Package 'PenCoxFrail'

July 21, 2025

Type Package

Title Regularization in Cox Frailty Models

Version 2.0.0

Date 2024-07-04

Author Andreas Groll

Maintainer Andreas Groll <groll@statistik.tu-dortmund.de>

Description
Different regularization approaches for Cox Frailty Models by penalization methods are provided.

License GPL (>= 2)

Imports Rcpp (>= 0.11.6), Matrix, methods, coxme

Depends survival

LinkingTo Rcpp, RcppArmadillo

Repository CRAN

SystemRequirements C++

NeedsCompilation yes

Date/Publication 2024-07-04 22:50:10 UTC

Contents

bs.design *Generate a B-spline design matrix*

Description

The function generates a B-spline design matrix with equidistant knots for given degree of the splines and number of basis functions.

Usage

```
bs.design(x, xl, xr, spline.degree, nbasis, comp = NULL)
```

Arguments

`x`	the positions where spline to be evaluated.
`xl`	lower intervall boundary where spline functions are relevant.
`xr`	upper intervall boundary where spline functions are relevant.
`spline.degree`	(polynomial) degree of the B-splines.
`nbasis`	number of basis functions used.
`comp`	Specify if only specific columns of the B-spline design matrix should be returned. Default is NULL and the whole B-spline design matrix is returned.

Value

The B-spline design matrix is returned.

Author(s)

Andreas Groll `<groll@math.lmu.de>`

See Also

`pencoxfrail`

Examples

```
x <- rnorm(100)
B <- bs.design(x=x, xl=min(x), xr=max(x), spline.degree=3, nbasis=5)
```

`coxFL` *A full likelihood approach for Cox Frailty Models.*

Description

A full likelihood approach for Cox Frailty Models based on the full likelihood is provided.

Usage

```
coxFL(fix=formula, rnd=NULL, vary.coef=NULL, data, control = list())
```

Arguments

`fix`	a two-sided linear formula object describing the unpenalized fixed (time-constant) effects part of the model, with the response on the left of a ~ operator and the terms, separated by + operators, on the right. The response must be a survival object as returned by the `Surv` function.
`rnd`	a two-sided linear formula object describing the random-effects part of the model, with the grouping factor on the left of a ~ operator and the random terms, separated by + operators, on the right.Default is NULL, so no random effects are present.
`vary.coef`	a one-sided linear formula object describing the time-varying effects part of the model, with the time-varying terms, separated by + operators, on the right side of a ~ operator.Default is NULL, so no time-varying effects are incorporated.
`data`	the data frame containing the variables named in the three preceding `formula` arguments.
`control`	a list of control values for the estimation algorithm to replace the default values returned by the function `coxlassoControl`. Defaults to an empty list.

Details

The `coxFL` algorithm is designed to investigate the effect structure in the Cox frailty model, which is a widely used model that accounts for heterogeneity in time-to-event data. Since in survival models one has to account for possible variation of the effect strength over time, some features can incorporated with time-varying effects.

Package:	pencoxfrail
Type:	Package
Version:	1.1.2
Date:	2023-08-25
License:	GPL-2
LazyLoad:	yes

for loading a dataset type data(nameofdataset)

Value

Generic functions such as `print`, `predict`, `plot` and `summary` have methods to show the results of the fit.

The `predict` function uses also estimates of random effects for prediction, if possible (i.e. for known subjects of the grouping factor). Either the survival stepfunction or the baseline hazard (not cumulative!) can be calculated by specifying one of two possible methods: `method=c("hazard","survival")`. By default, for each new subject in `new.data` an individual stepfunction is calculated on a pre-specified time grid, also accounting for covariate changes over time. Alternatively, for `new.data` a single vector of a specific (time-constant) covariate combination can be specified.

Usage: `predict(coxlasso.obj,new.data,time.grid,method=c("hazard","survival"))`

The `plot` function plots all time-varying effects, including the baseline hazard.

`call`	a list containing an image of the `coxlasso` call that produced the object.
`baseline`	a vector containing the estimated B-spline coefficients of the baseline hazard. If the covariates corresponding to the time-varying effects are centered (and standardized, see `coxlassoControl`), the coefficients are transformed back to the original scale.
`time.vary`	a vector containing the estimated B-spline coefficients of all time-varying effects. If the covariates corresponding to the time-varying effects are standardized (see `coxlassoControl`) the coefficients are transformed back to the original scale.
`coefficients`	a vector containing the estimated fixed effects.
`ranef`	a vector containing the estimated random effects.
`Q`	a scalar or matrix containing the estimates of the random effects standard deviation or variance-covariance parameters, respectively.
`Delta`	a matrix containing the estimates of fixed and random effects (columns) for each iteration (rows) of the main algorithm (i.e. before the final re-estimation step is performed, see details).
`Q_long`	a list containing the estimates of the random effects variance-covariance parameters for each iteration of the main algorithm.
`iter`	number of iterations until the main algorithm has converged.
`knots`	vector of knots used in the B-spline representation.
`Phi.big`	large B-spline design matrix corresponding to the baseline hazard and all time-varying effects. For the time-varying effects, the B-spline functions (as a function of time) have already been multiplied with their associated covariates.
`time.grid`	the time grid used in when approximating the (Riemann) integral involved in the model's full likelihood.
`m`	number of metric covariates with time-varying effects.
`m2`	number of categorical covariates with time-varying effects.

Author(s)

Andreas Groll `<groll@statistik.tu-dortmund.de>`

References

Groll, A., T. Hastie and G. Tutz (2016). Regularization in Cox Frailty Models. Ludwig-Maximilians-University. *Technical Report* 191.

See Also

`coxFLControl`, `Surv`, `pbc`

Examples

```
## Not run:
data(lung)

# remove NAs
lung <- lung[!is.na(lung$inst),]

# transform inst into factor variable
lung$inst <- as.factor(lung$inst)

# just for illustration, create factor with only three ph.ecog classes
lung$ph.ecog[is.na(lung$ph.ecog)] <- 2
lung$ph.ecog[lung$ph.ecog==3] <- 2
lung$ph.ecog <- as.factor(lung$ph.ecog)

fix.form <- as.formula("Surv(time, status) ~ 1 + age + ph.ecog + sex")

coxFL.obj <- coxFL(fix=fix.form, data=lung,
                control=list(print.iter=TRUE, exact = 1))
coef(coxFL.obj)

# For comparison: coxph
coxph.1 <- coxph(fix.form , data=lung)
coef(coxph.1)

# now add random institutional effect
coxFL.obj2 <- coxFL(fix=fix.form, rnd = list(inst=~1),
              data=lung, control=list(print.iter=TRUE, exact = 1))
coef(coxFL.obj2)
# print frailty Std.Dev.
print(coxFL.obj2$Q)
# print frailties
print(coxFL.obj2$ranef)

# For comparison:  coxph
fix.form.cox <- update(fix.form, ~ . + frailty(inst, distribution="gaussian"))
coxph.2 <- coxph(fix.form.cox , data=lung)
coef(coxph.2)
# print frailty Std.Dev.
print(sqrt(coxph.2$history[[1]]$history[nrow(coxph.2$history[[1]]$history), 1]))
# print frailties
print(coxph.2$frail)
```

```
# now fit a time-varying effect for age
fix.form <- as.formula("Surv(time, status) ~ 1 + ph.ecog + sex")
vary.coef <- as.formula("~ age")

coxFL.obj3 <- coxFL(fix=fix.form,vary.coef=vary.coef,
              data=lung, control=list(print.iter=TRUE))
summary(coxFL.obj3)

# show fit
plot(coxFL.obj3)

# predict survival curve of new subject, institution 1 and up to time 300
pred.obj <- predict(coxFL.obj2, newdata=data.frame(inst=1, time=NA, status=NA, age=26,
              ph.ecog=2,sex=1), time.grid=seq(0,300,by=1))

# plot predicted hazard function
plot(pred.obj$time.grid,pred.obj$haz,type="l",xlab="time",ylab="hazard")

# plot predicted survival function
plot(pred.obj$time.grid,pred.obj$survival,type="l",xlab="time",ylab="survival")

## specify a larger new data set
new.data <- data.frame(inst=c(1,1,6), time=c(20,40,200),
      status=c(NA,NA,NA), age=c(26,26,54), ph.ecog=c(0,0,2),sex=c(1,1,1))

## as here no frailties have been specified, id.var needs to be given!
pred.obj2 <- predict(coxFL.obj3, newdata=new.data,id.var = "inst")

# plot predicted hazard functions (for the available time intervals)
# for individual 1 and 3
plot(pred.obj2$time.grid[!is.na(pred.obj2$haz[,1])],
      pred.obj2$haz[,1][!is.na(pred.obj2$haz[,1])],
      type="l",xlab="time",ylab="hazard",xlim=c(0,200),
      ylim=c(0,max(pred.obj2$haz,na.rm=T)))
lines(pred.obj2$time.grid[!is.na(pred.obj2$haz[,3])],
      pred.obj2$haz[,3][!is.na(pred.obj2$haz[,3])],
      col="red",lty=2,)

# plot predicted survival functions (for the available time intervals)
# for individual 1 and 3
plot(pred.obj2$time.grid[!is.na(pred.obj2$survival[,1])],
    pred.obj2$survival[,1][!is.na(pred.obj2$survival[,1])],
    type="l",xlab="time",ylab="hazard",xlim=c(0,200),
    ylim=c(0,max(pred.obj2$survival,na.rm=T)))
lines(pred.obj2$time.grid[!is.na(pred.obj2$survival[,3])],
      pred.obj2$survival[,3][!is.na(pred.obj2$survival[,3])],
      col="red",lty=2,)


## End(Not run)
```

coxFLControl — *Control Values for* `coxFL` *fit*

Description

The values supplied in the function call replace the defaults and a list with all possible arguments is returned. The returned list is used as the `control` argument to the `coxFL` function.

Usage

```
coxFLControl(start = NULL, index=NULL, q_start = NULL, conv.eps = 1e-3,
                              standardize = FALSE, center = FALSE,
                              smooth=list(nbasis = 6, penal = 1e+2),
                              print.iter = FALSE, max.iter = 100,
                              exact = NULL, xr = NULL, eps = 1e-3, ...)
```

Arguments

`start`	a vector of suitable length containing starting values for the spline-coefficients of the baseline hazard and the time-varying effects, followed by the fixed and random effects. The correct ordering is important. Default is a vector full of zeros.
`index`	vector which defines the grouping of the variables. Components sharing the same number build a group and factor variables get a single number (and are automatically treated as a group). Non-penalized coefficients are marked with NA.
`q_start`	a scalar or matrix of suitable dimension, specifying starting values for the random-effects variance-covariance matrix. Default is a scalar 0.1 or diagonal matrix with 0.1 in the diagonal, depending on the dimension of the random effects.
`conv.eps`	controls the speed of convergence. Default is 1e-3.
`smooth`	a list specifying the number of basis functions `nbasis` (used for the baseline hazard and all time-varying effects) and the starting value for the smoothness penalty parameter `penal`, which is only applied in the first iteration. Then, in all following iterations the wiggly components of the smooths are treated as random effects. The degree of the B-splines is fixed to be three (i.e. cubic splines).
`standardize`	logical. If true, the covariates corresponding to the fixed effects will be scaled to a variance equal to one. Default is TRUE.
`center`	logical. If true, the covariates corresponding to the time-varying effects will be centered. Default is FALSE (and centering is only recommended if really necessary; it can also have a strong effect on the baseline hazard, in particular, if a strong penalty is selected).
`print.iter`	logical. Should the number of iterations be printed? Default is FALSE.
`max.iter`	the number of iterations for the final Fisher scoring re-estimation procedure. Default is 200.

`exact`	controls the exactness of the (Riemann) integral approximations. If not set by the user to a specific value, it will be automatically chosen such that the time interval [0,t_max] will be divided in about 1000 equal sized Riemann bars.
`xr`	maximal time point that is regarded. Default is NULL and the maximal event or censoring time point in the data is used.
`eps`	Small epsilon that controls which fixed effects are set to zero: parameters with an absolute value smaller than epsilon are taken to be zero. Default is 1e-3.
`...`	Futher arguments to be passed.

Value

a list with components for each of the possible arguments.

Author(s)

Andreas Groll <`groll@math.lmu.de`>

See Also

`coxFL`

Examples

```
# Use different weighting of the two penalty parts
# and lighten the convergence criterion
coxFLControl(c.app = 1e-5, conv.eps=1e-3)
```

`coxlasso` *A LASSO approach for Cox Frailty Models.*

Description

A LASSO approach for Cox Frailty Models based on the Cox full likelihood is provided.

Usage

```
coxlasso(fix=formula, rnd=NULL, vary.coef=NULL, data, xi,
                adaptive.weights = NULL, control = list())
```

Arguments

`fix`	a two-sided linear formula object describing the LASSO-penalized fixed (time-constant) effects part of the model, with the response on the left of a ~ operator and the terms, separated by + operators, on the right. The response must be a survival object as returned by the `Surv` function.

`rnd`	a two-sided linear formula object describing the random-effects part of the model, with the grouping factor on the left of a ~ operator and the random terms, separated by + operators, on the right.Default is NULL, so no random effects are present.
`vary.coef`	a one-sided linear formula object describing the time-varying effects part of the model, with the time-varying terms, separated by + operators, on the right side of a ~ operator.Default is NULL, so no time-varying effects are incorporated.
`data`	the data frame containing the variables named in the three preceding `formula` arguments.
`xi`	the LASSO-penalty parameter that controls the strenght of the penalty term. The optimal penalty parameter is a tuning parameter of the procedure that has to be determined, e.g. by K-fold cross validation (see `cv.coxlasso` for details or the quick demo for an example).
`adaptive.weights`	for the LASSO-penalized fixed effects a vector of adaptive weights can be passed to the procedure. If no adaptive weights are specified, an unpenalized model (i.e. $\xi = 0$) is fitted by the `coxFL` function and the obtained estimates are used as adaptive weights (see value section).
`control`	a list of control values for the estimation algorithm to replace the default values returned by the function `coxlassoControl`. Defaults to an empty list.

Details

The `coxlasso` algorithm is designed to investigate the effect structure in the Cox frailty model, which is a widely used model that accounts for heterogeneity in time-to-event data. Since in survival models one has to account for possible variation of the effect strength over time, some features can incorporated with time-varying effects.

The penalty is depending on the LASSO tuning parameter ξ, which has to be determined by a suitable technique, e.g. by K-fold cross validation.

Package:	pencoxfrail
Type:	Package
Version:	1.1.2
Date:	2023-08-25
License:	GPL-2
LazyLoad:	yes

for loading a dataset type data(nameofdataset)

Value

Generic functions such as `print`, `predict`, `plot` and `summary` have methods to show the results of the fit.

The `predict` function uses also estimates of random effects for prediction, if possible (i.e. for known subjects of the grouping factor). Either the survival stepfunction or the baseline hazard (not cumulative!) can be calculated by specifying one of two possible methods: `method=c("hazard","survival")`.

By default, for each new subject in `new.data` an individual stepfunction is calculated on a prespecified time grid, also accounting for covariate changes over time. Alternatively, for `new.data` a single vector of a specific (time-constant) covariate combination can be specified.

Usage: `predict(coxlasso.obj,new.data,time.grid,method=c("hazard","survival"))`

The `plot` function plots all time-varying effects, including the baseline hazard.

`call`	a list containing an image of the `coxlasso` call that produced the object.
`baseline`	a vector containing the estimated B-spline coefficients of the baseline hazard. If the covariates corresponding to the time-varying effects are centered (and standardized, see `coxlassoControl`), the coefficients are transformed back to the original scale.
`time.vary`	a vector containing the estimated B-spline coefficients of all time-varying effects. If the covariates corresponding to the time-varying effects are standardized (see `coxlassoControl`) the coefficients are transformed back to the original scale.
`coefficients`	a vector containing the estimated fixed effects.
`ranef`	a vector containing the estimated random effects.
`Q`	a scalar or matrix containing the estimates of the random effects standard deviation or variance-covariance parameters, respectively.
`Delta`	a matrix containing the estimates of fixed and random effects (columns) for each iteration (rows) of the main algorithm (i.e. before the final re-estimation step is performed, see details).
`Q_long`	a list containing the estimates of the random effects variance-covariance parameters for each iteration of the main algorithm.
`iter`	number of iterations until the main algorithm has converged.
`adaptive.weights`	if not given as an argument by the user, a two-column matrix of adaptive weights is calculated by the `coxFL` function; the first column contains the weights $w_{\Delta,k}$, the second column the weights v_k from $\xi \cdot J(\zeta, \alpha)$.
`knots`	vector of knots used in the B-spline representation.
`Phi.big`	large B-spline design matrix corresponding to the baseline hazard and all time-varying effects. For the time-varying effects, the B-spline functions (as a function of time) have already been multiplied with their associated covariates.
`time.grid`	the time grid used in when approximating the (Riemann) integral involved in the model's full likelihood.
`m`	number of metric covariates with time-varying effects.
`m2`	number of categorical covariates with time-varying effects.

Author(s)

Andreas Groll <`groll@statistik.tu-dortmund.de`>
Maike Hohberg <`mhohber@uni-goettingen.de`>

References

Groll, A., T. Hastie and G. Tutz (2017). Selection of Effects in Cox Frailty Models by Regularization Methods. *Biometrics* 73(3): 846-856.

See Also

`coxlassoControl`,`cv.coxlasso`,`coxFL`,`Surv`,`pbc`

Examples

```
## Not run:
data(lung)

# remove NAs
lung <- lung[!is.na(lung$inst),]

# transform inst into factor variable
lung$inst <- as.factor(lung$inst)

# just for illustration, create factor with only three ph.ecog classes
lung$ph.ecog[is.na(lung$ph.ecog)] <- 2
lung$ph.ecog[lung$ph.ecog==3] <- 2
lung$ph.ecog <- as.factor(lung$ph.ecog)

fix.form <- as.formula("Surv(time, status) ~ 1 + age + ph.ecog + sex")

lasso.obj <- coxlasso(fix=fix.form, data=lung, xi=10,
                control=list(print.iter=TRUE, exact = 1))
coef(lasso.obj)


# now add random institutional effect
lasso.obj2 <- coxlasso(fix=fix.form, rnd = list(inst=~1),
              data=lung, xi=10,control=list(print.iter=TRUE, exact = 1))
coef(lasso.obj2)
# print frailty Std.Dev.
print(lasso.obj2$Q)
# print frailties
print(lasso.obj2$ranef)


# now fit a time-varying effect for age
fix.form <- as.formula("Surv(time, status) ~ 1 + ph.ecog + sex")
vary.coef <- as.formula("~ age")

lasso.obj3 <- coxlasso(fix=fix.form,vary.coef=vary.coef,
              data=lung, xi=10,control=list(print.iter=TRUE))
summary(lasso.obj3)

# show fit
plot(lasso.obj3)
```

```
# predict survival curve of new subject, institution 1 and up to time 300
pred.obj <- predict(lasso.obj2, newdata=data.frame(inst=1, time=NA, status=NA, age=26,
              ph.ecog=2,sex=1), time.grid=seq(0,300,by=1))

# plot predicted hazard function
plot(pred.obj$time.grid,pred.obj$haz,type="l",xlab="time",ylab="hazard")

# plot predicted survival function
plot(pred.obj$time.grid,pred.obj$survival,type="l",xlab="time",ylab="survival")

## specify a larger new data set
new.data <- data.frame(inst=c(1,1,6), time=c(20,40,200),
      status=c(NA,NA,NA), age=c(26,26,54), ph.ecog=c(0,0,2),sex=c(1,1,1))

## as here no frailties have been specified, id.var needs to be given!
pred.obj2 <- predict(lasso.obj3, newdata=new.data,id.var = "inst")

# plot predicted hazard functions (for the available time intervals)
plot(pred.obj2$time.grid[!is.na(pred.obj2$haz[,1])],
      pred.obj2$haz[,1][!is.na(pred.obj2$haz[,1])],
      type="l",xlab="time",ylab="hazard",xlim=c(0,200),
      ylim=c(0,max(pred.obj2$haz,na.rm=T)))
lines(pred.obj2$time.grid[!is.na(pred.obj2$haz[,3])],
      pred.obj2$haz[,3][!is.na(pred.obj2$haz[,3])],
      col="red",lty=2,)

# plot predicted survival functions (for the available time intervals)
plot(pred.obj2$time.grid[!is.na(pred.obj2$survival[,1])],
    pred.obj2$survival[,1][!is.na(pred.obj2$survival[,1])],
    type="l",xlab="time",ylab="hazard",xlim=c(0,200),
    ylim=c(0,max(pred.obj2$survival,na.rm=T)))
lines(pred.obj2$time.grid[!is.na(pred.obj2$survival[,3])],
      pred.obj2$survival[,3][!is.na(pred.obj2$survival[,3])],
      col="red",lty=2,)


# see also demo("coxlasso-lung")

## End(Not run)
```

`coxlassoControl` *Control Values for* `coxlasso` *fit*

Description

The values supplied in the function call replace the defaults and a list with all possible arguments is returned. The returned list is used as the `control` argument to the `coxlasso` function.

Usage

```
coxlassoControl(start = NULL, index=NULL, q_start = NULL, conv.eps = 1e-3,
                          standardize = TRUE, center = FALSE,
                          smooth=list(nbasis = 6, penal = 1e+2),
                          print.iter = FALSE, max.iter = 100, c.app = 1e-6,
                      exact = NULL, xr = NULL, eps = 1e-2, quant.knots = TRUE,...)
```

Arguments

`start`	a vector of suitable length containing starting values for the spline-coefficients of the baseline hazard and the time-varying effects, followed by the fixed and random effects. The correct ordering is important. Default is a vector full of zeros.
`index`	vector which defines the grouping of the variables. Components sharing the same number build a group and factor variables get a single number (and are automatically treated as a group). Non-penalized coefficients are marked with NA.
`q_start`	a scalar or matrix of suitable dimension, specifying starting values for the random-effects variance-covariance matrix. Default is a scalar 0.1 or diagonal matrix with 0.1 in the diagonal, depending on the dimension of the random effects.
`conv.eps`	controls the speed of convergence. Default is 1e-3.
`center`	logical. If true, the covariates corresponding to the time-varying effects will be centered. Default is FALSE (and centering is only recommended if really necessary; it can also have a strong effect on the baseline hazard, in particular, if a strong penalty is selected).
`standardize`	logical. If true, the covariates corresponding to the fixed effects will be scaled to a variance equal to one. Default is TRUE.
`smooth`	a list specifying the number of basis functions `nbasis` (used for the baseline hazard and all time-varying effects) and the smoothness penalty parameter `penal`, which is only applied to the baseline hazard. All time-varying effects are penalized by the specific double-penalty $\xi \cdot J(\zeta, \alpha)$ (see `coxlasso`), which is based on the overall penalty parameter ξ (specified in the main function `coxlasso`) and on the weighting between the two penalty parts ζ. The degree of the B-splines is fixed to be three (i.e. cubic splines).
`print.iter`	logical. Should the number of iterations be printed? Default is FALSE.
`max.iter`	the number of iterations for the final Fisher scoring re-estimation procedure. Default is 200.
`c.app`	The parameter controlling the exactness of the quadratic approximations of the penalties. Default is 1e-6.
`exact`	controls the exactness of the (Riemann) integral approximations. If not set by the user to a specific value, it will be automatically chosen such that the time interval [0,t_max] will be divided in about 1000 equal sized Riemann bars.
`xr`	maximal time point that is regarded. Default is NULL and the maximal event or censoring time point in the data is used.

`eps`	Small epsilon that controls which fixed effects are set to zero: parameters with an absolute value smaller than epsilon are taken to be zero. Default is 1e-2.
`quant.knots`	Shall the knots be defined based on quantiles? Default is TRUE.
.	
`...`	Futher arguments to be passed.

Value

a list with components for each of the possible arguments.

Author(s)

Andreas Groll <`groll@math.lmu.de`>

See Also

`coxlasso`

Examples

```
# Use different weighting of the two penalty parts
# and lighten the convergence criterion
coxlassoControl(c.app = 1e-5, conv.eps=1e-3)
```

`coxridge` *A ridge approach for Cox Frailty Models.*

Description

A ridge regression approach for Cox Frailty Models based on the Cox full likelihood is provided.

Usage

```
coxridge(fix=formula, rnd=NULL, vary.coef=NULL, xi.ridge, data, control = list())
```

Arguments

`fix`	a two-sided linear formula object describing the LASSO-penalized fixed (time-constant) effects part of the model, with the response on the left of a ~ operator and the terms, separated by + operators, on the right. The response must be a survival object as returned by the `Surv` function.
`rnd`	a two-sided linear formula object describing the random-effects part of the model, with the grouping factor on the left of a ~ operator and the random terms, separated by + operators, on the right.Default is NULL, so no random effects are present.

`vary.coef`	a one-sided linear formula object describing the time-varying effects part of the model, with the time-varying terms, separated by + operators, on the right side of a ~ operator.Default is NULL, so no time-varying effects are incorporated.
`xi.ridge`	the ridge-penalty parameter that controls the strenght of the penalty term. The optimal penalty parameter is a tuning parameter of the procedure that has to be determined, e.g. by K-fold cross validation.
`data`	the data frame containing the variables named in the three preceding `formula` arguments.
`control`	a list of control values for the estimation algorithm to replace the default values returned by the function `coxlassoControl`. Defaults to an empty list.

Details

The `coxridge` algorithm is designed to investigate the effect structure in the Cox frailty model, which is a widely used model that accounts for heterogeneity in time-to-event data. Since in survival models one has to account for possible variation of the effect strength over time, some features can incorporated with time-varying effects.

The penalty is depending on the ridge tuning parameter ξ.ridge, which has to be determined by a suitable technique, e.g. by K-fold cross validation.

Package:	pencoxfrail
Type:	Package
Version:	1.1.2
Date:	2023-08-25
License:	GPL-2
LazyLoad:	yes

for loading a dataset type data(nameofdataset)

Value

Generic functions such as `print`, `predict`, `plot` and `summary` have methods to show the results of the fit.

The `predict` function uses also estimates of random effects for prediction, if possible (i.e. for known subjects of the grouping factor). Either the survival stepfunction or the baseline hazard (not cumulative!) can be calculated by specifying one of two possible methods: `method=c("hazard","survival")`. By default, for each new subject in `new.data` an individual stepfunction is calculated on a pre-specified time grid, also accounting for covariate changes over time. Alternatively, for `new.data` a single vector of a specific (time-constant) covariate combination can be specified.

Usage: `predict(coxlasso.obj,new.data,time.grid,method=c("hazard","survival"))`

The `plot` function plots all time-varying effects, including the baseline hazard.

`call`	a list containing an image of the `coxlasso` call that produced the object.
`baseline`	a vector containing the estimated B-spline coefficients of the baseline hazard. If the covariates corresponding to the time-varying effects are centered (and standardized, see `coxlassoControl`), the coefficients are transformed back to the original scale.

`time.vary`	a vector containing the estimated B-spline coefficients of all time-varying effects. If the covariates corresponding to the time-varying effects are standardized (see `coxlassoControl`) the coefficients are transformed back to the original scale.
`coefficients`	a vector containing the estimated fixed effects.
`ranef`	a vector containing the estimated random effects.
`Q`	a scalar or matrix containing the estimates of the random effects standard deviation or variance-covariance parameters, respectively.
`Delta`	a matrix containing the estimates of fixed and random effects (columns) for each iteration (rows) of the main algorithm (i.e. before the final re-estimation step is performed, see details).
`Q_long`	a list containing the estimates of the random effects variance-covariance parameters for each iteration of the main algorithm.
`iter`	number of iterations until the main algorithm has converged.
`adaptive.weights`	if not given as an argument by the user, a two-column matrix of adaptive weights is calculated by the `coxFL` function; the first column contains the weights $w_{\Delta,k}$, the second column the weights v_k from $\xi \cdot J(\zeta, \alpha)$.
`knots`	vector of knots used in the B-spline representation.
`Phi.big`	large B-spline design matrix corresponding to the baseline hazard and all time-varying effects. For the time-varying effects, the B-spline functions (as a function of time) have already been multiplied with their associated covariates.
`time.grid`	the time grid used in when approximating the (Riemann) integral involved in the model's full likelihood.
`m`	number of metric covariates with time-varying effects.
`m2`	number of categorical covariates with time-varying effects.

Author(s)

Andreas Groll <groll@statistik.tu-dortmund.de>
Maike Hohberg <mhohber@uni-goettingen.de>

References

Groll, A., T. Hastie and G. Tutz (2017). Selection of Effects in Cox Frailty Models by Regularization Methods. *Biometrics* 73(3): 846-856.

See Also

`coxlassoControl`,`cv.coxlasso`,`coxFL`,`Surv`,`pbc`

Examples

```
## Not run:
# remove NAs
lung <- lung[!is.na(lung$inst),]
```

```
# transform inst into factor variable
lung$inst <- as.factor(lung$inst)

# just for illustration, create factor with only three ph.ecog classes
lung$ph.ecog[is.na(lung$ph.ecog)] <- 2
lung$ph.ecog[lung$ph.ecog==3] <- 2
lung$ph.ecog <- as.factor(lung$ph.ecog)

fix.form <- as.formula("Surv(time, status) ~ 1 + age + ph.ecog + sex")

ridge.obj <- coxridge(fix=fix.form, data=lung, xi.ridge=10,
                control=list(print.iter=TRUE, exact = 1))
coef(ridge.obj)


# now add random institutional effect
ridge.obj2 <- coxridge(fix=fix.form, rnd = list(inst=~1),
              data=lung, xi.ridge=10,control=list(print.iter=TRUE, exact = 1))
coef(ridge.obj2)
# print frailty Std.Dev.
print(ridge.obj2$Q)
# print frailties
print(ridge.obj2$ranef)


# now fit a time-varying effect for age
fix.form <- as.formula("Surv(time, status) ~ 1 + ph.ecog + sex")
vary.coef <- as.formula("~ age")

ridge.obj3 <- ridgelasso(fix=fix.form,vary.coef=vary.coef,
              data=lung, xi.ridge=10,control=list(print.iter=TRUE))
summary(ridge.obj3)

# show fit
plot(ridge.obj3)

# predict survival curve of new subject, institution 1 and up to time 300
pred.obj <- predict(ridge.obj2, newdata=data.frame(inst=1, time=NA, status=NA, age=26,
              ph.ecog=2,sex=1), time.grid=seq(0,300,by=1))

# plot predicted hazard function
plot(pred.obj$time.grid,pred.obj$haz,type="l",xlab="time",ylab="hazard")

# plot predicted survival function
plot(pred.obj$time.grid,pred.obj$survival,type="l",xlab="time",ylab="survival")

## specify a larger new data set
new.data <- data.frame(inst=c(1,1,6), time=c(20,40,200),
      status=c(NA,NA,NA), age=c(26,26,54), ph.ecog=c(0,0,2),sex=c(1,1,1))

## as here no frailties have been specified, id.var needs to be given!
pred.obj2 <- predict(lasso.obj3, newdata=new.data,id.var = "inst")
```

```
# plot predicted hazard functions (for the available time intervals)
plot(pred.obj2$time.grid[!is.na(pred.obj2$haz[,1])],
      pred.obj2$haz[,1][!is.na(pred.obj2$haz[,1])],
      type="l",xlab="time",ylab="hazard",xlim=c(0,200),
      ylim=c(0,max(pred.obj2$haz,na.rm=T)))
lines(pred.obj2$time.grid[!is.na(pred.obj2$haz[,3])],
      pred.obj2$haz[,3][!is.na(pred.obj2$haz[,3])],
      col="red",lty=2,)

# plot predicted survival functions (for the available time intervals)
plot(pred.obj2$time.grid[!is.na(pred.obj2$survival[,1])],
    pred.obj2$survival[,1][!is.na(pred.obj2$survival[,1])],
    type="l",xlab="time",ylab="hazard",xlim=c(0,200),
    ylim=c(0,max(pred.obj2$survival,na.rm=T)))
lines(pred.obj2$time.grid[!is.na(pred.obj2$survival[,3])],
      pred.obj2$survival[,3][!is.na(pred.obj2$survival[,3])],
      col="red",lty=2,)


# see also demo("coxlasso-lung")

## End(Not run)
```

`cv.coxlasso` *Cross-validation for coxlasso*

Description

performs k-fold cross-validation for `coxlasso`, produces a plot, and returns a value for the LASSO tuning parameter ξ.

Usage

```
cv.coxlasso(fix, rnd = NULL, vary.coef = NULL, n.folds = 10, xi = NULL,
             data, adaptive.weights = NULL, print.fold = TRUE, print.xi = FALSE,
         len.xi = 100, lgrid = TRUE, ran.seed = 1909, xi.factor = 1.01, min.fold = 4,
             pass.on.start = TRUE, control = list(print.iter = FALSE))
```

Arguments

`fix`	a two-sided linear formula object describing the fixed (time-constant) effects part of the model, with the response on the left of a ~ operator and the terms, separated by + operators, on the right. The response must be a survival object as returned by the `Surv` function.
`rnd`	a two-sided linear formula object describing the random-effects part of the model, with the grouping factor on the left of a ~ operator and the random terms, separated by + operators, on the right. Default is NULL, so no random effects are present.

`vary.coef`	a one-sided linear formula object describing the time-varying effects part of the model, with the time-varying terms, separated by + operators, on the right side of a ~ operator. Default is NULL, so no time-varying effects are incorporated.
`n.folds`	number of folds. Default is 10.
`xi`	Optional user-supplied xi sequence; default is NULL, and cv.coxlasso chooses its own sequence
`data`	the data frame containing the variables named in the three preceding `formula` arguments.
`adaptive.weights`	for the LASSO-penalized fixed effects a vector of adaptive weights can be passed to the procedure. If no adaptive weights are specified, an unpenalized model (i.e. $\xi = 0$) is fitted by the `coxFL` function and the obtained estimates are used as adaptive weights (see value section).
`print.fold`	Should folds of CV be printed? Default is yes.
`print.xi`	Should current ξ value be printed? Default is no.
`len.xi`	Length of ξ grid. Default is 100.
`lgrid`	Logical; shall a logarithmized grid version for the penalty parameter be used? Default is TRUE.
`ran.seed`	Random seed number to be set. Default is 1909, the year of birth of Borussia Dortmund football club.
`xi.factor`	A factor which increases xi.max once again to be sure that xi is large enough on all sets. Default is 1.01
`min.fold`	Only those xi values are taken into account where at least min.fold folds are not NA. Default is 4.
`pass.on.start`	Shall starting values be passed onthroughout estimation? Default is TRUE
`control`	a list of control values for the estimation algorithm to replace the default values returned by the function `coxlassoControl`. Default is `print.iter = FALSE`.

Details

The function runs `coxlasso` over a grid of values ξ for each training data set with one fold omitted.

For each run, the value for the full likelihood is calculated and the average for each ξ on the grid is computed over the folds. The function choses the ξ that maximizes this likelihood value as the optimal tuning parameter value.

Value

The function returns a list `"cv.coxlasso"` which includes:

`cv.error`	a vector of mean CV error (i.e., negative likelihood) values for each ξ on the grid averaged over the folds.
`xi.opt`	a scalar value of ξ associated with the smallest CV error.
`xi.1se`	largest value of ξ such that error is within 1 standard error of the minimum.

The `plot` function plots the values of ξ against the corresponding CV error (i.e., negative likelihood) values.

Author(s)

Andreas Groll <groll@statistik.tu-dortmund.de>
Maike Hohberg <mhohber@uni-goettingen.de>

References

To appear soon.

See Also

coxlasso, coxlassoControl, coxFL, Surv, pbc

Examples

```
## Not run:
data(lung)

# remove NAs
lung <- lung[!is.na(lung$inst),]

# transform inst into factor variable
lung$inst <- as.factor(lung$inst)

# just for illustration, create factor with only three ph.ecog classes
lung$ph.ecog[is.na(lung$ph.ecog)] <- 2
lung$ph.ecog[lung$ph.ecog==3] <- 2
lung$ph.ecog <- as.factor(lung$ph.ecog)

fix.form <- as.formula("Surv(time, status) ~ 1 + age + ph.ecog + sex")

# find optimal tuning paramater
cv.coxlasso.obj <- cv.coxlasso(fix = fix.form, data = lung, n.folds = 5)

# estimate coxlasso model with optimal xi
lasso.obj <- coxlasso(fix=fix.form, data=lung, xi=cv.coxlasso.obj$xi.opt,
                control=list(print.iter=TRUE))

coef(lasso.obj)


# see also demo("coxlasso-lung")

## End(Not run)
```

---

int.approx   *Approximation of a Cox likelihood intergral*

Description

The function approximates the integral $\int_0^t exp(uB(s)\alpha)ds)$ which appears in the (full) Cox likelihood if the covariate u has a time-varying effect $\beta(t)$, which is expanded in B-splines, i.e. $\beta(t) = B(t)\alpha$.

Usage

```
int.approx(z,time.grid,B,nbasis,alpha)
```

Arguments

`z`	a vector which contains at the first component a time point up to which it should be integrated and the covariates u in the remaining components.
`time.grid`	an equally-spaced time grid on which the B-spline design matrix B has been generated. The maximal value of the time grid should usually be the maximal upper integral border that is of interest.
`B`	a B-spline design matrix, which has been created with the function `bs.design` on the full time grid `time.grid`.
`nbasis`	number of basis functions used when the B-spline design matrix B has been generated.
`alpha`	vector of B-spline coefficients.

Value

The B-spline design matrix is returned.

Author(s)

Andreas Groll <`groll@math.lmu.de`>

See Also

`pencoxfrail`

Examples

```
## generate time grid and corresponding B-spline design matrix
time.grid <- seq(0,200,by=1)
B <- bs.design(x=time.grid, xl=min(time.grid), xr=max(time.grid), spline.degree=3, nbasis=5)

## specify spline coefficients and covariate vector (with upper integral bound as first component)
alpha <- c(0.1,0.2,0.05,0.1,0.15)
z <- c(time=100,age=25)

## calculate intergal from 0 to 100
int.approx(z=z,time.grid=time.grid,B=B,nbasis=5,alpha=alpha)
```

`pencoxfrail` *Regularization in Cox Frailty Models.*

Description

A regularization approach for Cox Frailty Models by penalization methods is provided.

Usage

```
pencoxfrail(fix=formula, rnd=formula, vary.coef=formula, data, xi,
              adaptive.weights = NULL, control = list())
```

Arguments

`fix`	a two-sided linear formula object describing the unpenalized fixed (time-constant) effects part of the model, with the response on the left of a ~ operator and the terms, separated by + operators, on the right. The response must be a survival object as returned by the `Surv` function.
`rnd`	a two-sided linear formula object describing the random-effects part of the model, with the grouping factor on the left of a ~ operator and the random terms, separated by + operators, on the right.
`vary.coef`	a one-sided linear formula object describing the time-varying effects part of the model, with the time-varying terms, separated by + operators, on the right side of a ~ operator.
`data`	the data frame containing the variables named in the three preceding `formula` arguments.
`xi`	the overall penalty parameter that controls the strenght of both penalty terms in $\xi \cdot J(\zeta, \alpha)$ and, hence, controls the overall amount of smoothness (up to constant effects) and variable selection for a given proportion ζ. The optimal penalty parameter is a tuning parameter of the procedure that has to be determined, e.g. by K-fold cross validation. (See details or the quick demo for an example.)
`adaptive.weights`	a two-column matrix of adaptive weights passed to the procedure; the first column contains the weights $w_{\Delta,k}$, the second column the weights v_k from $\xi \cdot J(\zeta, \alpha)$. If no adaptive weights are specified all weights are set to one. The recommended strategy is to first fit an unpenalized model (i.e. $\xi = 0$) and then use the obtained adaptive weights (see value section) when fitting the model for all other combinations of ξ and ζ.
`control`	a list of control values for the estimation algorithm to replace the default values returned by the function `pencoxfrailControl`. Defaults to an empty list.

Details

The `pencoxfrail` algorithm is designed to investigate the effect structure in the Cox frailty model, which is a widely used model that accounts for heterogeneity in survival data. Since in survival models one has to account for possible variation of the effect strength over time the selection of the relevant features distinguishes between the folllowing cases: covariates can have time-varying effects, can have time-constant effects or be irrelevant. For this purpose, the following specific penality is applied on the vectors of B-spline coefficients α_k, assuming $k = 1, ..., r$ different, potentially time-varying effects, each expanded in M B-spline basis functions:

$$\xi \cdot J(\zeta, \alpha) = \xi \left(\zeta \sum_{k=1}^{r} \psi \, w_{\Delta,k} ||\Delta_M \alpha_k||_2 + (1-\zeta) \sum_{k=1}^{r} \phi \, v_k ||\alpha_k||_2 \right)$$

This penalty is able to distinguish between these types of effects to obtain a sparse representation that includes the relevant effects in a proper form.

The penalty is depending on two tuning parameters, ξ and ζ, which have to be determined by a suitable technique, e.g. by (2-dimensional) K-fold cross validation.

The first term of the penalty controls the smoothness of the time-varying covariate effects, whereby for values of ξ and ζ large enough, all differences $(\alpha_{k,l} - \alpha_{k,l-1}), l = 2, ..., M$, are removed from the model, resulting in constant covariate effects. As the B-splines of each variable with varying coefficients sum up to one, a constant effect is obtained if all spline coefficients are set equal. Hence, the first penalty term does not affect the spline's global level. The second term penalizes all spline coefficients belonging to a single time-varying effect in the way of a group LASSO and, hence, controls the selection of covariates.

Package:	pencoxfrail
Type:	Package
Version:	1.1.2
Date:	2023-08-25
License:	GPL-2
LazyLoad:	yes

for loading a dataset type data(nameofdataset)

Value

Generic functions such as `print`, `predict`, `plot` and `summary` have methods to show the results of the fit.

The `predict` function uses also estimates of random effects for prediction, if possible (i.e. for known subjects of the grouping factor). Either the survival stepfunction or the baseline hazard (not cumulative!) can be calculated by specifying one of two possible methods: `method=c("hazard","survival")`. By default, for each new subject in `new.data` an individual stepfunction is calculated on a prespecified time grid, also accounting for covariate changes over time. Alternatively, for `new.data` a single vector of a specific (time-constant) covariate combination can be specified.

Usage: `predict(pencoxfrail.obj,new.data,time.grid,method=c("hazard","survival"))`

The `plot` function plots all time-varying effects, including the baseline hazard.

`call`	a list containing an image of the `pencoxfrail` call that produced the object.

`baseline`	a vector containing the estimated B-spline coefficients of the baseline hazard. If the covariates corresponding to the time-varying effects are centered (and standardized, see `pencoxfrailControl`), the coefficients are transformed back to the original scale.
`time.vary`	a vector containing the estimated B-spline coefficients of all time-varying effects. If the covariates corresponding to the time-varying effects are standardized (see `pencoxfrailControl`) the coefficients are transformed back to the original scale.
`coefficients`	a vector containing the estimated fixed effects.
`ranef`	a vector containing the estimated random effects.
`Q`	a scalar or matrix containing the estimates of the random effects standard deviation or variance-covariance parameters, respectively.
`Delta`	a matrix containing the estimates of fixed and random effects (columns) for each iteration (rows) of the main algorithm (i.e. before the final re-estimation step is performed, see details).
`Q_long`	a list containing the estimates of the random effects variance-covariance parameters for each iteration of the main algorithm.
`iter`	number of iterations until the main algorithm has converged.
`adaptive.weights`	If $\xi = 0$, a two-column matrix of adaptive weights is calculated; the first column contains the weights $w_{\Delta,k}$, the second column the weights v_k from $\xi \cdot J(\zeta, \alpha)$. If $\xi > 0$, the adaptive weights that have been used in the function's argument are displayed.
`knots`	vector of knots used in the B-spline representation.
`Phi.big`	large B-spline design matrix corresponding to the baseline hazard and all time-varying effects. For the time-varying effects, the B-spline functions (as a function of time) have already been multiplied with their associated covariates.
`time.grid`	the time grid used in when approximating the (Riemann) integral involved in the model's full likelihood.
`m`	number of metric covariates with time-varying effects.
`m2`	number of categorical covariates with time-varying effects.

Author(s)

Andreas Groll <`groll@math.lmu.de`>

References

Groll, A., T. Hastie and G. Tutz (2017). Selection of Effects in Cox Frailty Models by Regularization Methods. *Biometrics* 73(3): 846-856.

See Also

`pencoxfrailControl`, `Surv`, `pbc`

Examples

```
## Not run:
data(lung)

# remove NAs
lung <- lung[!is.na(lung$inst),]

# transform inst into factor variable
lung$inst <- as.factor(lung$inst)

# Random institutional effect
fix.form <- as.formula("Surv(time, status) ~ 1")
vary.coef <- as.formula("~ age")

pen.obj <- pencoxfrail(fix=fix.form,vary.coef=vary.coef, rnd = list(inst=~1),
              data=lung, xi=10,control=list(print.iter=TRUE))

# show fit
plot(pen.obj)

# predict survival curve of new subject, institution 1 and up to time 500
pred.obj <- predict(pen.obj,newdata=data.frame(inst=1,time=NA,status=NA,age=26),
              time.grid=seq(0,500,by=1))

# plot predicted hazard function
plot(pred.obj$time.grid,pred.obj$haz,type="l",xlab="time",ylab="hazard")

# plot predicted survival function
plot(pred.obj$time.grid,pred.obj$survival,type="l",xlab="time",ylab="survival")

# see also demo("pencoxfrail-pbc")

## End(Not run)
```

`pencoxfrailControl` *Control Values for* `pencoxfrail` *fit*

Description

The values supplied in the function call replace the defaults and a list with all possible arguments is returned. The returned list is used as the `control` argument to the `pencoxfrail` function.

Usage

```
pencoxfrailControl(start = NULL, q_start = NULL, conv.eps = 1e-4,
                             standardize = FALSE, center = FALSE,
                             smooth=list(nbasis = 6, penal = 0.1),
                             ridge.pen = 1e-4, print.iter = FALSE,
                             max.iter = 100, c.app = 1e-6, zeta = 0.5,
                             exact = 1e-2, xr = NULL, ...)
```

Arguments

`start`	a vector of suitable length containing starting values for the spline-coefficients of the baseline hazard and the time-varying effects, followed by the fixed and random effects. The correct ordering is important. Default is a vector full of zeros.
`q_start`	a scalar or matrix of suitable dimension, specifying starting values for the random-effects variance-covariance matrix. Default is a scalar 0.1 or diagonal matrix with 0.1 in the diagonal, depending on the dimension of the random effects.
`conv.eps`	controls the speed of convergence. Default is 1e-4.
`center`	logical. If true, the covariates corresponding to the time-varying effects will be centered. Default is FALSE (and centering is only recommended if really necessary; it can also have a strong effect on the baseline hazard, in particular, if a strong penalty is selected).
`standardize`	logical. If true, the the covariates corresponding to the time-varying effects will be scaled to a variance equal to one (*after* possible centering). Default is FALSE.
`smooth`	a list specifying the number of basis functions `nbasis` (used for the baseline hazard and all time-varying effects) and the smoothness penalty parameter `penal`, which is only applied to the baseline hazard. All time-varying effects are penalized by the specific double-penalty $\xi\cdot J(\zeta,\alpha)$ (see `pencoxfrail`), which is based on the overall penalty parameter ξ (specified in the main function `pencoxfrail`) and on the weighting between the two penalty parts ζ. The degree of the B-splines is fixed to be three (i.e. cubic splines).
`ridge.pen`	On all time-varying effects (except for the baseline hazard) a slight ridge penalty is applied on the second order differences of the corresponding spline coefficients to stabilize estimation. Default is 1e-4.
`print.iter`	logical. Should the number of iterations be printed? Default is FALSE.
`max.iter`	the number of iterations for the final Fisher scoring re-estimation procedure. Default is 200.
`c.app`	The parameter controlling the exactness of the quadratic approximations of the penalties. Default is 1e-6.
`zeta`	The parameter controlling the weighting between the two penalty parts in the specific double-penalty $\xi \cdot J(\zeta,\alpha)$ (see `pencoxfrail`). Default is 0.5.
`exact`	controls the exactness of the (Riemann) integral approximations. Default is 1e-2.
`xr`	maximal time point that is regarded. Default is NULL and the maximal event or censoring time point in the data is used.
`...`	Futher arguments to be passed.

Value

a list with components for each of the possible arguments.

Author(s)

Andreas Groll <`groll@math.lmu.de`>

See Also

pencoxfrail

Examples

```
# Use different weighting of the two penalty parts
# and lighten the convergence criterion
pencoxfrailControl(zeta=0.3, conv.eps=1e-3)
```

Index