using the Kaplan-Meier and the Aalen-Johansen method, respectively, locally at the predicted risk.

Examples

```
library(prodlim)
# binary
set.seed(10)
db=sampleData(100, outcome="binary")
fb1=glm(Y~X1+X5+X7, data=db, family="binomial")
fb2=glm(Y~X1+X3+X6+X7, data=db, family="binomial")
xb=Score(list(model1=fb1, model2=fb2), Y~1, data=db, plots="cal")
plotCalibration(xb, brier.in.legend=TRUE)
plotCalibration(xb, bars=TRUE, model="model1")
plotCalibration(xb, models=1, bars=TRUE, names.cex=1.3)

# survival
library(survival)
library(prodlim)
dslearn=sampleData(56, outcome="survival")
dtest=sampleData(100, outcome="survival")
fs1=coxph(Surv(time,event)-X1+X5+X7, data=dslern, x=1)
fs2=coxph(Surv(time,event)-strata(X1)+X3+X6+X7, data=dslern, x=1)
xs=Score(list(Cox1=fs1, Cox2=fs2), Surv(time,event)-1, data=dtest, plots="cal", metrics=NULL)
plotCalibration(xs)
plotCalibration(xs, cens.method="local", pseudo=1)
plotCalibration(xs, method="quantile")

# competing risks

## Not run:
data(Melanoma)
f1 <- CSC(Hist(time,status)-age+sex+epicel+ulcer, data=Melanoma)
f2 <- CSC(Hist(time,status)-age+sex+logthick+epicel+ulcer, data=Melanoma)
x <- Score(list(model1=f1, model2=f2), Hist(time,status)-1, data=Melanoma, cause= 2, times=5*365.25, plots="cal")
plotCalibration(x)
## End(Not run)
```

plotEffects  
Plotting time-varying effects from a risk regression model.
Description

Plot time-varying effects from a risk regression model.

Usage

plotEffects(
  x, formula, level, ref.line = TRUE, conf.int = 0.95, xlim, ylim, xlab = "Time", ylab = "Cumulative coefficient", col, lty, lwd, add = FALSE, legend, axes = TRUE,
...
)

Arguments

x Fitted object obtained with one of ARR, LRR, riskRegression.
formula A formula to specify the variable(s) whose regression coefficients should be plotted.
level For categorical variables the level (group) whose contrast to the reference level (group) should be plotted.
ref.line Logical. If TRUE then add a horizontal line at zero.
conf.int Logical. If TRUE then add confidence limits. Can be controlled using smart arguments. See examples
xlim See plot
ylim See plot
xlab See plot
ylab See plot
col A vector of colors for the regression coefficients.
lty A vector of line types for the regression coefficients.
lwd A vector of line thicknesses for the regression coefficients.
add Logical. If TRUE then add lines to an existing plot.
legend Logical. If TRUE then add a legend. Can be controlled using smart arguments. See examples.
plotPredictRisk

Plotting predicted risks curves.

axes Logical. If FALSE then do not draw axes.

... Used for transclusion of smart arguments for plot, axis. See function SmartControl from prodlim.

Author(s)

Thomas H. Scheike <ts@biostat.ku.dk>

Thomas A. Gerds <tag@biostat.ku.dk>

Examples

library(survival)
library(prodlim)
data(Melanoma)

fit.tarr <- ARR(Hist(time,status)~strata(sex),
data=Melanoma,
cause=1)
plotEffects(fit.tarr)

fit.tarr <- ARR(Hist(time,status)~strata(sex)+strata(invasion),
data=Melanoma,
cause=1,
times=seq(800,3000,20))
plotEffects(fit.tarr,formula=~sex)
plotEffects(fit.tarr,formula=~invasion)
plotEffects(fit.tarr,
  formula=~invasion,
  level="invasionlevel.1")

## legend arguments are transcluded:
plotEffects(fit.tarr,
  formula=~invasion,
  legend.bty="b",
  legend.cex=1)

## and other smart arguments too:
plotEffects(fit.tarr,
  formula=~invasion,
  legend.bty="b",
  axis2.las=2,
  legend.cex=1)
Description

Time-dependent event risk predictions.

Usage

plotPredictRisk(
  x,
  newdata,
  times,
  cause = 1,
  xlim,
  ylim,
  xlab,
  ylab,
  axes = TRUE,
  col,
  density,
  lty,
  lwd,
  add = FALSE,
  legend = TRUE,
  percent = FALSE,
  ...
)

Arguments

  x          Object specifying an event risk prediction model.
  newdata    A data frame with the same variable names as those that were used to fit the
             model x.
  times      Vector of times at which to return the estimated probabilities.
  cause      Show predicted risk of events of this cause
  xlim       Plotting range on the x-axis.
  ylim       Plotting range on the y-axis.
  xlab       Label given to the x-axis.
  ylab       Label given to the y-axis.
  axes       Logical. If FALSE no axes are drawn.
  col        Vector of colors given to the survival curve.
  density    Density of the color – useful for showing many (overlapping) curves.
  lty        Vector of lty’s given to the survival curve.
  lwd        Vector of lwd’s given to the survival curve.
  add        Logical. If TRUE only lines are added to an existing device
  legend     Logical. If TRUE a legend is plotted by calling the function legend. Optional
             arguments of the function legend can be given in the form legend.x=val where
             x is the name of the argument and val the desired value. See also Details.
percent  Logical. If TRUE the y-axis is labeled in percent.

... Parameters that are filtered by SmartControl and then passed to the functions: plot, axis, legend.

Details
Arguments for the invoked functions legend and axis can be specified as legend.lty=2. The specification is not case sensitive, thus Legend.lty=2 or LEGEND.lty=2 will have the same effect. The function axis is called twice, and arguments of the form axis1.labels, axis1.at are used for the time axis whereas axis2.pos, axis1.labels, etc., are used for the y-axis. These arguments are processed via ...() of plotPredictRisk and inside by using the function SmartControl.

Value
The (invisible) object.

Author(s)
Ulla B. Mogensen and Thomas A. Gerds <tag@biostat.ku.dk>

References

See Also
plotRisk

Examples

library(survival)
# generate survival data
# no effect
set.seed(8)
d <- sampleData(80, outcome="survival", formula = ~f(X6, 0) + f(X7, 0))
d[,table(event)]
f <- coxph(Surv(time,event)~X6+X7,data=d,x=1)
plotPredictRisk(f)

# large effect
set.seed(8)
d <- sampleData(80, outcome="survival", formula = ~f(X6, 0.1) + f(X7, -0.1))
d[,table(event)]
f <- coxph(Surv(time,event)~X6+X7,data=d,x=1)
plotPredictRisk(f)

# generate competing risk data
# small effect
plotRisk

```
set.seed(8)
d <- sampleData(40, formula = ~f(X6, 0.01) + f(X7, -0.01))
d[, table(event)]
f <- CSC(Hist(time, event) ~ X5 + X6, data = d)
plotPredictRisk(f)

# large effect
set.seed(8)
d <- sampleData(40, formula = ~f(X6, 0.1) + f(X7, -0.1))
d[, table(event)]
f <- CSC(Hist(time, event) ~ X5 + X6, data = d)
plotPredictRisk(f)
```

---

**plotRisk**

*plot predicted risks*

**Description**

`plotRisk` is a function that plots predicted risks.

**Usage**

```r
plotRisk(
  x,  # Object obtained with function Score
  models,  # Choice of two models to plot. The predicted risks of
            # the first (second) are shown along the x-axis (y-axis).
  times,  # Time point specifying the prediction horizon.
  xlim = c(0, 1),  # x-axis limits
  ylim = c(0, 1),  # y-axis limits
  xlab,  # x-axis labels
  ylab,  # y-axis labels
  col,  # color
  pch,  # point character
  cex = 1,  # character expansion
  preclipse = 0,  # preclip level
  preclipse.shade = FALSE,  # preclip shade
  ...)
```

**Arguments**

- **x**: Object obtained with function `Score`
- **models**: Choice of two models to plot. The predicted risks of the first (second) are shown along the x-axis (y-axis).
- **times**: Time point specifying the prediction horizon.
- **xlim**: x-axis limits
- **ylim**: y-axis limits
- **xlab**: x-axis labels
plotRisk

- **ylab**: y-axis labels
- **col**: Colors used according to the outcome. Binary outcome (two colors: no event, event), survival outcome (three colors: censored, event, no event) competing risk outcome (4 or more colors: event, competing risk 1, ..., competing risk k, censored, no event)
- **pch**: Symbols used according to the outcome. Binary outcome (two symbols: no event, event), survival outcome (three symbols: censored, event, no event) competing risk outcome (4 or more symbols: event, competing risk 1, ..., competing risk k, censored, no event)
- **cex**: Point size
- **preclipse**: Value between 0 and 1 defining the preclipse area
- **preclipse.shade**: Logical. If TRUE shade the area of clinically meaningful change.

Details

Two rival prediction models are applied to the same data.

Value

A nice graph

Author(s)

Thomas A. Gerds <tag@biostat.ku.dk>

Examples

```r
library(prodlim)
## uncensored
set.seed(10)
learndat = sampleData(40,outcome="binary")
testdat = sampleData(40,outcome="binary")
lr1 = glm(Y~X1+X2+X7+X9,data=learndat,family="binomial")
lr2 = glm(Y~X3+X5+X6,data=learndat,family="binomial")
xb = Score(list("LR(X1+X2+X7+X9)"=lr1,"LR(X3+X5+X6)"=lr2),formula=Y~1,
data=testdat,summary="risks",null.model=0L)
plotRisk(xb)
## survival
library(survival)
set.seed(10)
learndat = sampleData(40,outcome="survival")
testdat = sampleData(40,outcome="survival")
cox1 = coxph(Surv(time,event)~X1+X2+X7+X9,data=learndat,x=TRUE)
cox2 = coxph(Surv(time,event)~X3+X5+X6,data=learndat,x=TRUE)
x = Score(list("Cox(X1+X2+X7+X9)"=cox1,"Cox(X3+X5+X6)"=cox2),formula=Surv(time,event)-1,
data=testdat,summary="risks",null.model=0L,times=c(3,5,6))
plotRisk(xs,times=5)
```
## competing risk

Not run:
```
library(prodlim)
library(survival)
set.seed(8)
learndat = sampleData(80,outcome="competing.risk")
testdat = sampleData(140,outcome="competing.risk")
m1 = FGR(Hist(time,event)~X2+X7+X9,data=learndat,cause=1)
m2 = CSC(Hist(time,event)~X2+X7+X9,data=learndat,cause=1)
xcr=Score(list("FGR"=m1,"CSC"=m2),formula=Hist(time,event)~1,
data=testdat,summary="risks",null.model=0L,times=c(3,5))
plotRisk(xcr,times=3)
```

## End(Not run)

---

### plotROC

#### Plot ROC curves

**Description**

Plot ROC curve

**Usage**

```
plotROC(
  x, 
  models, 
  times, 
  xlab = "1-Specificity", 
  ylab = "Sensitivity", 
  col, 
  lwd, 
  lty = 1, 
  cex = 1, 
  pch = 1, 
  legend = !add, 
  auc.in.legend = TRUE, 
  brier.in.legend = FALSE, 
  add = FALSE, 
  ...
)
```

**Arguments**

- **x** Object obtained with function Score
- **models** Choice of models to plot
- **times** Time point(s) specifying the prediction horizon
plotROC

xlab Label for x-axis
ylab Label for y-axis
col line color
lwd line width
lty line style
cex point size
pch point style
legend logical. If 1L draw a legend with the values of AUC.
auc.in.legend Logical. If TRUE add AUC to legend.
brier.in.legend Logical. If TRUE add Brier score to legend.
add logical. If 1L add lines to an existing plot.

... Used for additional control of the subroutines: plot, axis, lines, legend, addtable2plot. See SmartControl.

Examples

## binary
set.seed(18)
if (require("randomForest",quietly=TRUE)){
library(randomForest)
library(prodlim)
bd1 <- sampleData(40,outcome="binary")
bd2 <- sampleData(58,outcome="binary")
bd1[,y:=factor(Y)]
bd2[,y:=factor(Y)]
fb1 <- glm(y~X1+X2+X3+X4+X5+X6+X7+X8+X9+X10,data=bd1,family="binomial")
fb2 <- randomForest(y~X1+X2+X3+X4+X5+X6+X7+X8+X9+X10,data=bd1)
xb <- Score(list("glm"=fb1,"rf"=fb2),y~1,data=bd2,
plots="roc",metrics=c("auc","brier"))
plotROC(xb,brier.in.legend=1L)

# with cross-validation
## Not run:
xb3 <- Score(list("glm"=fb1,"rf"=fb2),y~1,data=bd1,
plots="roc",B=3,split.method="bootcv",
metrics=c("auc"))

## End(Not run)
}

## survival
set.seed(18)
library(survival)
sdl <- sampleData(40,outcome="survival")
stdt <- sampleData(58,outcome="survival")
fs1 <- coxph(Surv(time,event)~X3+X5+X6+X7+X8+X10,data=sdl,x=TRUE)
fs2 <- coxph(Surv(time,event)~X1+X2+X9,data=sdl,x=TRUE)
x <- Score(list(model1=fs1,model2=fs2),Hist(time,event)~1,data=sdt,
predict.CauseSpecificCox

Predicting Absolute Risk from Cause-Specific Cox Models

Description
Apply formula to combine two or more Cox models into absolute risk (cumulative incidence function).

Usage

## S3 method for class 'CauseSpecificCox'
predict(
  object,
  newdata,
  times,
  cause,
  type = "absRisk",
  landmark = NA,
  keep.times = 1L,
  keep.newdata = 1L,
  keep.strata = 1L,
  se = FALSE,
  band = FALSE,
  iid = FALSE,
  confint = (se + band) > 0,
  average.iid = FALSE,
  product.limit = TRUE,
  store.iid = "full",
  diag = FALSE,
  max.time = NULL,
  ...
)

Arguments

object The fitted cause specific Cox model
predict.CauseSpecificCox

newdata [data.frame or data.table] Contain the values of the predictor variables defining subject specific predictions relative to each cause. Should have the same structure as the data set used to fit the object.

times [numeric vector] Time points at which to return the estimated absolute risk.

cause [integer/character] Identifies the cause of interest among the competing events.

type [character] Can be changed to "survival" if the event free survival should be output instead of the absolute risk.

landmark [integer] The starting time for the computation of the cumulative risk.

keep.times [logical] If TRUE add the evaluation times to the output.

keep.newdata [logical] If TRUE add the value of the covariates used to make the prediction in the output list.

keep.strata [logical] If TRUE add the value of the strata used to make the prediction in the output list.

se [logical] If TRUE compute and add the standard errors to the output.

band [logical] If TRUE compute and add the quantiles for the confidence bands to the output.

iid [logical] If TRUE compute and add the influence function to the output.

confint [logical] If TRUE compute and add the confidence intervals/bands to the output. They are computed applying the confint function to the output.

average.iid [logical]. If TRUE add the average of the influence function over newdata to the output.

product.limit [logical]. If TRUE the survival is computed using the product limit estimator. Otherwise the exponential approximation is used (i.e. \( \exp(-\text{cumulative hazard}) \)).

store.iid [character] Implementation used to estimate the influence function and the standard error. Can be "full" or "minimal".

diag [logical] when FALSE the absolute risk/survival for all observations at all times is computed, otherwise it is only computed for the i-th observation at the i-th time.

max.time [numeric] maximum time of the response of the fitted data. Only relevant if model response element has been removed

... not used.

Details

This function computes the absolute risk as given by formula 2 of (Ozenne et al., 2017). Confidence intervals and confidence bands can be computed using a first order von Mises expansion. See the section "Construction of the confidence intervals" in (Ozenne et al., 2017).

A detailed explanation about the meaning of the argument store.iid can be found in (Ozenne et al., 2017) Appendix B "Saving the influence functions".

Note: for Cox regression models with time varying covariates it does not make sense to use this function, because the predicted risk has to be a measurable function of the data available at the time origin.

The iid decomposition is output using an array containing the value of the influence of each subject used to fit the object (dim 1), for each subject in newdata (dim 3), and each time (dim 2).
Author(s)
Brice Ozenne broz@sund.ku.dk, Thomas A. Gerds tag@biostat.ku.dk

References

See Also
confint.predictCSC to compute confidence intervals/bands. autoplot.predictCSC to display the predictions.

Examples
library(survival)
library(prodlim)

### generate data ###
set.seed(5)
d <- sampleData(80,outcome="comp") ## training dataset
nd <- sampleData(4,outcome="comp") ## validation dataset
d$time <- round(d$time,1) ## create tied events
ttt <- sort(sample(x = unique(d$time), size = 10))

## estimate a CSC model based on the coxph function
CSC.fit <- CSC(Hist(time,event)~ X3+X8, data=d, method = "breslow")

## compute the absolute risk of cause 1, in the validation dataset
## at time 1:10
CSC.risk <- predict(CSC.fit, newdata=nd, times=1:10, cause=1)

## other example
library(survival)
CSC.fit.s <- CSC(list(Hist(time,event)~ strata(X1)+X2+X9,
Hist(time,event)~ X2+strata(X4)+X8+X7),data=d, method = "breslow")
predict(CSC.fit.s,cause=1, times=ttt, se=1L) ## note: absRisk>1 due to small number of observations

## using the cph function instead of coxph
CSC.cph <- CSC(Hist(time,event)~ X1+X2,data=d, method = "breslow", fitter = "cph")
predict(CSC.cph, newdata = d, cause = 2, times = ttt)

## landmark analysis
T0 <- 1
predCSC.afterT0 <- predict(CSC.fit, newdata = d, cause = 2, times = ttt[ttt>T0], landmark = T0)
predCSC.afterT0
predict.FGR  
Predict subject specific risks (cumulative incidence) based on Fine-Gray regression model

Description

Predict subject specific risks (cumulative incidence) based on Fine-Gray regression model

Usage

```r
## S3 method for class 'FGR'
predict(object, newdata, times, ...)
```

Arguments

- `object`: Result of call to `FGR`
- `newdata`: Predictor values of subjects for who to predict risks
- `times`: Time points at which to evaluate the risks
- `...`: passed to `predict.crr`

Examples

```r
library(prodlim)
library(survival)
set.seed(10)
d <- sampleData(101, outcome = "competing.risk")
tFun<-function(t) {t}
fgr<-FGR(Hist(time, event)~X1+strata(X2)+X6+cov2(X7, tf=tFun),
data=d, cause=1)
predictRisk(fgr, times=5, newdata=d[1:10])
```

predict.riskRegression

*Predict individual risk.*

Description

Extract predictions from a risk prediction model.

Usage

```r
## S3 method for class 'riskRegression'
predict(object, newdata, ...)
```
predictCox

Arguments

- **object**: Fitted object obtained with one of ARR, LRR, riskRegression.
- **newdata**: A data frame containing predictor variable combinations for which to compute predicted risk.
- **...**: not used

Author(s)

Thomas H. Scheike <ts@biostat.ku.dk>
Thomas A. Gerds <tag@biostat.ku.dk>

References


Examples

```r
data(Melanoma)
library(prodlim)
library(survival)

fit.tarr <- ARR(Hist(time,status)~age+invasion+strata(sex),data=Melanoma,cause=1)
predict(fit.tarr,newdata=data.frame(age=48,
    invasion=factor("level.1",
      levels=levels(Melanoma$invasion)),
    sex=factor("Female",levels=levels(Melanoma$sex))))
predict(fit.tarr,newdata=data.frame(age=48,
    invasion=factor("level.1",
      levels=levels(Melanoma$invasion)),
    sex=factor("Male",levels=levels(Melanoma$sex))))
predict(fit.tarr,newdata=data.frame(age=c(48,58,68),
    invasion=factor("level.1",
      levels=levels(Melanoma$invasion)),
    sex=factor("Male",levels=levels(Melanoma$sex))))
predict(fit.tarr,newdata=Melanoma[1:4,])
```

predictCox

Fast computation of survival probabilities, hazards and cumulative hazards from Cox regression models

Description

Fast routine to get baseline hazards and subject specific hazards as well as survival probabilities from a survival::coxph or rms::cph object
Usage

predictCox(
  object, 
  times, 
  newdata = NULL, 
  centered = TRUE, 
  type = c("cumhazard", "survival"), 
  keep.strata = TRUE, 
  keep.times = TRUE, 
  keep.newdata = FALSE, 
  keep.infoVar = FALSE, 
  se = FALSE, 
  band = FALSE, 
  iid = FALSE, 
  confint = (se + band) > 0, 
  diag = FALSE, 
  average.iid = FALSE, 
  store.iid = "full"
)

Arguments

object [The fitted Cox regression model object either obtained with coxph (survival package) or cph (rms package).]

times [numeric vector] Time points at which to return the estimated hazard/cumulative hazard/survival.

newdata [data.frame or data.table] Contain the values of the predictor variables defining subject specific predictions. Should have the same structure as the data set used to fit the object.

centered [logical] If TRUE return prediction at the mean values of the covariates fit$mean, if FALSE return a prediction for all covariates equal to zero in the linear predictor. Will be ignored if argument newdata is used. For internal use.

type [character vector] the type of predicted value. Choices are

- "hazard" the baseline hazard function when argument newdata is not used and the hazard function when argument newdata is used.
- "cumhazard" the cumulative baseline hazard function when argument newdata is not used and the cumulative hazard function when argument newdata is used.
- "survival" the survival baseline hazard function when argument newdata is not used and the cumulative hazard function when argument newdata is used.

Several choices can be combined in a vector of strings that match (no matter the case) strings "hazard","cumhazard", "survival".

keep.strata [logical] If TRUE add the (newdata) strata to the output. Only if there any.

keep.times [logical] If TRUE add the evaluation times to the output.
predictCox

keep.newdata [logical] If TRUE add the value of the covariates used to make the prediction in the output list.
keep.infoVar [logical] For internal use.
se [logical] If TRUE compute and add the standard errors to the output.
band [logical] If TRUE compute and add the quantiles for the confidence bands to the output.
iid [logical] If TRUE compute and add the influence function to the output.
confint [logical] If TRUE compute and add the confidence intervals/bands to the output. They are computed applying the confint function to the output.
diag [logical] when FALSE the hazard/cumulative hazard/survival for all observations at all times is computed, otherwise it is only computed for the i-th observation at the i-th time.
average.iid [logical] If TRUE add the average of the influence function over newdata to the output.
store.iid [character] Implementation used to estimate the influence function and the standard error. Can be "full" or "minimal".

Details

When the argument newdata is not specified, the function computes the baseline hazard estimate. See (Ozenne et al., 2017) section "Handling of tied event times".

Otherwise the function computes survival probabilities with confidence intervals/bands. See (Ozenne et al., 2017) section "Confidence intervals and confidence bands for survival probabilities". The survival is computed using the exponential approximation (equation 3).

A detailed explanation about the meaning of the argument store.iid can be found in (Ozenne et al., 2017) Appendix B "Saving the influence functions".

The function is not compatible with time varying predictor variables.

The centered argument enables us to reproduce the results obtained with the basehaz function from the survival package but should not be modified by the user.

The iid decomposition is output using an array containing the value of the influence of each subject used to fit the object (dim 1), for each subject in newdata (dim 3), and each time (dim 2).

Author(s)

Brice Ozenne broz@sund.ku.dk, Thomas A. Gerds tag@biostat.ku.dk

References


See Also

confint.predictCox to compute confidence intervals/bands. autoplot.predictCox to display the predictions.
### Examples

```
library(survival)
library(data.table)

#### generate data ####
set.seed(10)
d <- sampleData(40,outcome="survival")  # training dataset
nd <- sampleData(4,outcome="survival")  # validation dataset
d$time <- round(d$time,1)  # create tied events
# table(duplicated(d$time))

#### stratified Cox model ####
fit <- coxph(Surv(time,event)~X1 + strata(X2) + X6,
data=d, ties="breslow", x = TRUE, y = TRUE)

## compute the baseline cumulative hazard
fit.haz <- predictCox(fit)
cbind(survival::basehaz(fit), fit.haz$cumhazard)

## compute individual specific cumulative hazard and survival probabilities
fit.pred <- predictCox(fit, newdata=nd, times=c(3,8), se = TRUE, band = TRUE)
fit.pred

#### other examples ####
# one strata variable
fitS <- coxph(Surv(time,event)~strata(X1)+X2,
data=d, ties="breslow", x = TRUE, y = TRUE)
predictCox(fitS)
predictCox(fitS, newdata=nd, times = 1)

# two strata variables
set.seed(1)
d$U=sample(letters[1:5],replace=TRUE,size=NROW(d))
d$V=sample(letters[4:10],replace=TRUE,size=NROW(d))
nd$U=sample(letters[1:5],replace=TRUE,size=NROW(nd))
nd$V=sample(letters[4:10],replace=TRUE,size=NROW(nd))
fit2S <- coxph(Surv(time,event)~X1+strata(U)+strata(V)+X2,
data=d, ties="breslow", x = TRUE, y = TRUE)
cbind(survival::basehaz(fit2S),predictCox(fit2S,type="cumhazard")$cumhazard)
predictCox(fit2S)
predictCox(fitS, newdata=nd, times = 3)

# left truncation
test2 <- list(start=c(1,2,5,2,1,7,3,4,8,8),
stop=c(2,3,6,7,8,9,9,9,14,17),
event=c(1,1,1,1,1,1,0,0,0,0),
x=c(1,0,0,1,0,1,1,1,0,0))
m.cph <- coxph(Surv(start, stop, event) ~ 1, test2, x = TRUE)
as.data.table(predictCox(m.cph))
```
predictCoxPL

Computation of survival probabilities from Cox regression models using the product limit estimator.

Description

Same as predictCox except that the survival is estimated using the product limit estimator.

Usage

predictCoxPL(
  object,
  times,
  newdata = NULL,
  type = c("cumhazard", "survival"),
  keep.strata = TRUE,
  keep.infoVar = FALSE,
  ...)

Arguments

object The fitted Cox regression model object either obtained with coxph (survival package) or cph (rms package).
times [numeric vector] Time points at which to return the estimated hazard/cumulative hazard/survival.
newdata [data.frame or data.table] Contain the values of the predictor variables defining subject specific predictions. Should have the same structure as the data set used to fit the object.
type [character vector] the type of predicted value. Choices are
  - "hazard" the baseline hazard function when argument newdata is not used and the hazard function when argument newdata is used.
  - "cumhazard" the cumulative baseline hazard function when argument newdata is not used and the cumulative hazard function when argument newdata is used.
  - "survival" the survival baseline hazard function when argument newdata is not used and the cumulative hazard function when argument newdata is used.

Several choices can be combined in a vector of strings that match (no matter the case) strings "hazard", "cumhazard", "survival".

keep.strata [logical] If TRUE add the (newdata) strata to the output. Only if there any.
keep.infoVar [logical] For internal use.
... additional arguments to be passed to predictCox.
Details

Note: the iid and standard errors are computed using the exponential approximation.

Examples

library(survival)

#### generate data ####
set.seed(10)
d <- sampleData(40,outcome="survival")
nd <- sampleData(4,outcome="survival")
d$time <- round(d$time,1)

#### Cox model ####
fit <- coxph(Surv(time,event)~ X1 + X2 + X6,
              data=d, ties="breslow", x = TRUE, y = TRUE)

## exponential approximation
predictCox(fit, newdata = d, times = 1:5)

## product limit
predictCoxPL(fit, newdata = d, times = 1:5)

#### stratified Cox model ####
fitS <- coxph(Surv(time,event)~ X1 + strata(X2) + X6,
              data=d, ties="breslow", x = TRUE, y = TRUE)

## exponential approximation
predictCox(fitS, newdata = d, times = 1:5)

## product limit
predictCoxPL(fitS, newdata = d, times = 1:5)

#### fully stratified Cox model ####
fitS <- coxph(Surv(time,event)~ 1,
              data=d, ties="breslow", x = TRUE, y = TRUE)

## product limit
GS <- survfit(Surv(time,event)~1, data = d)
range(predictCoxPL(fitS)$survival - GS$surv)

fitS <- coxph(Surv(time,event)~ strata(X2),
              data=d, ties="breslow", x = TRUE, y = TRUE)

## product limit
GS <- survfit(Surv(time,event)~X2, data = d)
range(predictCoxPL(fitS)$survival - GS$surv)
predictRisk

Extrating predicting risks from regression models

Description

Extract event probabilities from fitted regression models and machine learning objects. The function predictRisk is a generic function, meaning that it invokes specifically designed functions depending on the 'class' of the first argument. See predictRisk.

Usage

predictRisk(object, newdata, ...)  
## Default S3 method:  
predictRisk(object, newdata, times, cause, ...)

## S3 method for class 'double'  
predictRisk(object, newdata, times, cause, ...)

## S3 method for class 'integer'  
predictRisk(object, newdata, times, cause, ...)

## S3 method for class 'factor'  
predictRisk(object, newdata, times, cause, ...)

## S3 method for class 'numeric'  
predictRisk(object, newdata, times, cause, ...)

## S3 method for class 'glm'  
predictRisk(object, newdata, iid = FALSE, average.iid = FALSE, ...)

## S3 method for class 'multinom'  
predictRisk(object, newdata, iid = FALSE, average.iid = FALSE, cause = NULL, ...)

## S3 method for class 'formula'  
predictRisk(object, newdata, ...)

## S3 method for class 'BinaryTree'  
predictRisk(object, newdata, ...)
predictRisk(object, newdata, ...)
predictRisk

diag = FALSE,
iid = FALSE,
average.iid = FALSE,
...

## S3 method for class 'selectCox'
predictRisk(object, newdata, times, ...)

## S3 method for class 'prodlim'
predictRisk(object, newdata, times, cause, ...)

## S3 method for class 'survfit'
predictRisk(object, newdata, times, ...)

## S3 method for class 'psm'
predictRisk(object, newdata, times, ...)

## S3 method for class 'ranger'
predictRisk(object, newdata, times, cause, ...)

## S3 method for class 'rfsrc'
predictRisk(object, newdata, times, cause, ...)

## S3 method for class 'FGR'
predictRisk(object, newdata, times, cause, ...)

## S3 method for class 'riskRegression'
predictRisk(object, newdata, times, cause, ...)

## S3 method for class 'ARR'
predictRisk(object, newdata, times, cause, ...)

## S3 method for class 'CauseSpecificCox'
predictRisk(
  object,
  newdata,
  times,
  cause,
  product.limit = TRUE,
  diag = FALSE,
  iid = FALSE,
  average.iid = FALSE,
  truncate = FALSE,
  ...
)

## S3 method for class 'penfitS3'
predictRisk(object, newdata, times, ...)  

## S3 method for class 'SuperPredictor'  
predictRisk(object, newdata, ...)  

## S3 method for class 'gbm'  
predictRisk(object, newdata, times, ...)  

## S3 method for class 'flexsurvreg'  
predictRisk(object, newdata, times, ...)  

## S3 method for class 'Hal9001'  
predictRisk(object, newdata, times, cause, ...)  

## S3 method for class 'GLMnet'  
predictRisk(object, newdata, times = NA, ...)  

## S3 method for class 'singleEventCB'  
predictRisk(object, newdata, times, cause, ...)  

## S3 method for class 'CoxConfidential'  
predictRisk(object, newdata, ...)  

## S3 method for class 'wglm'  
predictRisk(  
  object,  
  newdata,  
  times = NULL,  
  product.limit = FALSE,  
  diag = FALSE,  
  iid = FALSE,  
  average.iid = FALSE,  
  ...  
)

Arguments

object A fitted model from which to extract predicted event probabilities.
newdata A data frame containing predictor variable combinations for which to compute predicted event probabilities.
... Additional arguments that are passed on to the current method.
times A vector of times in the range of the response variable, for which the cumulative incidences event probabilities are computed.
cause Identifies the cause of interest among the competing events.
iid Should the iid decomposition be output using an attribute?
average.iid Should the average iid decomposition be output using an attribute?
product.limit If TRUE the survival is computed using the product limit estimator. Otherwise the exponential approximation is used (i.e. \( \exp(-\text{cumulative hazard}) \)).

diag when FALSE the hazard/cumulative hazard/survival for all observations at all times is computed, otherwise it is only computed for the i-th observation at the i-th time.

landmark The starting time for the computation of the cumulative risk.

truncate If TRUE truncates the predicted risks to be in the range [0, 1]. For now only implemented for the Cause Specific Cox model.

Details
In uncensored binary outcome data there is no need to choose a time point.

When operating on models for survival analysis (without competing risks) the function still predicts the risk, as \( 1 - S(t|X) \) where \( S(t|X) \) is survival chance of a subject characterized by \( X \).

When there are competing risks (and the data are right censored) one needs to specify both the time horizon for prediction (can be a vector) and the cause of the event. The function then extracts the absolute risks \( F_c(t|X) \) aka the cumulative incidence of an event of type/cause \( c \) until time \( t \) for a subject characterized by \( X \). Depending on the model it may or not be possible to predict the risk of all causes in a competing risks setting. For example, a cause-specific Cox (CSC) object allows to predict both cases whereas a Fine-Gray regression model (FGR) is specific to one of the causes.

Value
For binary outcome a vector with predicted risks. For survival outcome with and without competing risks a matrix with as many rows as \( \text{NROW(newdata)} \) and as many columns as \( \text{length(times)} \). Each entry is a probability and in rows the values should be increasing.

Author(s)
Thomas A. Gerds <tag@biostat.ku.dk>

Examples
```r
## binary outcome
library(rms)
set.seed(7)
d <- sampleData(80, outcome="binary")
nd <- sampleData(80, outcome="binary")
fit <- lrm(Y~X1+X8, data=d)
predictRisk(fit, newdata=nd)

## survival outcome
## simulate learning and validation data
set.seed(10)
learndat <- sampleData(80, outcome="survival")
valdat <- sampleData(10, outcome="survival")
## use the learning data to fit a Cox model
library(survival)
fitCox <- coxph(Surv(time,event)~X8+X9, data=learndat, x=TRUE, y=TRUE)
```
## suppose we want to predict the survival probabilities for all subjects
## in the validation data at the time points 0, 5, 10:
psurv <- predictRisk(fitCox, newdata = valdat, times = c(0, 5, 10))
## This is a matrix with event probabilities,
## one column for each of the 5 time points
## one row for each validation set individual
## where event probability = 1 - survival probability
## Not run:
if (require("randomForestSRC", quietly = TRUE)){
  # Do the same with a randomSurvivalForest model
  library(randomForestSRC)
  rsfmodel <- rfsr(Surv(time, event) ~ X1 + X2, data = learndat, ntree = 5)
  prsfsurv = predictRisk(rsfmodel, newdata = valdat, times = seq(0, 5, 10))
}
## End(Not run)

## competing risks
library(survival)
library(riskRegression)
library(prodlim)
train <- sampleData(50)
test <- sampleData(10)
cox.fit <- CSC(Hist(time, event) ~ X7 + X8, data = train)
predictRisk(cox.fit, newdata = test, times = c(3, 8), cause = 1)

## with strata and cause 2
cox.fit2 <- CSC(list(Hist(time, event) ~ strata(X1) + X2, Hist(time, cause) ~ X1 + X2), data = train)
predictRisk(cox.fit2, newdata = test, times = c(3, 5), cause = 2)

---

### print.ate

Print Average Treatment Effects

**Description**

Print average treatment effects.

**Usage**

```r
## S3 method for class 'ate'
print(x, estimator = x$estimator, ...)
```

**Arguments**

- `x` object obtained with function ate
- `estimator` [character] The type of estimator relative to which the risks should be output.
- `...` for internal use
See Also

`summary.ate` to obtained a more detailed output `confint.ate` to compute confidence intervals/bands. `ate` to compute the average treatment effects.

print.CauseSpecificCox

Print of a Cause-Specific Cox regression model

Description

Print of a Cause-Specific Cox regression model

Usage

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

Arguments

- `x` Object obtained with CSC
- `...` Passed to `print`

print.FGR

Print of a Fine-Gray regression model

Description

Print of a Fine-Gray regression model

Usage

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

Arguments

- `x` Object fitted with function FGR
- `...` Passed to `cmprsk::summary.crr`
print.influenceTest   \textit{Output of the Difference Between Two Estimates}

\section*{Description}
Output of the difference between two estimates.

\section*{Usage}

\begin{verbatim}
## S3 method for class 'influenceTest'
print(x, digits = 3, ...)
\end{verbatim}

\section*{Arguments}
\begin{itemize}
\item \texttt{x} \hspace{1cm} object obtained with the function \texttt{influenceTest}.
\item \texttt{digits} \hspace{1cm} \texttt{[integer, >0]} indicating the number of decimal places.
\item \texttt{...} \hspace{1cm} Passed to print.
\end{itemize}

\section*{Details}
to display confidence intervals/bands, the \texttt{confint} method needs to be applied on the object.

\section*{See Also}
\texttt{confint.influenceTest} to compute confidence intervals/bands. \texttt{influenceTest} to perform the comparison.

print.IPA   \textit{Print IPA object}

\section*{Description}
Print method for IPA

\section*{Usage}

\begin{verbatim}
## S3 method for class 'IPA'
print(x, digits = 2, ...)
\end{verbatim}

\section*{Arguments}
\begin{itemize}
\item \texttt{x} \hspace{1cm} Object obtained with IPA
\item \texttt{digits} \hspace{1cm} Number of digits
\item \texttt{...} \hspace{1cm} passed to print
\end{itemize}
Author(s)
Thomas A. Gerds <tag@biostat.ku.dk>

print.predictCox  
Print Predictions From a Cox Model

Description
Print predictions from a Cox model.

Usage
## S3 method for class 'predictCox'
print(x, digits = 3, ...)

Arguments
x          object obtained with the function predictCox.
digits    [integer, >0] indicating the number of decimal places.
...        Passed to print.

Details
to display confidence intervals/bands, the confint method needs to be applied on the object.

See Also
confint.predictCox to compute confidence intervals/bands. predictCox to compute the predicted cumulative hazard/survival.

print.predictCSC  
Print Predictions From a Cause-specific Cox Proportional Hazard Regression

Description
Print predictions from a Cause-specific Cox proportional hazard regression.

Usage
## S3 method for class 'predictCSC'
print(x, digits = 3, ...)

print.riskRegression

Arguments

x     object obtained with the function predictCox.
digits [integer, >0] indicating the number of decimal places.
...   Passed to print.

Details

to display confidence intervals/bands, the confint method needs to be applied on the object.

See Also

confint.predictCSC to compute confidence intervals/bands. predict.CauseSpecificCox to compute the predicted risks.

print.riskRegression  Print function for riskRegression models

Description

Print function for riskRegression models

Usage

## S3 method for class 'riskRegression'
print(x, times, digits = 3, eps = 10^-4, verbose = TRUE, conf.int = 0.95, ...)

Arguments

x     Object obtained with ARR, LRR or riskRegression
times  Time points at which to show time-dependent coefficients
digits Number of digits for all numbers but p-values
eps    p-values smaller than this number are shown as such
verbose Level of verbosity
conf.int level of confidence. default is 0.95
...    not used
print.Score

Description
Print method for risk prediction scores

Usage
## S3 method for class 'Score'
print(x, digits, ...)

Arguments
x Object obtained with Score.list
digits Number of digits
... passed to print

print.subjectWeights

Description
Print subject weights

Usage
## S3 method for class 'subjectWeights'
print(x, digits = 3, ...)

Arguments
x Subject weights
digits Digits
... not used
reconstructData  

**Description**
Reconstruct the original dataset from the elements stored in the coxph object

**Usage**

reconstructData(object)

**Arguments**

object  
a coxph object.

**Author(s)**
Brice Ozenne broz@sund.ku.dk and Thomas A. Gerds tag@biostat.ku.dk

riskLevelPlot  

**Description**
Level plots for predicted risks

**Usage**

riskLevelPlot(
  object,
  formula,
  data = parent.frame(),
  horizon = NULL,
  cause = 1,
  ...
)

**Arguments**

object  
risk prediction model object
formula  
formula
data  
data
horizon  
time point
cause  
cause of interest
...  
passed to lattice::levelplot
riskLevelPlot

Details

Level plots for predicted risks

Author(s)

Thomas A. Gerds <tag@biostat.ku.dk>

Examples

# ---------- logistic regression ---------------------
expit <- function(x){exp(x)/(1+exp(x))}
partyData <- function(N={
    Age <- runif(N,.5,15)
    Parasites <- rnorm(N,mean=3.5-0.03*Age)
    Fever <- factor(rbinom(N,1,expit(-3.5-.3*Age+.55*Parasites+0.15*Age*Parasites)))
    data.frame(Fever,Age,Parasites)
})
d <- partyData(100)
f <- glm(Fever~Age+Parasites,data=d,family="binomial")
riskLevelPlot(f,Fever~Age+Parasites,d)
## Not run:
if (require("randomForest",quietly=TRUE)){
  rf <- randomForest::randomForest(Fever~Age+Parasites,data=d)
riskLevelPlot(f,Fever~Age+Parasites,d)
riskLevelPlot(rf,Fever~Age+Parasites,d)
}
## End(Not run)

# ---------- survival analysis ---------------------
# --simulate an artificial data frame
# with survival response and three predictors
library(survival)
library(prodlim)
set.seed(140515)
sdat <- sampleData(43,outcome="survival")
# -- fit a Cox regression model
survForm = Surv(time,event) ~ X8 + X9
cox <- coxph(survForm, data = sdat,x=TRUE)
# --choose a time horizon for the predictions and plot the risks
timeHorizon <- floor(median(sdat$times))
riskLevelPlot(cox, survForm, data = sdat, horizon = timeHorizon)

# ---------- competing risks ---------------------

# -- simulate an artificial data frame
# with competing cause response and three predictors
library(cmprsk)
library(riskRegression)
set.seed(140515)
crdat <- sampleData(49)

# -- fit a cause-specific Cox regression model
crForm <- Hist(time,event)~X8+X9
csCox <- CSC(crForm, data=crdat)

# -- choose a time horizon and plot the risk for a given cause
timeHorizon <- floor(median(crdat$time))
riskLevelPlot(csCox, crForm, data = crdat, horizon = timeHorizon, cause = 1)

---

riskRegression

**Risk Regression** Fits a regression model for the risk of an event – allowing for competing risks.

---

**Description**

This is a wrapper for the function `comp.risk` from the timereg package. The main difference is one marks variables in the formula that should have a time-dependent effect whereas in `comp.risk` one marks variables that should have a time constant (proportional) effect.

**Usage**

```r
riskRegression(
  formula,
  data,
  times,
  link = "relative",
  cause,
  conf.int = TRUE,
  cens.model,
  cens.formula,
  max.iter = 50,
  conservative = TRUE,
  ...
)
```

**Arguments**

- `formula`: Formula where the left hand side specifies the event history `event.history` and the right hand side the linear predictor. See examples.
- `data`: The data for fitting the model in which includes all the variables included in formula.
- `times`: Vector of times. For each time point in `times` estimate the baseline risk and the time-varying coefficients.
riskRegression

link  "relative" for the absolute risk regression model. "logistic" for the logistic risk regression model. "prop" for the Fine-Gray regression model.

cause  The cause of interest.

conf.int  If TRUE return the iid decomposition, that can be used to construct confidence bands for predictions.

cens.model  Specified the model for the (conditional) censoring distribution used for deriving weights (IFPW). Defaults to "KM" (the Kaplan-Meier method ignoring covariates) alternatively it may be "Cox" (Cox regression).

cens.formula  Right hand side of the formula used for fitting the censoring model. If not specified the right hand side of formula is used.

max.iter  Maximal number of iterations.

conservative  If TRUE use variance formula that ignores the contribution by the estimate of the inverse of the probability of censoring weights

...  Further arguments passed to comp.risk

Author(s)

Thomas A. Gerds <tag@biostat.ku.dk>, Thomas H. Scheike <ts@biostat.ku.dk>

References


Scheike, Zhang and Gerds (2008), Predicting cumulative incidence probability by direct binomial regression, Biometrika, 95, 205-220.


Examples

library(prodlim)
data(Melanoma,package="riskRegression")
## tumor thickness on the log-scale
Melanoma$logthick <- log(Melanoma$thick)

# Single binary factor

## absolute risk regression
library(survival)
library(prodlim)
fit.arr <- ARR(Hist(time,status)~sex,data=Melanoma,cause=1)
print(fit.arr)
# show predicted cumulative incidences
plot(fit.arr,col=3:4,newdata=data.frame(sex=c("Female","Male")))
# compare with non-parametric Aalen-Johansen estimate
library(prodlim)
fit.aj <- prodlim(Hist(time,status)~sex,data=Melanoma)
plot(fit.aj,conf.int=FALSE)
plot(fit.arr,add=TRUE,col=3:4,newdata=data.frame(sex=c("Female","Male")))

# with time-dependent effect
fit.tarr <- ARR(Hist(time,status)~strata(sex),data=Melanoma,cause=1)
plot(fit.tarr,newdata=data.frame(sex=c("Female","Male")))

# logistic risk regression
fit.lrr <- LRR(Hist(time,status)~sex,data=Melanoma,cause=1)
summary(fit.lrr)

# Single continuous factor

## tumor thickness on the log-scale
Melanoma$logthick <- log(Melanoma$thick)

# absolute risk regression
fit2.arr <- ARR(Hist(time,status)~logthick,data=Melanoma,cause=1)
print(fit2.arr)
# show predicted cumulative incidences
plot(fit2.arr,col=1:5,newdata=data.frame(logthick=quantile(Melanoma$logthick)))

# comparison with nearest neighbor non-parametric Aalen-Johansen estimate
library(prodlim)
fit2.aj <- prodlim(Hist(time,status)~logthick, data=Melanoma)
plot(fit2.aj,conf.int=FALSE,newdata=data.frame(logthick=quantile(Melanoma$logthick)))
plot(fit2.arr,add=TRUE,col=1:5,lty=3,newdata=data.frame(logthick=quantile(Melanoma$logthick)))

# logistic risk regression
fit2.lrr <- LRR(Hist(time,status)~logthick,data=Melanoma,cause=1)
summary(fit2.lrr)

# change model for censoring weights
library(rms)
fit2a.lrr <- LRR(Hist(time,status)~logthick,
data=Melanoma,
cause=1,
cens.model="cox",
cens.formula=~sex+epicel+ulcer+age+logthick)
summary(fit2a.lrr)

# compare prediction performance
Score(list(ARR=fit2.arr,AJ=fit2.aj,LRR=fit2.lrr),formula=Hist(time,status)~1,data=Melanoma)

# multiple regression
library(riskRegression)
library(prodlim)
# absolute risk model
multi.arr <- ARR(Hist(time,status)~logthick+sex+age+ulcer,data=Melanoma,cause=1)

# stratified model allowing different baseline risk for the two gender
multi.arr <- ARR(Hist(time,status)~thick+strata(sex)+age+ulcer,data=Melanoma,cause=1)

# stratify by a continuous variable: strata(age)
multi.arr <- ARR(Hist(time,status)~tp(thick,power=0)+strata(age)+sex+ulcer,
data=Melanoma,
cause=1)

fit.arr2a <- ARR(Hist(time,status)~tp(thick,power=1),data=Melanoma,cause=1)
summary(fit.arr2a)
fit.arr2b <- ARR(Hist(time,status)~timevar(thick),data=Melanoma,cause=1)
summary(fit.arr2b)

## logistic risk model
fit.lrr <- LRR(Hist(time,status)~thick,data=Melanoma,cause=1)
summary(fit.lrr)

## nearest neighbor non-parametric Aalen-Johansen estimate
library(prodlim)
fit.aj <- prodlim(Hist(time,status)~thick,data=Melanoma)
plot(fit.aj,conf.int=FALSE)

# prediction performance
x <- Score(list(fit.arr2a,fit.arr2b,fit.lrr),
data=Melanoma,
formula=Hist(time,status)~1,
cause=1,
split.method="none")

---

**riskRegression.options**

*Global options for riskRegression*

**Description**

Output and set global options for the riskRegression package.

**Usage**

riskRegression.options(...)
Arguments

... for now limited to method.predictRisk and method.predictRiskIID.

Details

only used by the ate function.

Examples

```r
options <- riskRegression.options()

## add new method.predictRiskIID
riskRegression.options(method.predictRiskIID = c(options$method.predictRiskIID,"xx"))
riskRegression.options()
```

---

```r
rowCenter_cpp Apply - by row
```

Description

Fast computation of `sweep(X, MARGIN = 2, FUN = ", STATS = center)`

Usage

```r
rowCenter_cpp(X, center)
```

Arguments

- **X**: A matrix.
- **center**: a numeric vector of length equal to the number of rows of `x`

Value

A matrix of same size as `X`.

Author(s)

Brice Ozenne <broz@sund.ku.dk>

Examples

```r
x <- matrix(1,6,5)
sweep(x, MARGIN = 2, FUN = ", STATS = 1:5)
rowCenter_cpp(x, 1:5 )
rowCenter_cpp(x, colMeans(x) )
```
**rowCumSum**

*Apply cumsum in each row*

---

**Description**

Fast computation of `t(apply(x,1,cumsum))`

**Usage**

```r
rowCumSum(x)
```

**Arguments**

- `x`: A matrix.

**Value**

A matrix of same size as `x`.

**Author(s)**

Thomas Alexander Gerds <tag@biostat.ku.dk>

**Examples**

```r
x <- matrix(1:8, ncol=2)
rowCumSum(x)
```

---

**rowMultiply_cpp**

*Apply * by row*

---

**Description**

Fast computation of `sweep(X, MARGIN = 2, FUN = "+", STATS = scale)`

**Usage**

```r
rowMultiply_cpp(X, scale)
```

**Arguments**

- `X`: A matrix.
- `scale`: a numeric vector of length equal to the number of rows of `X`

**Value**

A matrix of same size as `X`. 
Author(s)
Brice Ozenne <broz@sund.ku.dk>

Examples
\[
x <- \text{matrix}(1,6,5) \\
sweep(x, MARGIN = 2, FUN = "+", STATS = 1:5) \\
\text{rowMultiply\_cpp}(x, 1:5)
\]
\[
\text{rowMultiply\_cpp}(x, 1/\text{colMeans}(x))
\]

---

**rowPaste**  
Collapse Rows of Characters.

Description
Collapse rows of characters. Fast alternative to `apply(x,1,paste0,collapse="")`

Usage

\[
\text{rowPaste}(\text{object})
\]

Arguments

- **object**  
  A matrix/data.frame/list containing the characters.

Examples

```r
## Not run:
M <- \text{matrix}(\text{letters}, \text{nrow} = 26, \text{ncol} = 2)
\text{rowPaste}(M)
## End(Not run)
```

---

**rowScale\_cpp**  
Apply / by row

Description
Fast computation of `sweep(X, MARGIN = 2, FUN = "/", STATS = scale)`

Usage

\[
\text{rowScale\_cpp}(X, \text{scale})
\]
**rowSumsCrossprod**

**Arguments**

- **X**
  A matrix.
- **scale**
  A numeric vector of length equal to the number of rows of `x`.

**Value**

A matrix of same size as `X`.

**Author(s)**

Brice Ozenne <broz@sund.ku.dk>

**Examples**

```r
x <- matrix(1, 6, 5)
sweep(x, MARGIN = 2, FUN = "/", STATS = 1:5)
rowScale_cpp(x, 1:5)
rowScale_cpp(x, colMeans(x))
```

**Description**

Fast computation of `crossprod(rowSums(X), Y)`

**Usage**

`rowSumsCrossprod(X, Y, transposeY)`

**Arguments**

- **X**
  A matrix with dimensions n*k. Hence the result of `rowSums(X)` has length n.
- **Y**
  A matrix with dimensions n*m. Can be a matrix with dimension m*n but then `transposeY` should be TRUE.
- **transposeY**
  Logical. If TRUE transpose `Y` before matrix multiplication.

**Value**

A vector of length m.

**Author(s)**

Thomas Alexander Gerds <tag@biostat.ku.dk>
Examples

```r
x <- matrix(1:10, nrow = 5)
y <- matrix(1:20, ncol = 4)
rowSumsCrossprod(x, y, 0)

x <- matrix(1:10, nrow = 5)
y <- matrix(1:20, ncol = 5)
rowSumsCrossprod(x, y, 1)
```

---

**sampleData**

*Simulate data with binary or time-to-event outcome*

**Description**

Simulate data with binary outcome and 10 covariates.

**Usage**

```r
sampleData(n, outcome = "competing.risks",
formula = ~ f(X1, 2) + f(X2, -0.033) + f(X3, 0.4) + f(X6, 1) + f(X7, -1) + f(X8, 0.5) + f(X9, -1),
intercept = 0)
```

**Arguments**

- `n`: Sample size
- `outcome`: Character vector. Response variables are generated according to keywords: "binary" = binary response, "survival" = survival response, "competing.risks" = competing risks response
- `formula`: Specify regression coefficients
- `intercept`: For binary outcome the intercept of the logistic regression.
- `n.intervals`: `sampleDataTD` only: the maximum number of episodes in which the covariates are updated.

**Details**

For the actual lava::regression parameters see the function definition.

**Value**

Simulated data as data.table with `n` rows and the following columns: `Y` (binary outcome), `time` (non-binary outcome), `event` (non-binary outcome), `X1`-`X5` (binary predictors), `X6`-`X10` (continuous predictors)
Author(s)

Thomas A. Gerds <tag@biostat.ku.dk>

See Also

lvm

Examples

set.seed(10)
sampleData(10,outcome="binary")
sampleData(10,outcome="survival")
sampleData(10,outcome="competing.risks")

Description

Save confidential Cox objects

Usage

saveCoxConfidential(object, times)

Arguments

object An object of class coxph.
times The times at which we want to predict risk.

Details

This function can save coxph objects such that we do not need to export the data on which it was fitted at given times.

Examples

library(survival)
library(lava)
set.seed(18)
trainSurv <- sampleData(300,outcome="survival")
testSurv <- sampleData(40,outcome="survival")
fit = coxph(Surv(time,event)=X1+X2+X3+X7+X9,data=trainSurv, y=TRUE, x = TRUE)
u=saveCoxConfidential(fit,times=3)
## Not run:
# write object as plain text file
sink("~/tmp/u.R")
cat("U <- ")
Score

Score risk predictions

Description

Methods to score the predictive performance of risk markers and risk prediction models

Usage

Score(object, ...)

## S3 method for class 'list'
Score(
oBJECT, 
formula, 
data, 
metrics = c("auc", "brier"), 
summary = NULL, 
plots = NULL, 
cause, 
times, 
landmarks, 
use.event.times = FALSE, 
null.model = TRUE, 
se.fit = TRUE, 
conservative = FALSE, 
multi.split.test = FALSE, 
conf.int = 0.95,

doput(u)
sink(NULL)
# reload object
source("~/tmp/u.R")
class(u) <- "CoxConfidential"

## End(Not run)
predictRisk(u,newdata=testSurv)
cox1 = coxph(Surv(time,event)~strata(X1)+X2+X3+X7+X9,data=trainSurv, y=TRUE, x = TRUE)
z<-saveCoxConfidential(cox1,c(2,5))
dput(z) ## get output to copy object

all.equal(predictRisk(z,newdata=testSurv),
predictRisk(cox1,newdata=testSurv,times=c(2,5)))

cox2 = coxph(Surv(time,event)-X1+X2+X7+X9,data=trainSurv, y=TRUE, x = TRUE)
z<-saveCoxConfidential(cox2,c(2,5))

all.equal(predictRisk(z,testSurv),predictRisk(z,testSurv,c(2,5)))
contrasts = TRUE,
probs = c(0, 0.25, 0.5, 0.75, 1),
cens.method = "ipcw",
cens.model = "cox",
split.method,
B,
M,
seed,
trainseeds,
parallel = c("no", "multicore", "snow", "as.registered"),
cpus = 1,
cl = NULL,
progress.bar = 3,
errorhandling = "pass",
keep,
predictRisk.args,
debug = 0L,
censoring.save.memory = FALSE,
breaks = seq(0, 1, 0.01),
roc.method = "vertical",
roc.grid = switch(roc.method, vertical = seq(0, 1, 0.01), horizontal = seq(1, 0, -0.01)),
cutpoints = NULL,
... 
)

Arguments

object List of risk predictions (see details and examples).
...
Named list containing additional arguments that are passed on to the predictRisk methods corresponding to object. See examples.

formula A formula which identifies the outcome (left hand side). E.g., Y ~ 1 for binary and Hist(time, status) ~ 1 for time-to-event outcome. In right censored data, the right hand side of the formula is used to estimate the inverse probability of censoring weights (IPCW) model.

data data.frame or data.table in which the formula can be interpreted.

metrics Character vector specifying which metrics to apply. Case does not matter. Choices are "AUC" and "Brier".

summary Character vector specifying which summary statistics to apply to the predicted risks. Choices are "risks", "IPA", "riskQuantile" and "ibs". Can be all c("risks", "IPA", "riskQuantile", "ibs") or a subset thereof.

- "risks" adds the predicted risks to the output.
- "ipa" computes the index of prediction accuracy (AKA R-squared) based on Brier scores for model vs null model
- "riskQuantile" calculates time-point specific boxplots for the predicted risks (or biomarker values) conditional on the outcome at the time-point.
"ibs" calculates integrated Brier scores across the time points at which the Brier score is computed. This works only with time-to-event outcome and the results depend on the argument times.

Set to NULL to avoid estimation of summary statistics.

**plots**
Character vector specifying for which plots to put data into the result. Currently implemented are "ROC", "Calibration" and "boxplot". In addition, one can plot AUC and Brier score as function of time as soon as times has at least two different values.

**cause**
Event of interest. Used for binary outcome Y to specify that risks are risks of the event Y=event and for competing risks outcome to specify the cause of interest.

**times**
For survival and competing risks outcome: list of prediction horizons. All times which are greater than the maximal observed time in the data set are automatically removed. Note that the object returned by the function may become huge when the prediction performance is estimated at many prediction horizons.

**landmarks**
Not yet implemented.

**use.event.times**
If TRUE merge all unique event times with the vector given by argument times.

**null.model**
If TRUE fit a risk prediction model which ignores the covariates and predicts the same value for all subjects. The model is fitted using data and the left hand side of formula. For binary outcome this is just the empirical prevalence. For (right censored) time to event outcome, the null models are equal to the Kaplan-Meier estimator (no competing risks) and the Aalen-Johansen estimator (with competing risks).

**se.fit**
Logical or 0 or 1. If FALSE or 0 do not calculate standard errors.

**conservative**
Logical, only relevant in right censored data. If TRUE ignore variability of the estimate of the inverse probability of censoring weights when calculating standard errors for prediction performance parameters. This can potentially reduce computation time and memory usage at a usually very small expense of a slightly higher standard error.

**multi.split.test**
Logical or 0 or 1. If FALSE or 0 do not calculate multi-split tests. This argument is ignored when split.method is "none".

**conf.int**
Either logical or a numeric value between 0 and 1. In right censored data, confidence intervals are based on Blanche et al (see references). Setting FALSE prevents the computation of confidence intervals. TRUE computes 95 percent confidence intervals and corresponding p-values for AUC and Brier score. If set to 0.87, the level of significance is 13 percent. So, do not set it to 0.87.

**contrasts**
Either logical or a list of contrasts. A list of contrasts defines which risk prediction models (markers) should be contrasted with respect to their prediction performance. If TRUE do all possible comparisons. For example, when object is a list with two risk prediction models and null.model=TRUE setting TRUE is equivalent to list(c(0,1,2),c(1,2)) where c(0,1,2) codes for the two comparisons: 1 vs 0 and 2 vs 0 (positive integers refer to elements of object, 0 refers to the benchmark null model which ignores the covariates). This again is equivalent to explicitly setting list(c(0,1),c(0,2),c(1,2)). A more complex example: Suppose object has 7 elements and you want to do the following 3 comparisons: 6 vs 3, 2 vs 5 and 2 vs 3, you should set contrasts=c(6,3),c(2,5,3).
probs

Quantiles for retrospective summary statistics of the predicted risks. This affects
the result of the function boxplot.Score.

cens.method

Method for dealing with right censored data. Either "ipcw" or "pseudo". Here
IPCW refers to inverse probability of censoring weights and pseudo for jack-
knife pseudo values. Right now pseudo values are only used for calibration
curves.

cens.model

Model for estimating inverse probability of censored weights (IPCW). Imple-
mented are the Kaplan-Meier method ("km") and Cox regression ("cox") both
applied to the censored times. If the right hand side of formula does not spec-
ify covariates, the Kaplan-Meier method is used even if this argument is set to
"cox". Also implemented is a template for users specifying other models to
estimate the IPCW. Here the user should be supply a function, taking as input
a "formula" and "data". This does come at the cost of only being able to
calculate conservative confidence intervals.

split.method

Method for cross-validation. Right now the only choices are bootcv, cvk and
loob. In the first case, bootstrap learning sets are drawn without replacement (argument M) from data. The data not included in the current boot-
strap learning set are used as validation set to compute the prediction perfor-
mance. In the second case, k-fold cross-validation is performed. Note that k has
to be an explicit number, e.g. 5 or 10, when passing this as an argument. In the
third case, leave-one-out bootstrap cross-validation is performed for the Brier
score and leave-pair-out bootstrap cross-validation is performed for the AUC.

B

Number of bootstrap sets for cross-validation. B should be set to 1, when k-fold
cross-validation is used.

M

Size of subsamples for bootstrap cross-validation. If specified it has to be an
integer smaller than the size of data.

seed

Super seed for setting training data seeds when randomly splitting (bootstrapping) the data during cross-validation.

trainseeds

Seeds for training models during cross-validation.

parallel

The type of parallel operation to be used (if any). If missing, the default is "no".

ncpus

integer: number of processes to be used in parallel operation.

c1

An optional parallel or snow cluster for use if parallel = "snow". If not
supplied, a cluster on the local machine is created for the duration of the Score
call.

progress.bar

Style for txtProgressBar. Can be 1,2,3 see help(txtProgressBar) or NULL
to avoid the progress bar.

errorhandling

Argument passed as .errorhandling to foreach. Default is "pass".

keep

list of characters (not case sensitive) which determines additional output. "residuals"
provides Brier score residuals and "splitindex" provides sampling index used
to split the data into training and validation sets. It is a function, whose argu-
ment is the bootstrap sample, which one wishes to look at. "vcov" provides
the variance-covariance matrix for the estimates. "iid" provides the estimated
influence function of the estimates.
predictRisk.args

A list of argument-lists to control how risks are predicted. The names of the lists should be the S3-classes of the object. The argument-lists are then passed on to the S3-class specific predictRisk method. For example, if your object contains one or several random forest model fitted with the function randomForestSRC::rfsrc then you can specify additional arguments for the function riskRegression::predictRisk.rfsrc which will pass these on to the function randomForestSRC::predict.rfsrc. A specific example in this case would be list(rfsrc=list(na.action="na.impute").

A more flexible approach is to write a new predictRisk S3-method. See Details.

debuge

Logical. If TRUE indicate landmarks in progress of the program.

censoring.save.memory

Only relevant in censored data where censoring weights are obtained with Cox regression and argument conservative is set to FALSE. If TRUE, save memory by not storing the influence function of the cumulative hazard of the censoring as a matrix when calculating standard errors with Cox censoring. This can allow one to use Score on larger test data sets, but may be slower.

breaks

Break points for computing the ROC curve. Defaults to seq(0,1,.01) when some form of crossvalidation is applied, otherwise to all unique values of the predictive marker.

roc.method


roc.grid

Grid points for the averaging of ROC curves. A sequence of values at which to compute averages across the ROC curves obtained for different data splits during crossvalidation.

cutpoints

If not NULL, estimates and standard errors of the TPR (True Positive Rate), FPR (False Positive Rate), PPV (Positive Predictive Value), and NPV (Negative Predictive Value) are given at the cutpoints. These values are saved in object$AUC$res.cut.

Details

The function implements a toolbox for the risk prediction modeller: all tools work for the three outcomes: (1) binary (uncensored), (2) right censored time to event without competing risks, (3) right censored time to event with competing risks.

Computed are the (time-dependent) Brier score and the (time-dependent) area under the ROC curve for a list of risk prediction models either in external validation data or in the learning data using bootstrap cross-validation. The function optionally provides results for plotting (time-point specific) ROC curves, for (time-point specific) calibration curves and for (time-point specific) retrospective boxplots.

For uncensored binary outcome the Delong-Delong test is used to contrast AUC of rival models. In right censored survival data (with and without competing risks) the p-values correspond to Wald tests based on standard errors obtained with an estimate of the influence function as described in detail in the appendix of Blanche et al. (2015).

This function works with one or multiple models that predict the risk of an event R(t|X) for a subject characterized by predictors X at time t. With binary endpoints (outcome 0/1 without time
component) the risk is simply \( R(X) \). In case of a survival object without competing risks the function still works with predicted event probabilities, i.e., \( R(t|X) = 1 - S(t|X) \) where \( S(t|X) \) is the predicted survival chance for subject \( X \) at time \( t \).

The already existing predictRisk methods (see methods(predictRisk)) may not cover all models and methods for predicting risks. But users can quickly extend the package as explained in detail in Mogensen et al. (2012) for the predecessors `pec::predictSurvProb` and `pec::predictEventProb` which have been unified as `riskRegression::predictRisk`.

Bootstrap Crossvalidation (see also Gerds & Schumacher 2007 and Mogensen et al. 2012)

\( B=10, M \) (not specified or \( M=\text{NROW(data)} \)) Training of each of the models in each of 10 bootstrap data sets (learning data sets). Learning data sets are obtained by sampling \( \text{NROW(data)} \) subjects of the data set with replacement. There are roughly \( .632 * \text{NROW(data)} \) subjects in the learning data (inbag) and \( .368 * \text{NROW(data)} \) subjects not in the validation data sets (out-of-bag).

These are used to estimate the scores: AUC, Brier, etc. Reported are averages across the 10 splits.

```r
## Bootstrap with replacement set.seed(13) N=17 data = data.frame(id=1:N, y=rbinom(N,1,.3),x=rnorm(N)) boot.index = sample(1:N,size=N,replace=TRUE) boot.index inbag = 1:N outofbag = !inbag learn.data = data[inbag] val.data = data[!inbag] riskRegression:::getSplitMethod("bootcv",B=10,N=17)$index
NOTE: the number .632 is the expected probability to draw one subject (for example subject 1) with replacement from the data, which does not depend on the sample size: B=10000 N=137
N=300 mean(sapply(1:B, function(b){match(1,sample(1:N,size=N,replace=TRUE),nomatch=0)})
## Bootstrap without replacement (training size set to be 70 percent of data) B=10, M=.7
Training of each of the models in each of 10 bootstrap data sets (learning data sets). Learning data sets are obtained by sampling \( \text{round(.8*\text{NROW(data)})} \) subjects of the data set without replacement. There are \( \text{NROW(data)} - \text{round(.8*\text{NROW(data)})} \) subjects not in the learning data sets. These are used to estimate the scores: AUC, Brier, etc. Reported are averages across the 10 splits. set.seed(13) N=17 data = data.frame(id=1:N, y=rbinom(N,1,.3),x=rnorm(N)) boot.index = sample(1:N,size=M,replace=FALSE) boot.index inbag = 1:N outofbag = !inbag learn.data = data[inbag] val.data = data[!inbag] riskRegression:::getSplitMethod("bootcv",B=10,N=17,M=.7)
```

**Value**

List with scores and assessments of contrasts, i.e., tests and confidence limits for performance and difference in performance (AUC and Brier), summaries and plots. Most elements are in `data.table` format.

**Author(s)**

Thomas A Gerds <tag@biostat.ku.dk> and Paul Blanche <paul.blanche@univ-ubs.fr>

**References**


E. Graf et al. (1999), Assessment and comparison of prognostic classification schemes for survival data. Statistics in Medicine, vol 18, pp= 2529–2545.


Examples

# binary outcome
library(lava)
set.seed(18)
learndat <- sampleData(48,outcome="binary")
testdat <- sampleData(40,outcome="binary")

## score logistic regression models
lr1 = glm(Y~X1+X2+X7+X9,data=learndat,family=binomial)
lr2 = glm(Y~X3+X5,data=learndat,family=binomial)
Score(list("LR(X1+X2+X7+X9)"=lr1,"LR(X3+X5)"=lr2),formula=Y~1,data=testdat)

## ROC curve and calibration plot
xb=Score(list("LR(X1+X2+X7+X9)"=lr1,"LR(X3+X5+X6)"=lr2),formula=Y~1,
data=testdat,plots=c("calibration","ROC"))

## Not run: plotROC(xb)
plotCalibration(xb)

## End(Not run)

## compute AUC for a list of continuous markers
markers = as.list(testdat[,.(X6,X7,X8,X9,X10)])
Score(markers,formula=Y~1,data=testdat,metrics=c("auc"))
# cross-validation
## Not run:
set.seed(10)
learndat=sampleData(400,outcome="binary")
lr1a = glm(Y~X6, data=learndat, family=binomial)
lr2a = glm(Y~X7+X8+X9, data=learndat, family=binomial)
## bootstrap cross-validation
x1=Score(list("LR1"=lr1a,"LR2"=lr2a), formula=Y~1, data=learndat, split.method="bootcv", B=100)
x1
## leave-one-out and leave-pair-out bootstrap
x2=Score(list("LR1"=lr1a,"LR2"=lr2a), formula=Y~1, data=learndat, split.method="loob",
        B=100, plots="calibration")
x2
## 5-fold cross-validation
x3=Score(list("LR1"=lr1a,"LR2"=lr2a), formula=Y~1, data=learndat, split.method="cv5",
        B=1, plots="calibration")
x3

## End(Not run)

# survival outcome

# Score Cox regression models
## Not run: library(survival)
library(rms)
library(prodlim)
set.seed(18)
trainSurv <- sampleData(100, outcome="survival")
testSurv <- sampleData(40, outcome="survival")
cox1 = coxph(Surv(time, event)~X1+X2+X7+X9, data=trainSurv, y=TRUE, x = TRUE)
cox2 = coxph(Surv(time, event)~X3+X5+X6, data=trainSurv, y=TRUE, x = TRUE)
x=Score(list("Cox(X1+X2+X7+X9)"=cox1,"Cox(X3+X5+X6)"=cox2),
        formula=Surv(time, event)~1, data=testSurv, conf.int=FALSE, times=c(5,8))
## Use Cox to estimate censoring weights
y=Score(list("Cox(X1+X2+X7+X9)"=cox1,"Cox(X3+X5+X6)"=cox2),
        formula=Surv(time, event)~X1+X8, data=testSurv, conf.int=FALSE, times=c(5,8))
## Use GLMnet to estimate censoring weights
z=Score(list("Cox(X1+X2+X7+X9)"=cox1,"Cox(X3+X5+X6)"=cox2),
        formula=Surv(time, event)~X1+X8, cens.model = "GLMnet", data=testSurv,
        conf.int=FALSE, times=c(5,8))
## Use hal9001 to estimate censoring weights
w=Score(list("Cox(X1+X2+X7+X9)"=cox1,"Cox(X3+X5+X6)"=cox2),
        formula=Surv(time, event)~X1+X8, cens.model = "Hal9001", data=testSurv,
        conf.int=FALSE, times=c(5,8))
x
y
z
w

## End(Not run)

## Not run: library(survival)
library(rms)
library(prodlim)
set.seed(18)
trainSurv <- sampleData(100, outcome="survival")
testSurv <- sampleData(40, outcome="survival")
cox1 = coxph(Surv(time,event)~X1+X2+X7+X9, data=trainSurv, y=TRUE, x = TRUE)
cox2 = coxph(Surv(time,event)~X3+X5+X6, data=trainSurv, y=TRUE, x = TRUE)
xs=Score(list("Cox(X1+X2+X7+X9)"=cox1,"Cox(X3+X5+X6)"=cox2),
  formula=Surv(time,event)-1, data=testSurv, conf.int=FALSE, times=c(5,8))
xs
# Integrated Brier score
# Not run:
xs=Score(list("Cox(X1+X2+X7+X9)"=cox1,"Cox(X3+X5+X6)"=cox2),
  formula=Surv(time,event)-1, data=testSurv, conf.int=FALSE, summary="ibs",
  times=sort(unique(testSurv\$time)))
# time-dependent AUC for list of markers
# Not run: survmarkers = as.list(testSurv[,.(X6,X7,X8,X9,X10)])
Score(survmarkers,
  formula=Surv(time,event)-1, metrics="auc", data=testSurv,
  conf.int=TRUE, times=c(5,8))
# compare models on test data
Score(list("Cox(X1+X2+X7+X9)"=cox1,"Cox(X3+X5+X6)"=cox2),
  formula=Surv(time,event)-1, data=testSurv, conf.int=TRUE, times=c(5,8))
# crossvalidation models in train data
# Not run:
library(survival)
set.seed(18)
trainSurv <- sampleData(400, outcome="survival")
cox1 = coxph(Surv(time,event)~X1+X2+X7+X9, data=trainSurv, y=TRUE, x = TRUE)
cox2 = coxph(Surv(time,event)~X3+X5+X6, data=trainSurv, y=TRUE, x = TRUE)
x1 = Score(list("Cox(X1+X2+X7+X9)"=cox1,"Cox(X3+X5+X6)"=cox2),
  formula=Surv(time,event)-1, data=trainSurv, conf.int=TRUE, times=c(5,8),
  split.method="loob", B=100, plots="calibration")
x2 = Score(list("Cox(X1+X2+X7+X9)"=cox1,"Cox(X3+X5+X6)"=cox2),
  formula=Surv(time,event)-1, data=trainSurv, conf.int=TRUE, times=c(5,8),
  split.method="bootcv", B=100)
# restrict number of comparisons
# Not run:
Score(list("Cox(X1+X2+X7+X9)"=cox1,"Cox(X3+X5+X6)"=cox2),
  formula=Surv(time,event)-1, data=trainSurv, contrasts=TRUE,
null.model=FALSE, conf.int=TRUE, times=c(5,8), split.method="bootcv", B=3)

# competing risks outcome
set.seed(18)
trainCR <- sampleData(400, outcome="competing.risks")
testCR <- sampleData(400, outcome="competing.risks")
library(riskRegression)
library(cmprsk)
# Cause-specific Cox regression
csc1 = CSC(Hist(time,event)-X1+X2+X7+X9, data=trainCR)
csc2 = CSC(Hist(time,event)-X3+X5+X6, data=trainCR)
# Fine-Gray regression
fgr1 = FGR(Hist(time,event)-X1+X2+X7+X9, data=trainCR, cause=1)
fgr2 = FGR(Hist(time,event)-X3+X5+X6, data=trainCR, cause=1)
Score(list("CSC(X1+X2+X7+X9)" = csc1, "CSC(X3+X5+X6)" = csc2,
"FGR(X1+X2+X7+X9)" = fgr1, "FGR(X3+X5+X6)" = fgr2),
formula=Hist(time,event)-1, data=testCR, se.fit=1L, times=c(5,8))

## End(Not run)

## Not run:
# reproduce some results of Table IV of Blanche et al. Stat Med 2013
data(Paquid)
ResPaquid <- Score(list("DSST" = -Paquid$DSST, "MMSE" = -Paquid$MMSE),
formula=Hist(time,status)-1, data=Paquid,
null.model = FALSE, conf.int=TRUE, metrics=c("auc"),
times=c(3,5,10), plots="ROC")
ResPaquid
plotROC(ResPaquid, time=5)

## End(Not run)

## Not run:
# parallel options
# by erikvona: Here is a generic example of using future
# and doFuture, works great with the current version:
library(riskRegression)
library(future)
library(foreach)
library(doFuture)
library(survival)
# Register all available cores for parallel operation
plan(multiprocess, workers = availableCores())
registerDoFuture()
set.seed(10)
trainSurv <- sampleData(400, outcome="survival")
cox1 = coxph(Surv(time,event)-X1+X2+X7+X9, data=trainSurv, y=TRUE, x = TRUE)
# Bootstrapping on multiple cores
x1 = Score(list("Cox(X1+X2+X7+X9)"=cox1),
    formula=Surv(time,event)-1,data=trainSurv, times=c(5,8),
    parallel = "as.registered", split.method="bootcv",B=100)
## End(Not run)

---

## `selectCox`

### Description
This is a wrapper function which first selects variables in the Cox regression model using `fastbw` from the `rms` package and then returns a fitted Cox regression model with the selected variables.

### Usage

```r
selectCox(formula, data, rule = "aic")
```

---

## `score.wglm`

### Description
Compute the first derivative of the log-likelihood for IPCW logistic regressions.

### Usage

```r
## S3 method for class 'wglm'
score(x, indiv = FALSE, times = NULL, simplifies = TRUE, ...)
```

### Arguments

- `x`: a `wglm` object.
- `indiv`: [logical] should the individual score be output? Otherwise the total score (i.e. summed over all individuals will be output).
- `times`: [numeric vector] time points at which the score should be output.
- `simplifies`: [logical] should the output be converted to a matrix when only one timepoint is requested. Otherwise will always return a list.
- `...`: Not used.
**selectJump**

**Description**

Evaluate the influence function at selected times

**Usage**

```r
selectJump(IF, times, type)
```

**Arguments**

- **IF**: influence function returned by `iidCox`
- **times**: the times at which the influence function should be assessed
- **type**: can be "hazard" or/and "cumhazard".

**Value**

An object with the same dimensions as IF

**Author(s)**

Brice Ozenne broz@sund.ku.dk
### simActiveSurveillance

**Simulate data of a hypothetical active surveillance prostate cancer study**

**Description**

Simulate data of a hypothetical active surveillance prostate cancer study

**Usage**

```r
simActiveSurveillance(n)
```

**Arguments**

- `n` sample size

**Details**

This is based on the functionality of `library(lava)`.

**Value**

data table of size n

**Author(s)**

Thomas A. Gerds <tag@biostat.ku.dk>

**Examples**

```r
set.seed(71)
simActiveSurveillance(3)
```

---

### simMelanoma

**Simulate data alike the Melanoma data**

**Description**

Simulate data alike the Melanoma data

**Usage**

```r
simMelanoma(n)
```

**Arguments**

- `n` sample size
**Details**

This is based on the functionality of `library(lava)`.

**Value**

data table of size n

**Author(s)**

Thomas A. Gerds <tag@biostat.ku.dk>

**Examples**

```r
set.seed(71)
simMelanoma(3)
```

---

**simPBC**  
*simulating data alike the pbc data*

**Description**

This function can be used to simulate data alike the pbc data from the survival package.

**Usage**

`simPBC(n)`

**Arguments**

- `n`  
  Sample size

**Details**

using lava to synthesize data

**Value**

The simulated data.

**Author(s)**

Thomas A. Gerds <tag@biostat.ku.dk>
Examples

```r
library(survival)
library(lava)
# simulate data alike pbc data
set.seed(98)
d=simPBC(847)
d$protimegrp1 <- d$protimegrp=="10-11"
d$protimegrp2 <- d$protimegrp==">11"
d$sex <- factor(d$sex,levels=0:1,labels=c("m","f"))
sF1 <- survreg(Surv(time,status==1)~sex+age+logbili+protimegrp1+protimegrp2+stage3+stage4, data=d)
coxF1 <- coxph(Surv(time,status==1)~sex+age+logbili+protimegrp1+protimegrp2+stage3+stage4, data=d)
# load real pbc data
data(pbc,package="survival")
pbc <- na.omit(pbc[,c("time","status","age","sex","stage","bili","protime","trt")])
pbc$stage <- factor(pbc$stage)
levels(pbc$stage) <- list("1/2"=c(1,2),"3"=3,"4"=4)
pbc$logbili <- log(pbc$bili)
pbc$logprotime <- log(pbc$protime)
pbc$stage3 <- 1*(pbc$stage=="3")
pbc$stage4 <- 1*(pbc$stage=="4")
pbc$protimegrp <- cut(pbc$protime,c(-Inf,10,11,Inf),labels=c("<=10","10-11",">11"))
pbc$protimegrp1 <- pbc$protimegrp=="10-11"
pbc$protimegrp2 <- pbc$protimegrp==">11"
form1=Surv(time,status==1)~sex+age+logbili+protimegrp1+protimegrp2+stage3+stage4
F1 <- survival::survreg(form1,data=pbc)
form2=Surv(time,status==2)~sex+age+logbili+protimegrp1+protimegrp2+stage3+stage4
F2 <- survival::survreg(form2,data=pbc)
sF2 <- survreg(Surv(time,status==2)~sex+age+logbili+protimegrp1+protimegrp2+stage3+stage4, data=d)
G <- survreg(Surv(time,status==0)~sex+age+logbili+protimegrp1+protimegrp2+stage3+stage4, data=pbc)
sG <- survreg(Surv(time,status==0)~sex+age+logbili+protimegrp1+protimegrp2+stage3+stage4, data=d)
# compare fits in real and simulated pbc data
cbind(coef(F1),coef(sF1))
cbind(coef(F2),coef(sF2))
cbind(coef(G),coef(sG))
cbind(coef(glm(protimegrp1~age+sex+logbili,data=pbc,family="binomial")),
 coef(glm(protimegrp1~age+sex+logbili,data=d,family="binomial"))
cbind(coef(lm(logbili~age+sex,data=pbc)),coef(lm(logbili~age+sex,data=d)))
```
Usage

simsynth(object, n = 200, drop.latent = FALSE, ...)

Arguments

object generated with synthesize
n sample size
drop.latent if TRUE remove the latent event times from the resulting data set.
... additional arguments passed on to lava::sim

Examples

library(survival)
m=synthesize(Surv(time,status)~sex+age+bili,data=pbc)
simsynth(m,10,drop.latent=TRUE)

Description

TODO

Usage

SmcFcs(formula, data, m = 5, method, fitter = "glm", fit.formula, ...)

Arguments

formula TODO
data TODO
m TODO
method TODO
fitter TODO
fit.formula TODO
... TODO
# @export
Reconstruct each of the strata variables from the strata variable stored in the coxph object.

Usage

splitStrataVar(object)

Arguments

object a coxph object.

Author(s)

Brice Ozenne broz@sund.ku.dk and Thomas A. Gerds tag@biostat.ku.dk

Estimation of censoring probabilities at subject specific times

This function is used internally to construct pseudo values by inverse of the probability of censoring weights.

Usage

subjectWeights(
  formula,
  data,
  method = c("cox", "marginal", "km", "nonpar", "forest", "none"),
  args,
  lag = 1
)

Arguments

formula A survival formula like, Surv(time,status)-1 or Hist(time,status)-1 where status=0 means censored. The status variable is internally reversed for estimation of censoring rather than survival probabilities. Some of the available models, see argument model, will use predictors on the right hand side of the formula.
data The data used for fitting the censoring model
method Censoring model used for estimation of the (conditional) censoring distribution.
**subjectWeights**

**args**
Arguments passed to the fitter of the method.

**lag**
If equal to 1 then obtain $G(T_i - |X_i)$, if equal to 0 estimate the conditional censoring distribution at the subject.times, i.e. $(G(T_i |X_i))$.

**Details**
Inverse of the probability of censoring weights usually refer to the probabilities of not being censored at certain time points. These probabilities are also the values of the conditional survival function of the censoring time given covariates. The function subjectWeights estimates the conditional survival function of the censoring times and derives the weights.

IMPORTANT: the data set should be ordered, order(time,-status) in order to get the weights in the right order for some choices of method.

**Value**
- **times**: The times at which weights are estimated
- **weights**: Estimated weights at individual time values subject.times
- **lag**: The time lag.
- **fit**: The fitted censoring model
- **method**: The method for modelling the censoring distribution
- **call**: The call

**Author(s)**
Thomas A. Gerds <tag@biostat.ku.dk>

**Examples**

```r
library(prodlim)
library(survival)
dat=SimSurv(300)

dat <- dat[order(dat$time,-dat$status),]
# using the marginal Kaplan-Meier for the censoring times
WKM=subjectWeights(Hist(time,status)-X2,data=dat,method="marginal")
plot(WKM$fit)
WKM$fit
WKM$weights

# using the Cox model for the censoring times given X2
WCox=subjectWeights(Surv(time,status)-X2,data=dat,method="cox")
WCox
plot(WCox$weights,WKM$weights)

# using the stratified Kaplan-Meier for the censoring times given X2
```
subsetIndex <- subjectWeights(Surv(time,status)~X2,data=dat,method="nonpar")
plot(subsetIndex$fit,add=FALSE)

---

### subsetIndex

#### Extract Specific Elements From An Object

**Description**

Extract specific elements from an object.

**Usage**

```r
subsetIndex(object, index, default, ...)  
```

**Arguments**

- `object`: A vector or a matrix.
- `index`: Index of the elements to be extracted. 0 indicates that the column should be set to the default value. NA indicates that the column should be set to NA.
- `default`: The default value.
- `...`: Only used by the generic method.
- `col`: If object is a matrix, TRUE lead to extract the columns and FALSE the rows.

**Examples**

```r
M <- matrix(rnorm(50),5,10)
subsetIndex(M, index = c(0,0,1), default = 0)
subsetIndex(M, index = c(0,2,3,NA), default = 0)
subsetIndex(M, index = c(0,NA,2,3,NA), default = 0)

C <- 1:10
subsetIndex(C, index = c(0,0,1,5,NA), default = 0)
```
summary.ate

Summary Average Treatment Effects

Description

Summary average treatment effects.

Usage

```r
## S3 method for class 'ate'
summary(
  object,
  estimator = object$estimator[1],
  short = FALSE,
  type = c("meanRisk", "diffRisk"),
  se = FALSE,
  quantile = FALSE,
  estimate.boot = TRUE,
  digits = 3,
  ...)
```

Arguments

- `object`: object obtained with function `ate`
- `estimator` [character] The type of estimator relative to which the estimates should be displayed.
- `short` [logical] If TRUE, only displays the estimated risks.
- `type` [character vector] what to displayed. Can be "meanRisk" to display the risks specific to each treatment group, "diffRisk" to display the difference in risks between treatment groups, or "ratioRisk" to display the ratio of risks between treatment groups.
- `se` [logical] should the standard error of the risks be displayed?
- `quantile` [logical] should the quantile of the confidence bands be displayed?
- `estimate.boot` [logical] should the average estimate on the bootstrap samples be displayed?
- `digits` [integer, >0] Number of digits.
- `...` passed to `confint`

Details

to display confidence intervals/bands and p.value, the `confint` method needs to be applied on the object.
See Also

- `as.data.table` to extract the estimates in a data.table object.
- `autoplot.ate` for a graphical representation the standardized risks.
- `confint.ate` to compute p-values and adjusted p-values or perform statistical inference using a transformation.
- `confint.ate` to compute (pointwise/simultaneous) confidence intervals and (unadjusted/adjusted) p-values, possibly using a transformation.

---

### summary.FGR

**Summary of a Fine-Gray regression model**

**Description**

Summary of a Fine-Gray regression model

**Usage**

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

**Arguments**

- `object` Object fitted with function FGR
- `...` passed to cmprsk::summary.crr

---

### summary.riskRegression

**Summary of a risk regression model**

**Description**

Summary of a risk regression model

**Usage**

```r
## S3 method for class 'riskRegression'
summary(
  object,
  times,
  digits = 3,
  pvalue.digits = 4,
  eps = 10^-4,
  verbose = TRUE,
  ...
)
```
Summary of prediction performance metrics

Arguments

object Object obtained with ARR, LRR or riskRegression
times Time points at which to show time-dependent coefficients
digits Number of digits for all numbers but p-values
pvalue.digits Number of digits for p-values
eps p-values smaller than this number are shown as such
verbose Level of verbosity

Usage

## S3 method for class 'Score'
summary(
  object,
  times,
  what = c("score", "contrasts"),
  models,
  digits = 1,
  pvalue.digits = 4,
  ...
)

Arguments

times Select time points
what Either "score", "contrasts" or both, i.e., c("score", "contrasts")
models Select which models to summarize. Need to be a subset of object$models
digits For rounding everything but p-values
pvalue.digits For rounding p-values

Details

The AUC and the Brier score are put into tables
SuperPredictor

Value
List of tables

Author(s)
Thomas A. Gerds <tag@biostat.ku.dk>

See Also
Score

SuperPredictor  

Formula interface for SuperLearner::SuperLearner

Description
Formula interface for SuperLearner::SuperLearner

Usage
SuperPredictor(
  formula,
  data,
  family = "binomial",
  SL.library = c("SL.glm", "SL.glm.interaction", "SL.ranger"),
  ...
)

Arguments
formula  where the left hand side specifies the outcome and the right hand side the predictors
data  data set in which formula can be evaluated
family  the outcome family. default is binomial
SL.library  the SuperLearner libraries
...  passed to SuperLearner::SuperLearner

Details
Formula interface for SuperLearner::SuperLearner
##' @param formula
SurvResponseVar

Extract the time and event variable from a Cox model

Description

Extract the time and event variable from a Cox model

Usage

SurvResponseVar(formula)

Arguments

formula a formula

Author(s)

Brice Ozenne broz@sund.ku.dk

Examples

## Not run:
SurvResponseVar(Surv(time, event) ~ X1 + X2)
SurvResponseVar(Hist(time, event == 0) ~ X1 + X2)
SurvResponseVar(Surv(start, time, status, type = "counting") ~ X3 + X5)
SurvResponseVar(Surv(start, event = status, time2 = time, type = "counting") ~ X3 + X5)
SurvResponseVar(survival::Surv(start, event = status, time2 = time, type = "counting") ~ X3 + X5)
SurvResponseVar(I(status == 1) ~ X3 + X5)
SurvResponseVar(list(Hist(time, event) ~ X1 + X6, Hist(time, event) ~ X6))

## End(Not run)
synthesize

Cooking and synthesizing survival data

Description

Fit parametric regression models to the outcome distribution and optionally also parametric regression models for the joint distribution of the predictors structural equation models. Then the function sim.synth can be called on the resulting object to to simulate from the parametric model based on the machinery of the lava package.

Usage

synthesize(object, data, ...)

## S3 method for class 'formula'
synthesize(
  object,
  data,
  recursive = FALSE,
  max.levels = 10,
  verbose = FALSE,
  ...
)

## S3 method for class 'lvm'
synthesize(
  object,
  data,
  max.levels = 10,
  logtrans = NULL,
  verbose = FALSE,
  fix.names = FALSE,
  ...
)

Arguments

object Specification of the synthesizing model structures. Either a formula or a lvm object. See examples.
data Data to be synthesized.
... Not used yet.
recursive Let covariates recursively depend on each other.
max.levels Integer used to guess which variables are categorical. When set to 10, the default, variables with less than 10 unique values in data are treated as categorical.
verbose Logical. If TRUE then more messages and warnings are provided.
**synthesize**

- **logtrans**  Vector of covariate names that should be log-transformed. This is primarily for internal use.
- **fix.names**  Fix possible problematic covariate names.

**Details**

Synthesizes survival data (also works for linear models and generalized linear models). The idea is to be able to simulate new data sets that mimic the original data. See the vignette `vignette("synthesize",package = "riskRegression")` for more details.

The simulation engine is: lava.

**Value**

lava object

**Author(s)**

Thomas A. Gerds <tag@biostat.ku.dk>

**See Also**

lvm

**Examples**

```r
# pbc data
library(survival)
library(lava)
data(pbc)
pbc <- na.omit(pbc[,c("time","status","sex","age","bili")])
pbc$logbili <- log(pbc$bili)
v_synt <- synthesize(object=Surv(time,status)~sex+age+logbili,data=pbc)
d <- simsynth(v_synt,1000)
fit_sim <- coxph(Surv(time,status==1)~age+sex+logbili,data=d)
fit_real <- coxph(Surv(time,status==1)~age+sex+logbili,data=pbc)
# compare estimated log-hazard ratios between simulated and real data
cbind(coef(fit_sim),coef(fit_real))

u <- lvm()
distribution(u,~sex) <- binomial.lvm()
distribution(u,~age) <- normal.lvm()
distribution(u,~trt) <- binomial.lvm()
distribution(u,~logbili) <- normal.lvm()
u <-eventTime(u,time=min(time.cens=0,time.transplant=1,time.death=2), "status")
lava::regression(u,logbili~age+sex) <- 1
lava::regression(u,time.transplant~sex+age+logbili) <- 1
lava::regression(u,time.death~sex+age+logbili) <- 1
lava::regression(u,time.cens~1) <- 1
transform(u,logbili-bili) <- function(x){log(x)}
u_synt <- synthesize(object=u, data=na.omit(pbc))
set.seed(8)
```
d <- simsynth(u_synt,n=1000)
# note: synthesize may relabel status variable
fit_sim <- coxph(Surv(time,status==1)-age+sex+logbili,data=d)
fit_real <- coxph(Surv(time,status==1)-age+sex+log(bili),data=pbc)
# compare estimated log-hazard ratios between simulated and real data
cbind(coef(fit_sim),coef(fit_real))

# Cancer data
#
data(cancer)
b <- lvm()
distribution(b,~rx) <- binomial.lvm()
distribution(b,~age) <- normal.lvm()
distribution(b,~resid.ds) <- binomial.lvm()
distribution(b,~ecog.ps) <- binomial.lvm()
lava::regression(b,time.death~age+rx+resid.ds) <- 1
b<-eventTime(b,futime=min(time.cens=0,time.death=1), "fustat")
b_synt <- synthesize(object = b, data = ovarian)
D <- simsynth(b_synt,1000)
fit_real <- coxph(Surv(futime,fustat)-age+rx+resid.ds, data=ovarian)
fit_sim <- coxph(Surv(futime,fustat)-age+rx+resid.ds, data=D)
cbind(coef(fit_sim),coef(fit_real))
w_synt <- synthesize(object=Surv(futime,fustat)-age+rx+resid.ds, data=ovarian)
D <- simsynth(w_synt,1000)
fit_sim <- coxph(Surv(futime,fustat==1)-age+rx+resid.ds, data=D)
fit_real <- coxph(Surv(futime,fustat==1)-age+rx+resid.ds, data=ovarian)
# compare estimated log-hazard ratios between simulated and real data
cbind(coef(fit_sim),coef(fit_real))

---

terms.phreg

### Extract terms for phreg objects

**Description**

Extract terms for phreg objects

**Usage**

```r
## S3 method for class 'phreg'
terms(x, ...)
```

**Arguments**

- `x`: a phreg object.
- `...`: not used.
transformCIBP

Compute Confidence Intervals/Bands and P-values After a Transformation

Description

Compute confidence intervals/bands and p-values after a transformation

Usage

transformCIBP(
  estimate,  # numeric matrix] the estimate value before transformation.
  se,        # numeric matrix] the standard error before transformation.
  iid,       # numeric array] the iid decomposition before transformation.
  null,      # numeric] the value of the estimate (before transformation) under the null hypothesis.
  conf.level, # numeric, 0-1] Level of confidence.
  alternative, # character] a character string specifying the alternative hypothesis, must be one of "two.sided" (default), "greater" or "less".
  ci,        # logical] should confidence intervals be computed.
  type,      # character] the transformation. Can be "log", "loglog", "cloglog", or "atanh" (Fisher transform), or "atanh2" (modified Fisher transform for [0-1] variable).
  min.value, # numeric] if not NULL and the lower bound of the confidence interval is below min, it will be set at min.
  max.value, # numeric]
  band,      # numeric]
  method.band, # numeric]
  n.sim,     # numeric]
  seed,      # numeric]
  p.value,   # numeric]
  df = NULL  # numeric]
)
max.value [numeric] if not NULL and the lower bound of the confidence interval is below max, it will be set at max.

band [integer 0,1,2] When non-0, the confidence bands are computed for each contrasts (band=1) or over all contrasts (band=2).

method.band [character] method used to adjust for multiple comparisons. Can be any element of p.adjust.methods (e.g. "holm"), "maxT-integration", or "maxT-simulation".

n.sim [integer, >0] the number of simulations used to compute the quantiles for the confidence bands.

seed [integer, >0] seed number set before performing simulations for the confidence bands.

p.value [logical] should p-values and adjusted p-values be computed. Only active if ci=TRUE or band>0.

df [integer, >0] optional. Degrees of freedom used for the student distribution of the test statistic. If not specified, use a normal distribution instead.

Details

The iid decomposition must have dimensions [n.obs,time,n.prediction] while estimate and se must have dimensions [n.prediction,time].

Single step max adjustment for multiple comparisons, i.e. accounting for the correlation between the test statistics but not for the ordering of the tests, can be performed setting the argument method.band to "maxT-integration" or "maxT-simulation". The former uses numerical integration (pmvnorm and qmvnorm to perform the adjustment while the latter using simulation. Both assume that the test statistics are jointly normally distributed.

Examples

```r
set.seed(10)
n <- 100X <- rnorm(n)

res2sided <- transformCIBP(estimate = mean(X), se = cbind(sd(X)/sqrt(n)), null = 0,
                           type = "none", ci = TRUE, conf.level = 0.95, alternative = "two.sided",
                           min.value = NULL, max.value = NULL, band = FALSE,
                           p.value = TRUE, seed = 10, df = n-1)

resLess <- transformCIBP(estimate = mean(X), se = cbind(sd(X)/sqrt(n)), null = 0,
                         type = "none", ci = TRUE, conf.level = 0.95, alternative = "less",
                         min.value = NULL, max.value = NULL, band = FALSE,
                         p.value = TRUE, seed = 10, df = n-1)

resGreater <- transformCIBP(estimate = mean(X), se = cbind(sd(X)/sqrt(n)), null = 0,
                          type = "none", ci = TRUE, conf.level = 0.95, alternative = "greater",
                          min.value = NULL, max.value = NULL, band = FALSE,
                          p.value = TRUE, seed = 10, df = n-1)

## comparison with t-test
GS <- t.test(X, alternative = "two.sided")
```
wglm

Logistic Regression Using IPCW

Description
Logistic regression over multiple timepoints where right-censoring is handled using inverse probability of censoring weighting (IPCW).

Usage
wglm(
  regressor.event,
  formula.censor,
  times,
  data,
  cause = NA,
  fitter = "coxph",
  product.limit = FALSE
)

Arguments
  regressor.event
    [formula] a formula with empty left hand side and the covariates for the logistic regression on the right hand side.
  formula.censor
    [formula] a formula used to fit the censoring model.
  times
    [numeric vector] time points at which to model the probability of experiencing an event.
  data
    [data.frame] dataset containing the time at which the event occurred, the type of event, and regressors used to fit the censoring and logistic models.
  cause
    [character or numeric] the cause of interest. Defaults to the first cause.
  fitter
    [character] routine to fit the Cox regression models.
  product.limit
    [logical] if TRUE the survival is computed using the product limit estimator.
Details

First, a Cox model is fitted (argument formula.censor) and the censoring probabilities are computed relative to each timepoint (argument times) to obtain the censoring weights. Then, for each timepoint, a logistic regression is fitted with the appropriate censoring weights and where the outcome is the indicator of having experience the event of interest (argument cause) at or before the timepoint.

Value

an object of class "wglm".

Examples

library(survival)

set.seed(10)
n <- 250
tau <- 1:5
d <- sampleData(n, outcome = "competing.risks")
dFull <- d[event!=0]  ## remove censoring
dSurv <- d[event!=2]  ## remove competing risk

##### no censoring #####
e.wglm <- wglm(regressor.event = ~ X1, formula.censor = Surv(time,event==0) ~ 1,
                 times = tau, data = dFull, product.limit = TRUE)
e.wglm  ## same as a logistic regression

summary(ate(e.wglm, data = dFull, times = tau, treatment = "X1", verbose = FALSE))

##### right-censoring #####
## no covariate in the censoring model (independent censoring)
e.C.wglm <- wglm(regressor.event = ~ X1, formula.censor = Surv(time,event==0) ~ 1,
                  times = tau, data = dSurv, product.limit = TRUE)
e.C.wglm

## with covariates in the censoring model
e.C2.wglm <- wglm(regressor.event = ~ X1 + X8,
                   formula.censor = Surv(time,event==0) ~ X1*X8,
                   times = tau, data = dSurv)
e.C2.wglm

##### Competing risks #####
## here Kaplan-Meier as censoring model
e.CR.wglm <- wglm(regressor.event = ~ X1,
                   formula.censor = Surv(time,event==0) ~ strata(X1),
                   times = tau, data = d, cause = 1, product.limit = TRUE)
e.CR.wglm
Index

* datasets
  Melanoma, 73
  Paquid, 75

* survival
  Cforest, 32
  CSC, 52
  FGR, 56
  ipcw, 70
  plot.riskRegression, 77
  plotEffects, 84
  plotPredictRisk, 86
  predict.riskRegression, 96
  predictRisk, 103
  riskRegression, 116
  selectCox, 136
  subjectWeights, 142

anova.ate, 4
ARR (riskRegression), 116
as.data.table, 12, 146
as.data.table.ate, 7
as.data.table.influenceTest, 8
as.data.table.predictCox, 8
as.data.table.predictCSC, 9
ate, 9, 17, 109
autoplot.ate, 12, 15, 146
autoplot.predictCox, 18, 99
autoplot.predictCSC, 21, 95
autoplot.Score, 23
axis, 88

calcSeCox, 29
calcSeCSC, 31
Cforest, 32
coef.CauseSpecificCox, 33
coef.riskRegression, 34
colCenter_cpp, 34
colCumSum, 35
colMultiply_cpp, 35
colScale_cpp, 36
confint.ate, 12, 37, 109, 146
confint.influenceTest, 40, 110
confint.predictCox, 41, 99, 111
confint.predictCSC, 42, 95, 112
coxBaseEstimator, 44
coxCenter, 45
coxFormula, 45
coxLP, 46
coxModelFrame, 47
coxN, 47
coxph, 53
coxSpecial, 48
coxStrata, 49
coxStrataLevel, 50
coxVarCov, 51
coxVariableName, 51
CSC, 32, 52
Ctree, 55
discreteRoot, 56
fastbw, 137
FGR, 56
getSplitMethod, 58
GLMnet, 60

Hal9001, 61
Hist, 57

iid.wglm, 61
iidCox, 62
influenceTest, 65, 110
information.wglm, 67
IPA, 68
ipcw, 70
legend, 88  
lines, 83  
LRR (riskRegression), 116  
  
Melanoma, 73  
model.frame, 83  
model.matrix.cph, 74  
model.matrix.phreg, 74  
  
Paquid, 75  
penalizedS3, 76  
plot, 88  
plot рискRegression, 77  
plotAUC, 78  
plotBrier, 80  
plotCalibration, 81  
plotEffects, 84  
plotPredictRisk, 86  
plotRisk, 88, 89  
plotROC, 91  
predict.CauseSpecificCox, 22, 93, 112  
predict.FGR, 96  
predict.riskRegression, 96  
predictBig.CauseSpecificCox  
  (predict.CauseSpecificCox), 93  
predictCox, 19, 97, 101, 111  
predictCoxPL, 101  
predictRisk, 60, 61, 103, 103  
predict.ate, 108  
predict.CauseSpecificCox, 109  
predict.FGR, 109  
predict.influenceTest, 110  
predict.IPA, 110  
predict.predictCox, 111  
predict.predictCSC, 111  
predict.riskRegression, 112  
predict.Score, 113  
predict.subjectWeights, 113  
reconstructData, 114  
riskLevelPlot, 114  
riskRegression, 57, 116  
riskRegression.options, 119  
rowCenter.cpp, 120  
rowCumSum, 121  
rowMultiply.cpp, 121  
rowPaste, 122  
rowScale.cpp, 122  
rowSumsCrossprod, 123  
  
r_squared (IPA), 68  
sampleData, 124  
sampleDataTD (sampleData), 124  
saveCoxConfidential, 125  
Score, 126  
score.wglm, 136  
selectCox, 136  
selectJump, 137  
simActiveSurveillance, 138  
simMelanoma, 138  
simPBC, 139  
simsynth, 140  
SmartControl, 78, 81, 83, 86, 88, 90, 92  
SmCfs, 141  
splitStrataVar, 142  
subjectWeights, 142  
subsetIndex, 144  
summary.ate, 12, 109, 145  
summary.FGR, 146  
summary.riskRegression, 146  
summary.Score, 147  
SuperPredictor, 148  
SurvResponseVar, 149  
synthesize, 150  
  
terms.phreg, 152  
transformCIBP, 153  
  
wglm, 155  
  