The proptest Package

September 29, 2006

Type Package

Title Tests of the Proportional Hazards Assumption

Version 0.1-1

Date 2006-09-25

Author David Kraus <kraus@karlin.mff.cuni.cz>

Maintainer David Kraus <kraus@karlin.mff.cuni.cz>

Depends survival

Description Tests of the proportional hazards assumption in the Cox model: data-driven Neyman type smooth tests and score process based tests for identifying nonproportional covariates.

License GPL

URL http://www.davidkraus.net/proptest/

R topics documented:

allsubsets .............................................................. 1
mini.survival ........................................................... 2
plot.scoreproptest .................................................... 3
proptest-package ...................................................... 4
scoreproptest .......................................................... 6
smoothproptest ........................................................ 8

Index 11
### allsubsets  
**All Subsets**

**Description**

The function generates all subsets of the set \{1, \ldots, n\}.

**Usage**

\[ \text{allsubsets}(n) \]

**Arguments**

- **n**
  
  the number of elements of the set whose subsets are to be returned.

**Value**

Returns a \(2^n\) by \(n\) matrix of logical values. Each row corresponds to one subset.

**Note**

This function is used in the all subsets variant of the data-driven smooth tests of the proportional hazards assumption, see `smoothproptest`.

**Author(s)**

David Kraus, [http://www.davidkraus.net/](http://www.davidkraus.net/)

**Examples**

\[ \text{allsubsets}(4) \]

---

### mini.survival  
**Minimalist Versions of Functions from the Package survival**

**Description**

Minimalist (lightweight) versions of some functions from the package `survival` used in `smoothproptest` and `scoreproptest`. For internal use only.

**Usage**

- `mini.agreg.fit(x, y, init, control, sort.start, sort.end)`
- `mini.coxph.detail(object)`
- `mini.survfit.coxph(object)`
Arguments

x  
covariate matrix.

y  
response (matrix with three columns: start time, stop time, event indicator).

init  
initial estimates.

control  
control options (an object of class coxph.control).

sort.start, sort.end  
order.

object  
a Cox model fit.

Details

These functions are minimalist versions of the functions agreg.fit.coxph.detail and survfit.coxph extracted from the package survival. They do not perform error checks and do not work with data frames, model frames etc., which makes them much faster. They are used in smoothproptest and scoreproptest.

Warning

These functions are for internal use only.

Author(s)

David Kraus, http://www.davidkraus.net/

plot.scoreproptest  Plotting the Observed Score Process and its Simulations

Description

The function plots the observed score process and a number of its realisations simulated under the hypothesis of proportional hazards.

Usage

plot.scoreproptest(x, nsim.plot = x$nsim.plot, ...)

Arguments

x  
an object of class "scoreproptest" (output of scoreproptest).

nsim.plot  
the number of simulated paths of the score process to be plotted. It must not be greater than x$nsim.plot.

...  
further plotting parameters.

Details

By plotting the observed path of the score process along with its simulations, one can visually assess the time-constancy of the effect of the corresponding covariate.

The function plots x$score.process and the first nsim.plot realisations contained in x$score.process.sim.
Author(s)

David Kraus, http://www.davidkraus.net/

References


See Also

`scoreproptest`

Examples

```r
## Case 4 of Kvaloy & Neef (2004, Lifetime Data Anal.):
## data generated from the distribution with hazard rate
## \lambda(t)=\exp(0.5tZ_1+Z_2-8)
## (Z_1,Z_2) jointly normal with \mu=4, \sigma=1, \rho=rho
## censoring times uniform(0,5)

n = 200
rho = .3
z = matrix(rnorm(n*2),ncol=2) %*% chol(matrix(c(1,rho,rho,1),2)) + 4
a = .5
tim = 1/(a*z[,1]) * log(1-a*z[,1]*exp(-z[,2]+8)*log(runif(n)))
ct = 5*runif(n)
nc = tim<=ct
tim = pmin(tim,ct)
fit = coxph(Surv(tim,nc)~z)
par(mfrow=c(2,1))
test1 = scoreproptest(fit,covariate=1) # testing Z_1 (nonproportional)
print(test1)
plot(test1,main="Score process for z1")
test2 = scoreproptest(fit,covariate=2) # testing Z_2 (proportional)
print(test2)
plot(test2,main="Score process for z2")
par(mfrow=c(1,1))
```

---

**Description**

Tests of the proportional hazards assumption in the Cox model: data-driven Neyman type smooth tests and score process based tests for identifying nonproportional covariates.
Details

Package: proptest
Type: Package
Version: 0.1-1
Date: 2006-09-25
License: GPL

The package provides two functions for testing proportional hazards: smoothproptest for smooth tests, and scoreproptest for tests based on the score process.

Smooth tests consist of expressing the coefficient of the tested covariate as a linear combination of some basis functions, i.e., they consist of introducing artificial time-dependent covariates and testing their significance. A data-driven choice of these covariates is implemented.

Score process tests are tests based on functionals (Kolmogorov–Smirnov, Cramer–von Mises, Anderson–Darling) of components of the score process. Both numerical and visual assessment based on simulations is possible.

Both of the tests may be used for testing individual covariates because the covariates that are not tested may be modeled by smooth functions.

Author(s)

David Kraus, http://www.davidkraus.net/

References


Examples

## Case 4 of Kvaloy & Neef (2004, Lifetime Data Anal.):
## data generated from the distribution with hazard rate
## \( \lambda(t) = \exp(0.5tZ_1 + Z_2 - 8) \)
## \((Z_1, Z_2)\) jointly normal with \( E=4, \ var=1, \ cor=rho \)
## censoring times uniform(0,5)

n = 200
rho = .3
z = matrix(rnorm(n*2),ncol=2) %*% chol(matrix(c(1,rho,rho,1),2)) + 4
a = .5
tim = 1/(a*z[,1]) * log(1-a*z[,1]*exp(-z[,2]+8)*log(runif(n)))
ct = 5*runif(n)
nc = tim<=ct
tim = pmin(tim,ct)
fit = coxph(Surv(tim,nc)~z)

## Data-driven smooth tests

test1 = smoothproptest(fit,covariate=1) # testing Z_1 (nonproportional)
### scoreproptest

**Test of the Proportional Hazards Assumption Based on the Score Process**

**Description**

The function performs tests of the proportional hazards assumption for an individual covariate in the Cox model for right-censored survival data. The tests of the Kolmogorov–Smirnov, Cramer–von Mises and Anderson–Darling type based on the component of the score process are computed and simulated \( p \)-values are returned.

**Usage**

```r
cscoreproptest(fit, covariate = 1, dims = 4, basis = "legendre", time.transf = "F", nsim = 1000, nsim.plot = 50, weight = "unit")
```

**Arguments**

- `fit`: a Cox model fit (an output of `coxph`).
- `covariate`: integer determining which covariate is to be tested for proportionality.
- `dims`: a vector or a single value. `dims` gives dimensions for smooth modeling of the effects of the covariates that are not tested. If `dims` is a single value and there is more than one covariate, the value is replicated.
- `basis`: a character string. Which basis of smooth functions is to be used? Possible values are "legendre" and "cos" (or "cosine").
- `time.transf`: a character string. The basis functions are evaluated at transformed times. With `time.transf="F"`, the transformation is \( F_0(t)/F_0(\tau) \) (\( F_0 \) is the distribution function corresponding to the baseline hazard). For `time.transf="L"`, the transformation is \( \Lambda_0(t)/\Lambda_0(\tau) \) (\( \Lambda_0 \) is the cumulative baseline hazard). \( F_0 \) and \( \Lambda_0 \) are estimated from the input model `fit`.
- `nsim`: the number of simulations to be carried out to approximate the \( p \)-value.
The score process is used for assessment of the proportional hazards assumption (Lin, Wei and Ying, 1993). Each component of the score process reflects departures from proportionality (time-constancy of the effect) of the corresponding covariate.

However, tests based on individual components of the process are generally not capable to distinguish which covariate is proportional and which not. The method is only valid if the other covariates are proportional.

Therefore, the potentially time-varying effects of the covariates that are not tested are modeled as combinations of basis functions. The test is then based on the score process from this large model with artificial time-dependent covariates. This makes it possible to perform individual covariate tests. See Kraus (2006).

The vector `dims` gives the number of the basis functions for each covariate.

Details

The score process is used for assessment of the proportional hazards assumption (Lin, Wei and Ying, 1993). Each component of the score process reflects departures from proportionality (time-constancy of the effect) of the corresponding covariate.

However, tests based on individual components of the process are generally not capable to distinguish which covariate is proportional and which not. The method is only valid if the other covariates are proportional.

Therefore, the potentially time-varying effects of the covariates that are not tested are modeled as combinations of basis functions. The test is then based on the score process from this large model with artificial time-dependent covariates. This makes it possible to perform individual covariate tests. See Kraus (2006).

The vector `dims` gives the number of the basis functions for each covariate.

Value

A list (an object of class "scoreproptest"). The most important components are:

- `score.process` the component of the score process corresponding to the tested covariate.
- `time` the vector of the event times.
- `stat.ks` the Kolmogorov–Smirnov test statistic.
- `p.ks` the simulated p-value for the Kolmogorov–Smirnov test.
- `stat.cm` the Cramer–von Mises test statistic.
- `p.cm` the simulated p-value for the Cramer–von Mises test.
- `stat.ad` the Anderson–Darling test statistic.
- `p.ad` the simulated p-value for the Anderson–Darling test.
- `score.process.sim` a matrix with `nsim.plot` columns containing simulated paths of the score process. These may be plotted with `plot.scoreproptest`.

Author(s)

David Kraus, http://www.davidkraus.net/

References


See Also

plot.scoreproptest, smoothproptest, coxph

Examples

```r
## Case 4 of Kvaloy & Neef (2004, Lifetime Data Anal.):
## data generated from the distribution with hazard rate
## \( \lambda(t) = \exp(0.5tZ_1 + Z_2 - 8) \)
## \( (Z_1, Z_2) \) jointly normal with \( E=4, \text{var}=1, \text{cor}=\rho \)
## censoring times uniform(0,5)

n = 200
rho = 0.3
z = matrix(rnorm(n*2), ncol=2) %*% chol(matrix(c(1,rho,rho,1),2)) + 4
a = 0.5
tim = 1/(a*z[,1]) * log(1-a*z[,1]%*%exp(-z[,2]+8)*log(runif(n)))
ct = 5*runif(n)
nc = tim<=ct
tim = pmin(tim,ct)
fit = coxph(Surv(tim,nc)~z)

par(mfrow=c(2,1))
test1 = scoreproptest(fit, covariate=1) # testing Z_1 (nonproportional)
print(test1)
plot(test1, main="Score process for z1")
test2 = scoreproptest(fit, covariate=2) # testing Z_2 (proportional)
print(test2)
plot(test2, main="Score process for z2")
par(mfrow=c(1,1))
```

smoothproptest

Data-driven Smooth Test of the Proportional Hazards Assumption

Description

The function performs the Neyman type smooth test of the proportional hazards assumption for an individual covariate in the Cox model for right censored survival data. Both a fixed and data-driven choice of the alternative model is possible.

Usage

```r
smoothproptest(fit, covariate = 1, dims = 4, basis = "legendre",
time.transf = "F", data.driven = TRUE,
all.subsets = FALSE, h.approx = TRUE, sim = FALSE,
nsim = 1000)
```
smoothproptest

Arguments

fit 
  a Cox model fit (an output of `coxph`).
covariate 
  integer determining which covariate is to be tested for proportionality.
dims 
  a vector or a single value. `dims` gives dimensions for smooth modeling of the effects of the covariates that are not tested, and of the tested covariate. If `dims` is a single value and there is more than one covariate, the value is replicated.
basis 
  a character string. Which basis of smooth functions is to be used? Possible values are "legendre" and "cos" (or "cosine").
time.transf 
  a character string. The basis functions are evaluated at transformed times. With `time.transf="F"`, the transformation is \( F_0(t)/F_0(\tau) \) (\( F_0 \) is the distribution function corresponding to the baseline hazard). For `time.transf="L"`, the transformation is \( \Lambda_0(t)/\Lambda_0(\tau) \) (\( \Lambda_0 \) is the cumulative baseline hazard). \( F_0 \) and \( \Lambda_0 \) are estimated.
data.driven 
  logical. Should the BIC be used?
all.subsets 
  logical. If `TRUE` then the BIC selects out of all the nonempty subsets, otherwise only the nested subsets are used.
h.approx 
  logical. Should we compute the \( p \)-value using the two term \( H \)-approximation?
sim 
  logical. Should we compute the \( p \)-value using the LWY simulation approximation?
nsim 
  the number of simulations to be carried out to approximate the \( p \)-value.

Details

The Neyman type smooth test of proportionality (time-constancy of the coefficient) of a covariate against the alternative of the time-varying coefficient consists of expressing the coefficient of the tested covariate as a linear combination of basis functions and testing significance of the new artificial time-dependent covariates using the partial likelihood score test.

In the data-driven version, the alternative is selected by a BIC-like rule. The distribution of the test statistic then may be approximated by the two term \( H \)-approximation (Kraus, 2005) or by simulations (Lin, Wei and Ying, 1993).

The potentially time-varying effects of the covariates that are not tested should be modeled as combinations of basis functions too. This makes it possible to perform individual covariate tests. Not doing so would be dangerous: the test generally would not be capable to distinguish which covariate is proportional and which not. See Kraus (2006).

The vector `dims` gives the number of the basis functions for each covariate.

Value

A list (an object of class "smoothproptest") containing some of input values, test statistics and \( p \)-values computed by various methods (some of them are NULL if not computed). The most important components are:

- `stat` 
  the test statistic (`stat.bic` if `data.driven=TRUE`, `stat.d` otherwise).
- `p` 
  the \( p \)-value corresponding to `stat` (one of the \( p \)-values below).
- `stat.d` 
  the test statistic of the fixed dimension test.
- `p.d.chisqd`, `p.d.sim` 
  the \( p \)-value of the fixed dimension test based on the asymptotic \( \chi^2 \) distribution and on simulations.
stat.bic  the test statistic of the data-driven test.

p.bic.h, p.bic.sim, p.bic.chisq1  the $p$-value of the data-driven test based on the $H$-approximation, simulations, and asymptotic $\chi^2$ (only for nested alternatives; theoretically works, practically not).

Author(s)

David Kraus, http://www.davidkraus.net/

References


See Also

scoreproptest, coxph

Examples

## Case 4 of Kvaloy & Neef (2004, Lifetime Data Anal.):
## data generated from the distribution with hazard rate
## $\lambda(t)=\exp(0.5t Z_1+Z_2-8)$
## $(Z_1,Z_2)$ jointly normal with $E=4$, var=1, cor=rho
## censoring times uniform(0,5)

n = 200
rho = .3
z = matrix(rnorm(n*2),ncol=2) %*% chol(matrix(c(1,rho,rho,1),2)) + 4
a = .5
tim = 1/(a*z[,1]) * log(1-a*z[,1]*exp(-z[,2]+8)*log(runif(n)))
ct = 5*runif(n)
nc = tim<ct
tim = pmin(tim,ct)
fit = coxph(Surv(tim,nc)~z)

## Tests using nested subsets; p-values computed by H-approximation

test1 = smoothproptest(fit,covariate=1)  # testing Z_1 (nonproportional)
print(test1,print.alt=TRUE)  # print details on the alternative models

test2 = smoothproptest(fit,covariate=2)  # testing Z_2 (proportional)
print(test2,print.alt=TRUE)  # print details on the alternative models

## Tests using all subsets; p-values computed by H-approximation

test1 = smoothproptest(fit,covariate=1,all.subsets=TRUE)  # Z_1
print(test1,print.alt=TRUE)  # print details on the alternative models
test2 = smoothproptest(fit, covariate=2, all.subsets=TRUE)  # Z_2
print(test2, print.alt=TRUE)  # print details on the alternative models
Index

*Topic package
  proptest-package, 4
*Topic survival
  allsubsets, 1
  mini.survival, 2
  plot.scoreproptest, 3
  proptest-package, 4
  scoreproptest, 5
  smoothproptest, 8
agreg.fit, 2
allsubsets, 1
coxph, 6–9
coxph.control, 2
coxph.detail, 2
mini.agreg.fit (mini.survival), 2
mini.coxph.detail
  (mini.survival), 2
mini.survfit.coxph
  (mini.survival), 2
mini.survival, 2
plot.scoreproptest, 3, 6, 7
print.scoreproptest
  (scoreproptest), 5
print.smoothproptest
  (smoothproptest), 8
proptest (proptest-package), 4
proptest-package, 4
scoreproptest, 2–4, 5, 9
smoothproptest, 2, 4, 7, 8
survfit.coxph, 2