Package ‘kmc’

March 16, 2020

Version 0.2-4

Title Kaplan-Meier Estimator with Constraints for Right Censored Data
-- a Recursive Computational Algorithm

Author Yifan Yang <yifan.yang@transwarp.io>, Mai Zhou<mai@ms.uky.edu>

Maintainer Yifan Yang <yifan.yang@uky.edu>

Description Given constraints for right censored data, we use a recursive computational algo-

rithm to calculate the the "constrained" Kaplan-Meier estimator. The constraint is as-

sumed given in linear estimating equations or mean functions. We also illus-

trate how this leads to the empirical likelihood ratio test with right censored data and acceler-

ated failure time model with given coefficients. EM algorithm from emplik pack-

age is used to get the initial value. The properties and performance of the EM algorithm is dis-


URL http://github.com/yfyang86/kmc

License LGPL-3

Depends R (>= 2.13.1), compiler, rootSolve, emplik

LinkingTo Rcpp

NeedsCompilation yes

Repository CRAN

Date/Publication 2020-03-16 13:30:02 UTC

R topics documented:

  kmc.bjtest ............................. 2
  kmc.solve ................................ 3
  plotkmc2D .............................. 6

Index 8
The function `kmc.bjtest` calculates the NPMLE with constraints for accelerated failure time model with given coefficients.

**Description**

Use the empirical likelihood ratio and Wilks theorem to test if the regression coefficient equals beta.

\[
EL(F) = \prod_{i=1}^{n} (\Delta F(T_i))^{\delta_i} (1 - F(T_i))^{1-\delta_i}
\]

with constraints

\[
\sum_{i} g(T_i) \Delta F(T_i) = 0, \quad i = 1, 2, \ldots
\]

Instead of EM algorithm, this function calculates the Kaplan-Meier estimator with mean constraints recursively to test \( H_0 : \beta = \beta_0 \) in the accelerated failure time model:

\[
\log(T_i) = y_i = x_i \beta^T + \epsilon_i,
\]

where \( \epsilon \) is distribution free.

**Usage**

`kmc.bjtest(y, d, x, beta, init.st="naive")`

**Arguments**

- `y`: Response variable vector (length n).
- `d`: Status vector (length n), 0: right censored; 1 uncensored.
- `x`: n by p explanatory variable matrix.
- `beta`: The value of the regression coefficient vector (length p) to be tested.
- `init.st`: Type of methods to initialize the algorithm. By default, init.st is set to 1/n

**Details**

The empirical likelihood is the likelihood of the error term when the coefficients are specified. Model assumptions are the same as requirements of a standard Buckley-James estimator.

**Value**

A list with the following components:

- `prob`: the probabilities that max the empirical likelihood under estimating equation.
- `logel1`: the log empirical likelihood without constraints, i.e. under Kaplan-Meier of residuals'
- `logel2`: the log empirical likelihood with constraints, i.e. under null hypotheses or estimation equations.
- `"-2LLR"`: the -2 loglikelihood ratio; have approximate chisq distribution under null hypotheses.
kmc.solve

Author(s)
Mai Zhou(mai@ms.uky.edu), Yifan Yang(yifan.yang@uky.edu)

References
Buckley, J. and James, I. (1979). Linear regression with censored data. *Biometrika, 66* 429-36

See Also
plotkmc2D, bjtest.

Examples
```r
x <- c(1, 1.5, 2, 3, 4.2, 5.0, 6.1, 5.3, 4.5, 0.9, 2.1, 4.3) # positive time
d <- c(1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1) # status censored/uncensored
```

Description
This function calculate the Kaplan-Meier estimator with mean constraints recursively.

\[ El(F) = \prod_{i=1}^{n} (\Delta F(T_i))^{\delta_i} (1 - F(T_i))^{1-\delta_i} \]

with constraints
\[
\sum_{i} g(T_i) \Delta F(T_i) = 0, \quad i = 1, 2, ...
\]

It uses Lagrange multiplier directly.

Usage
```r
kmc.solve(x, d, g, em.boost = T, using.num = T, using.Fortran = T, using.C = F, tmp.tag = T, rtol = 1e-09, control = list(nr.it = 20, nr.c = 1, em.it = 3),...)
```
Arguments

- **x**
  - Non-negative real vector. The observed time.

- **d**
  - 0/1 vector. Censoring status indicator, 0: right censored; 1 uncensored

- **g**
  - list of constraint functions. It should be a list of functions list(f1,f2,...)

- **em.boost**
  - A logical value. It determines whether the EM algorithm is used to get the initial value, default=TRUE. See 'Details' for EM control.

- **using.num**
  - A logical value. It determines whether the numeric derivatives is used in iterations, default=TRUE.

- **using.Fortran**
  - A logical value. It determines whether Fortran is used in root solving, default=F.

- **using.C**
  - A logical value. It determines whether to use Rcpp in iteraruiib, default=T. This option will promote the computational efficiency of the KMC algorithm. Development version works on one constraint only, otherwise it will generate a Error information. It won’t work on using.num=F.

- **tmp.tag**
  - Development version needs it, keep it as TRUE.

- **rtol**
  - Tolerance used in rootSolve(multiroot) package, see 'rootSolve::multiroot'.

- **control**
  - A list. The entry nr.it controls max iterations allowed in N-R algorithm default=20; nr.c is the scaler used in N-R algorithm default=1; em.it is max iteration if use EM algorithm (em.boost) to get the initial value of lambda, default=3.

... specifies unspecified yet.

Details

The function check_G_mat checks whether the solution space is null or not under the constraint. But due to the computational complexity, it will detect at most two conditions.

Value

A list with the following components:

- **loglik.ha**
  - The log empirical likelihood without constraints

- **loglik.h0**
  - The log empirical likelihood with constraints

- **"-2LLR"**
  - The -2 Log empirical likelihood ratio

- **phat**
  - $\Delta F(T_i)$

- **pvalue**
  - The p-value of the test

- **df**
  - Degree(s) of freedom. It equals the number of constraints.

- **lambda**
  - The lambda is the Lagrangian multiplier described in reference.

Author(s)

Mai Zhou(mai@ms.uky.edu), Yifan Yang(yifan.yang@uky.edu)
kmc.solve

References


See Also

plotkmc2D.

Examples

```r
x <- c(1, 1.5, 2, 3, 4.2, 5.0, 6.1, 5.3, 4.5, 0.9, 2.1, 4.3) # positive time
d <- c(1, 1, 0, 1, 1, 1, 1, 0, 0, 1) # status censored/uncensored

# dim =1
f <- function(x) { x-3.7} # \sum f(ti) wi ~ 0
g = list( f=f) ; #define constraint as a list
kmc.solve( x,d,g) ; #using default
kmc.solve( x,d,g,using.C=TRUE) ; #using Rcpp

# dim =2
myfun5 <- function( x) {x^2-16.5}
g = list( f1=f,f2=myfun5) ; #define constraint as a list
re0 <- kmc.solve( x,d,g);

# Print Estimation and other information
# with option: digits = 5
f_print <- function(x, digits = 5){
cat("\n---------------------------------------------------------------------------------
")
cat("A Recursive Formula for the Kaplan-Meier Estimator with Constraint\n")
cat("Information:\n")
cat("Number of Constraints:\t", length(x$g), "\n")
cat("\n")
cat("\n")
cat("-------------------------------------------\n")
}
```

```r
cat("\n")
cat("Log-likelihood(Ha)\", "Log-likelihood(H0)\", 
"-2LLR\", paste("p-Value(df=", length(x$g), ")",sep = ")")
re <- matrix(c(x[[1]], x[[2]], x[[3]], 1 - pchisq(x[[3]], 
length(x$g))), nrow = 1)
colnames(re) <- names
```
rownames(re) <- "Est"
print.default(format(re, digits = digits), print.gap = 2,
quote = FALSE, df = length(x$g))
cat("---------------------------------------------------------------
")
}
f_print(re0)

---

**plotkmc2D**  
*Plot the contour plot of log-likelihood around the H0 (dim=2).*

**Description**  
Given a kmc object, this function will produce contour plot if there were two constraints.

**Usage**  
```r
plotkmc2D(resultkmc, flist=list(f1=function(x){x}, f2=function(x){x^2}), range0=c(0.2, 3, 20))
```

**Arguments**
- `resultkmc`: S3 Object of kmcS3.
- `flist`: list of two functions, `flist=list( f1=function( x) x , f2=function( x) x^2 )`
- `range0`: A vector that helps to determine the range of the contour plot, i.e (center[1]-range0[1], center[2]-range0[2]) to (center+range0[1], center[2]+range0[2]). The third parameter defines the number of grids would be used.

**Value**  
- `X`: x.grid
- `Y`: y.grid
- `Z`: grid value

**Author(s)**
Yifan Yang

**Examples**
```r
x <- c( 1, 1.5, 2, 3, 4.2, 5.0, 6.1, 5.3, 4.5, 0.9, 2.1, 4.3)
d <- c( 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1)

f<-function( x) { x-3.7}
myfun5 <- function( x) {
  x^2-16.5
}
```
# construnct g as a LIST!

g=list( f1=f,f2=myfun5) ;
kmc.solve( x,d,g) ->re0;

#plotkmc2D(re0) ->ZZ; # run this to generate contour plot
#Advanced PLOT option using ggplot2: not run
#library(reshape2)
#volcano3d <- melt(ZZ$Z)
#names(volcano3d) <- c("x", "y", "z")

#volcano3d$x <- ZZ$X[volcano3d$x];
#volcano3d$y <- ZZ$Y[volcano3d$y];

#library(ggplot2)
#v <- ggplot(volcano3d, aes(x, y, z=z));
#v +geom_tile(aes(fill = z)) + stat_contour()+scale_fill_gradientn("Custom
#Colours",colours=grey.colors(10));
#c("lightblue","blue","green","yellow","orange","red")
#X11();
#qplot(x, y, z = z, data = volcano3d, stat = "contour", geom = "path")
Index

bjtest, 3
kmc.bjtest, 2
kmc.solve, 3
plotkmc2D, 3, 5, 6