

Package ‘marginalizedRisk’

January 28, 2024

LazyLoad yes

Version 2024.1-27

Title Estimating Marginalized Risk

Depends R (>= 4.0)

Imports

Suggests RUnit, R.rsp, survival

Description Estimates risk as a function of a marker by integrating over other covariates in a conditional risk model.

VignetteBuilder R.rsp

License GPL (>= 2)

NeedsCompilation no

Author Youyi Fong [cre],
Peter Gilbert [aut],
Marco Carone [aut]

Maintainer Youyi Fong <youyifong@gmail.com>

Repository CRAN

Date/Publication 2024-01-28 04:30:02 UTC

R topics documented:

bias.factor	2
marginalized.risk	3
marginalized.risk.threshold	5

Index	7
--------------	----------

`bias.factor`*E-value and Controlled Risk Curve*

Description

Compute E-values (Equation 4 in Gilbert et al.) and controlled risk curve bias factor (Equation 6 in Gilbert et al.).

Usage

```
bias.factor(RRud, RReu)
```

```
E.value(rr)
```

```
controlled.risk.bias.factor(ss, s.cent, s1, s2, RRud)
```

Arguments

RRud	RRud
RReu	RReu
rr	rr
ss	A vector of marker values
s.cent	Central marker value
s1	s1 and s2 are a pair of marker values for which we set a RRud.
s2	s2

Details

These three functions constitute an implementation of the core functionality in Gilbert et al. (2020).

For examples on how to use these functions, see the code for Gilbert et al. at <https://github.com/youyifong/CoPveryhighVE>

Value

`controlled.risk.bias.factor` returns a vector of bias factors corresponding to the vector of marker values in `ss`.

References

Gilbert, Fong, Carone (2020) Assessment of Immune Correlates of Protection via Controlled Risk of Vaccine Recipients

marginalized.risk	<i>Compute Marginalized Risk</i>
-------------------	----------------------------------

Description

Computes risk of disease as a function of marker s by marginalizing over a covariate vector Z .

Usage

```
marginalized.risk(fit.risk, marker.name, data, categorical.s, weights =
  rep(1, nrow(data)), t = NULL, ss = NULL, verbose =
  FALSE, t.end = NULL)
```

```
marginalized.risk.cont(fit.risk, marker.name, data,
  weights = rep(1, nrow(data)), t=NULL, ss = NULL, verbose = FALSE)
```

```
marginalized.risk.cont.2(fit.risk, marker.name, data,
  weights=rep(1, nrow(data)), t, ss, marker.name.2, s.2, verbose=FALSE)
```

```
marginalized.risk.cat(fit.risk, marker.name, data, weights = rep(1,
  nrow(data)), t = NULL, verbose = FALSE, t.end = NULL)
```

Arguments

fit.risk	A regression object where the outcome is risk of disease, e.g. $y \sim Z + \text{marker}$. Need to support <code>predict(fit.risk)</code>
marker.name	string
marker.name.2	string
data	A data frame containing the phase 2 data
ss	A vector of marker values
s.2	s.2
weights	Inverse prob sampling weight, optional
t	If fit.risk is Cox regression, t is the time at which distribution function will be assessed
t.end	t.end
categorical.s	TRUE if the marker is categorical, FALSE otherwise
verbose	Boolean

Details

See the vignette file for more details.

Value

If `ss` is not `NULL`, a vector of probabilities are returned. If `ss` is `NULL`, a matrix of two columns are returned, where the first column is the marker value and the second column is the probabilities.

Examples

```
#### suppose wt.loss is the marker of interest

if(requireNamespace("survival")) {

library(survival)

dat=subset(lung, !is.na(wt.loss) & !is.na(ph.ecog))

f1=Surv(time, status) ~ wt.loss + ph.ecog + age + sex

fit.risk = coxph(f1, data=dat)

ss=quantile(dat$wt.loss, seq(.05,.95,by=0.01))
t0=1000
prob = marginalized.risk(fit.risk, "wt.loss", dat, categorical.s=FALSE, t = t0, ss=ss)

plot(ss, prob, type="l", xlab="Weight loss", ylab=paste0("Probability of survival at day ", t0))

}

## Not run:

#### Efron bootstrap to get confidence band

# store the current rng state
save.seed <- try(get(".Random.seed", .GlobalEnv), silent=TRUE)
if (class(save.seed)=="try-error") {set.seed(1); save.seed <- get(".Random.seed", .GlobalEnv) }

B=10 # bootstrap replicates, 1000 is good
numCores=1 # multiple cores can speed things up
library(doParallel)
out=mclapply(1:B, mc.cores = numCores, FUN=function(seed) {
  set.seed(seed)
  # a simple resampling scheme here. needs to be adapted to the sampling scheme
  dat.tmp=dat[sample(row(dat), replace=TRUE),]
  fit.risk = coxph(f1, data=dat)
  marginalized.risk(fit.risk, "wt.loss", dat.tmp, categorical.s=FALSE, t = t0, ss=ss)
})
res=do.call(cbind, out)

# restore rng state
assign(".Random.seed", save.seed, .GlobalEnv)
```

```
# quantile bootstrap CI
ci.band=t(apply(res, 1, function(x) quantile(x, c(.025,.975))))

plot(ss, prob, type="l", xlab="Weight loss", ylab=paste0("Probability of survival at day ", t0),
      ylim=range(ci.band))
lines(ss, ci.band[,1], lty=2)
lines(ss, ci.band[,2], lty=2)

## End(Not run)
```

```
marginalized.risk.threshold
```

Compute Marginalized Risk as a Function of $S \geq s$

Description

Computes risk of disease conditional on $S \geq s$ by marginalizedizing over a covariate vector Z .

Usage

```
marginalized.risk.threshold(formula, marker.name, data, weights=rep(1, nrow(data)),
  t, ss=NULL, verbose=FALSE)
```

Arguments

formula	A formula for coxph
marker.name	string
data	A data frame containing the phase 2 data
ss	A vector of marker values
weights	Inverse prob sampling weight, optional
t	t is the time at which survival will be assessed
verbose	Boolean

Details

See the vignette file for more details.

Value

If `ss` is not `NULL`, a vector of probabilities are returned. If `ss` is `NULL`, a matrix of two columns are returned, where the first column is the marker value and the second column is the probabilities.

Examples

```
#### suppose wt.loss is the marker of interest

if(requireNamespace("survival")) {

  library(survival)

  dat=subset(lung, !is.na(wt.loss) & !is.na(ph.ecog))
  f1=Surv(time, status) ~ ph.ecog + age + sex
  ss=quantile(dat$wt.loss, seq(.05,.95,by=0.01))
  t0=1000
  prob = marginalized.risk.threshold(f1, "wt.loss", dat, t = t0, ss=ss)

  plot(ss, prob, type="l", xlab="Weight loss (S>=s)",
        ylab=paste0("Probability of survival at day ", t0))

}
```

Index

`bias.factor`, [2](#)

`controlled.risk.bias.factor`
`(bias.factor)`, [2](#)

`E.value(bias.factor)`, [2](#)

`marginalized.risk`, [3](#)

`marginalized.risk.threshold`, [5](#)

`marginalizedRisk(marginalized.risk)`, [3](#)