# Package 'peperr'

February 20, 2015

**Type** Package

**Title** Parallelised Estimation of Prediction Error

**Date** 2013

**Version** 1.1-7

**Depends** snowfall, survival

**Suggests** CoxBoost, locfit, penalized, codetools

**Author** Christine Porzelius, Harald Binder

**Maintainer** Christine Porzelius <cp@imbi.uni-freiburg.de>

**Description** Package peperr is designed for prediction error estimation through resampling techniques, possibly accelerated by parallel execution on a compute cluster. Newly developed model fitting routines can be easily incorporated.

**License** GPL (>= 2)

**NeedsCompilation** yes

**Repository** CRAN

**Date/Publication** 2013-04-08 16:10:47

## R topics documented:

---

`aggregation.brier` *Determine the Brier score for a fitted model*

---

## Description

Evaluate the Brier score, i.e. prediction error, for a fitted model on new data. To be used as argument `aggregation.fun` in `peperr` call.

## Usage

```
aggregation.brier(full.data=NULL, response, x, model, cplx=NULL,  type=c("apparent", "noinf"),
   fullsample.attr = NULL, ...)
```

## Arguments

| | |
|---|---|
| `full.data` | passed from `peperr`, but not used for calculation of the Brier score. |
| `response` | vector of binary response. |
| `x` | `n*p` matrix of covariates. |
| `model` | model fitted as returned by a `fit.fun`, as used in a call to `peperr`. |
| `cplx` | passed from `peperr`, but not necessary for calculation of the Brier score. |
| `type` | character. |
| `fullsample.attr` | passed from `peperr`, but not necessary for calculation of the Brier score. |
| `...` | additional arguments, passed to `predict` function. |

## Details

The empirical Brier score is the mean of the squared difference of the risk prediction and the true value of all observations and takes values between 0 and 1, where small values indicate good prediction performance of the risk prediction model.

## Value

Scalar, indicating the empirical Brier score.

`aggregation.misclass` *Determine the missclassification rate for a fitted model*

## Description

Evaluate the misclassification rate, i.e. prediction error, for a fitted model on new data. To use as argument `aggregation.fun` in `peperr` call.

## Usage

```
aggregation.misclass(full.data=NULL, response, x, model, cplx=NULL,  type=c("apparent", "noinf"),
   fullsample.attr = NULL, ...)
```

## Arguments

| | |
|---|---|
| `full.data` | passed from `peperr`, but not used for calculation of the misclassification rate. |
| `response` | vector of binary response. |
| `x` | `n*p` matrix of covariates. |
| `model` | model fitted with `fit.fun`. |
| `cplx` | passed from `peperr`, but not necessary for calculation of the misclassification rate. |
| `type` | character. |
| `fullsample.attr` | passed from `peperr`, but not necessary for calculation of the misclassification rate. |
| `...` | additional arguments, passed to `predict` function. |

## Details

Misclassification rate is the ratio of observations for which prediction of response is wrong.

## Value

Scalar, indicating the misclassification rate.

aggregation.pmpec *Determine the prediction error curve for a fitted model*

## Description

Interface to pmpec, for conforming to the structure required by the argument aggregation.fun in peperr call. Evaluates the prediction error curve, i.e. the Brier score tracked over time, for a fitted survival model.

## Usage

```
aggregation.pmpec(full.data, response, x, model, cplx=NULL, times = NULL,
   type=c("apparent", "noinf"), fullsample.attr = NULL, ...)
```

## Arguments

| | |
|---|---|
| full.data | data frame with full data set. |
| response | Either a survival object (with Surv(time, status), where time is an n-vector of censored survival times and status an n-vector containing event status, coded with 0 and 1) or a matrix with columns time containing survival times and status containing integers, where 0 indicates censoring, 1 the interesting event and larger numbers other competing risks. |
| x | n*p matrix of covariates. |
| model | survival model as returned by fit.fun as used in call to peperr. |
| cplx | numeric, number of boosting steps or list, containing number of boosting steps in argument stepno. |
| times | vector of evaluation time points. If given, used as well as in calculation of full apparent and no-information error as in resampling procedure. Not used if fullsample.attr is specified. |
| type | character. |
| fullsample.attr | vector of evaluation time points, passed in resampling procedure. Either user-defined, if times were passed as args.aggregation, or the determined time points from the aggregation.fun call with the full data set. |
| ... | additional arguments passed to pmpec call. |

## Details

If no evaluation time points are passed, they are generated using all uncensored time points if their number is smaller than 100, or 100 time points up to the 95% quantile of the uncensored time points are taken.

pmpec requires a predictProb method for the class of the fitted model, i.e. for a model of class class predictProb.class.

## Value

A matrix with one row. Each column represents the estimated prediction error of the fit at the time points.

## See Also

peperr, predictProb, pmpec

---

`complexity.ipec.CoxBoost` *Interface function for complexity selection for CoxBoost via integrated prediction error curve and the bootstrap*

---

## Description

Determines the number of boosting steps for a survival model fitted by CoxBoost via integrated prediction error curve (IPEC) estimates, conforming to the calling convention required by argument `complexity` in `peperr` call.

## Usage

```
complexity.ipec.CoxBoost(response, x, boot.n.c = 10, boost.steps = 100,
   eval.times = NULL, smooth = FALSE, full.data, ...)

complexity.ripec.CoxBoost(response, x, boot.n.c = 10, boost.steps = 100,
   eval.times = NULL, smooth = FALSE, full.data, ...)
```

## Arguments

| | |
|---|---|
| `response` | a survival object (with `Surv(time, status)`). |
| `x` | `n*p` matrix of covariates. |
| `boot.n.c` | number of bootstrap samples. |
| `boost.steps` | maximum number of boosting steps, i.e. number of boosting steps is selected out of interval (1, boost.steps). |
| `eval.times` | vector of evaluation time points. |
| `smooth` | logical. Shall prediction error curve be smoothed by local polynomial regression before integration? |
| `full.data` | Data frame containing response and covariates of the full data set. |
| `...` | additional arguments passed to `CoxBoost` call. |

## Details

Plotting the .632+ estimator for each time point given in `eval.times` results in a prediction error curve. A summary measure can be obtained by integrating over time. `complexity.ripec.CoxBoost` computes a Riemann integral, while `complexity.ipec.CoxBoost` uses a Lebesgue-like integral taking Kaplan-Meier estimates as weights. The number of boosting steps of the interval (0, `boost.steps`), for which the minimal IPEC is obtained, is returned.

## Value

Scalar value giving the number of boosting steps.

## See Also

peperr, CoxBoost

---

`complexity.LASSO` *Interface for selection of optimal parameter for lasso fit*

---

## Description

Determines the optimal value for tuning parameter lambda for a regression model with lasso penalties via cross-validation. Conforming to the calling convention required by argument `complexity` in `peperr` call.

## Usage

```
complexity.LASSO(response, x, full.data, ...)
```

## Arguments

| | |
|---|---|
| `response` | a survival object (`Surv(time, status)`). |
| `x` | `n*p` matrix of covariates. |
| `full.data` | data frame containing response and covariates of the full data set. |
| `...` | additional arguments passed to `optL1` of package **penalized** call. |

## Details

Function is basically a wrapper around `optL1` of package **penalized**. Calling `peperr`, default arguments of `optL1` can be changed by passing a named list containing these as argument `args.complexity`.

## Value

Scalar value giving the optimal value for lambda.

## See Also

peperr, optL1

`complexity.mincv.CoxBoost` *Interface for CoxBoost selection of optimal number of boosting steps via cross-validation*

## Description

Determines the number of boosting steps for a survival model fitted by CoxBoost via cross-validation, conforming to the calling convention required by argument `complexity` in `peperr` call.

## Usage

```
complexity.mincv.CoxBoost(response, x, full.data, ...)
```

## Arguments

| | |
|---|---|
| `response` | a survival object (`Surv(time, status)`). |
| `x` | `n*p` matrix of covariates. |
| `full.data` | data frame containing response and covariates of the full data set. |
| `...` | additional arguments passed to `cv.CoxBoost` call. |

## Details

Function is basically a wrapper around `cv.CoxBoost` of package `CoxBoost`. A K-fold cross-validation (default K=10) is performed to search the optimal number of boosting steps, per default in the interval (0, `maxstepno`=100). The number of boosting steps with minimum mean partial log-likelihood is returned. Calling `peperr`, the default arguments of `cv.CoxBoost` can be changed by passing a named list containing these as argument `args.complexity`.

## Value

Scalar value giving the optimal number of boosting steps.

## See Also

`peperr`, `cv.CoxBoost`

`extract.fun` *Extract functions, libraries and global variables to be loaded onto a compute cluster*

## Description

Automatic extraction of functions, libraries and global variables employed passed functions. Designed for `peperr` call, see Details section there.

## Usage

```
extract.fun(funs = NULL)
```

## Arguments

| | |
|---|---|
| `funs` | list of function names. |

## Details

This function is necessary for compute cluster situations where for computation on nodes required functions, libraries and variables have to be loaded explicitly on each node. Avoids loading of whole global environment which might include the unnecessary loading of huge data sets.

It might have problems in some cases, especially it is not able to extract the library of a function that has no namespace. Similarly, it can only extract a required library if it is loaded, or if the function contains a require or library call.

## Value

list containing

| | |
|---|---|
| `packages` | vector containing quoted names of libraries |
| `functions` | vector containing quoted names of functions |
| `variables` | vector containing quoted names of global variables |

## See Also

`peperr`

## Examples

```
# 1. Simplified example for illustration
## Not run:
library(CoxBoost)
a <- function(){
# some calculation
}

b<-function(){
```

```
# some other calculation
x <- cv.CoxBoost()
# z is global variable
y <- a(z)
}

# list with packages, functions and variables required for b:
extract.fun(list(b))

# 2. As called by default in peperr example
extract.fun(list(fit.CoxBoost, aggregation.pmpec))

## End(Not run)
```

---

`fit.CoxBoost` *Interface function for fitting a CoxBoost model*

---

## Description

Interface for fitting survival models by `CoxBoost`, conforming to the requirements for argument `fit.fun` in `peperr` call.

## Usage

```
fit.CoxBoost(response, x, cplx, ...)
```

## Arguments

| | |
|---|---|
| `response` | a survival object (with `Surv(time, status)`). |
| `x` | `n*p` matrix of covariates. |
| `cplx` | number of boosting steps or list, containing number of boosting steps in argument `stepno` and penalty factor in argument `penalty.factor`. |
| `...` | additional arguments passed to `CoxBoost` call. |

## Details

Function is basically a wrapper around `CoxBoost` of package **CoxBoost**. A Cox proportional hazards model is fitted by componentwise likelihood based boosting, especially suited for models with many covariates and few observations.

## Value

CoxBoost object

## See Also

`peperr`, `CoxBoost`

## fit.coxph *Interface function for fitting a Cox proportional hazards model*

### Description

Interface for fitting survival models by Cox proporional hazards model, conforming to the requirements for argument `fit.fun` in `peperr` call.

### Usage

```
fit.coxph(response, x, cplx, ...)
```

### Arguments

| | |
|---|---|
| `response` | a survival object (with `Surv(time, status)`). |
| `x` | `n*p` matrix of covariates. |
| `cplx` | not used. |
| `...` | additional arguments passed to `coxph` call. |

### Details

Function is basically a wrapper around `coxph` of package **survival**.

### Value

CoxBoost object

### See Also

`peperr`, `coxph`

## fit.LASSO *Interface function for fitting a generalised linear model with the lasso*

### Description

Interface for fitting survival models with the lasso, conforming to the requirements of argument `fit.fun` in `peperr` call.

### Usage

```
fit.LASSO(response, x, cplx, ...)
```

### Arguments

| | |
|---|---|
| response | response. Could be numeric vector for linear regression, Surv object for Cox regression or a binary vector for logistic regression. |
| x | n*p matrix of covariates. |
| cplx | LASSO penalty. lambda1 of penalized call. |
| ... | additional arguments passed to penalized call. |

### Details

Function is basically a wrapper around function penalized of package penalized.

### Value

penfit object

### See Also

peperr, penalized

---

## ipec *Integrated prediction error curve*

---

### Description

Summary measures of prediction error curves

### Usage

```
ipec(pe, eval.times, type=c("Riemann", "Lebesgue", "relativeLebesgue"), response=NULL)
```

### Arguments

| | |
|---|---|
| pe | prediction error at different time points. Vector of length of eval.times or matrix (columns correspond to evaluation time points, rows to different prediction error estimates) |
| eval.times | evalutation time points |
| type | type of integration. 'Riemann' estimates Riemann integral, 'Lebesgue' uses the probability density as weights, while 'relativeLebesgue' delivers the difference to the null model (using the same weights as for 'Lebesgue'). |
| response | survival object (Surv(time, status)), required only if type is 'Lebesgue' or 'relativeLebesgue' |

### Details

For survival data, prediction error at each evaluation time point can be extracted of a peperr object by function perr. A summary measure can then be obtained via intgrating over time. Note that the time points used for evaluation are stored in list element attribute of the peperr object.

## Value

ipec — Value of integrated prediction error curve. Integer or vector, if pe is vector or matrix, respectively, i.e. one entry per row of the passed matrix.

## See Also

perr

## Examples

```
## Not run:
n <- 200
p <- 100
beta <- c(rep(1,10),rep(0,p-10))
x <- matrix(rnorm(n*p),n,p)
real.time <- -(log(runif(n)))/(10*exp(drop(x %*% beta)))
cens.time <- rexp(n,rate=1/10)
status <- ifelse(real.time <= cens.time,1,0)
time <- ifelse(real.time <= cens.time,real.time,cens.time)

# Example:
# Obtain prediction error estimate fitting a Cox proportional hazards model
# using CoxBoost
# through 10 bootstrap samples
# with fixed complexity 50 and 75
# and aggregate using prediction error curves
peperr.object <- peperr(response=Surv(time, status), x=x,
   fit.fun=fit.CoxBoost, complexity=c(50, 75),
   indices=resample.indices(n=length(time), method="sub632", sample.n=10))
# 632+ estimate for both complexity values at each time point
prederr <- perr(peperr.object)
# Integrated prediction error curve for both complexity values
ipec(prederr, eval.times=peperr.object$attribute, response=Surv(time, status))

## End(Not run)
```

---

# peperr *Parallelised Estimation of Prediction Error*

---

## Description

Prediction error estimation for regression models via resampling techniques. Potentially parallelised, if compute cluster is available.

## Usage

```
peperr(response, x,
indices = NULL,
fit.fun, complexity = NULL, args.fit = NULL, args.complexity = NULL,
```

```
parallel = NULL, cpus = 2, clustertype=NULL, clusterhosts=NULL,
noclusterstart = FALSE, noclusterstop=FALSE,
aggregation.fun=NULL, args.aggregation = NULL,
load.list = extract.fun(list(fit.fun, complexity, aggregation.fun)),
load.vars = NULL, load.all = FALSE,
trace = FALSE, debug = FALSE,
peperr.lib.loc=NULL,
        RNG=c("RNGstream", "SPRNG", "fixed", "none"), seed=NULL,
        lb=FALSE, sr=FALSE, sr.name="default", sr.restore=FALSE)
```

## Arguments

| | |
|---|---|
| response | Either a survival object (with `Surv(time, status)`, where time is an n-vector of censored survival times and status an n-vector containing event status, coded with 0 and 1) or a matrix with columns `time` containing survival times and `status` containing integers, where 0 indicates censoring, 1 the interesting event and larger numbers other competing risks. In case of binary response, vector with entries 0 and 1. |
| x | n*p matrix of covariates. |
| indices | named list, with two elements (both expected to be lists) `sample.index`, containing the vector of indices of observations used to fit the model, and list `not.in.sample`, containing the vector of indices of observations used for assessment. One list entry per split. Function `resample.indices` provides the most common resampling methods. If argument `indices` is not specified (default), the indices are determined as follows: If number of observations in the passed data matrix is smaller than number of covariates, 500 bootstrap samples without replacement are generated ("subsampling"), else 500 bootstrap samples with replacement. |
| fit.fun | function returning a fitted model, see Details. |
| complexity | if the choice of a complexity parameter is necessary, for example the number of boosting steps in boosting techniques, a function returning complexity parameter for model fitted with fit.fun, see Details. Alternatively, one explicit value for the complexity or a vector of values can be passed. In the latter case, the model fit is carried out for each of the complexity parameters. Alternatively, a named list can be passed, if complexity is a tuple of different parameter values. |
| args.fit | named list of arguments to be passed to the function given in `fit.fun`. |
| args.complexity | if `complexity` is a function, a named list of arguments to be passed to this function. |
| parallel | the default setting corresponds to the case that sfCluster is used or if R runs sequential, i.e. without any parallelisation. If sfCluster is used, settings from sfCluster commandline call are taken, i.e. the required number of nodes has to be specified as option of the sfCluster call (and not using argument `cpus`). If another cluster solution (specified by argument `clustertype`) shall be used, a cluster with `cpus` CPUs is started if `parallel=TRUE`. `parallel=FALSE` switches back to sequential execution. See Details. |

| | |
|---|---|
| cpus | number of nodes, i.e., number of parallel running R processes, to be set up in a cluster, if not specified by commandline call. Only needed if parallel=TRUE. |
| clustertype | type of cluster, character. 'SOCK' for socket cluster, 'MPI', 'PVM' or 'NWS'. Only considered if parallel=TRUE. If so, a socket cluster, which does not require any additional installation, is started as default. |
| clusterhosts | host list for socket and NWS clusters, if parallel=TRUE. Has to be specified only if using more than one machine. |
| noclusterstart | if function is used in already parallelised code. If set to TRUE, no cluster is initialised even if a compute cluster is available and function works in sequential mode. Additionally usable if calls on the slaves should be executed before calling function peperr, for example to load data on slaves, see Details. |
| noclusterstop | if TRUE, cluster stop is suppressed. Useful for debugging of sessions on slaves. Note that the next peperr call forces cluster stop, except if called with noclusterstart=TRUE. |
| aggregation.fun | function that evaluates the prediction error for a model fitted by the function given in fit.fun, see Details. If not specified, function aggregation.pmpec is taken if response is survival object, in case of binary response function aggregation.brier. |
| args.aggregation | named list of arguments to be passed to the function given in argument aggregation.fun. |
| load.list | a named list with element packages, functions and variables containing quoted names of libraries, functions and global variables required for computation on cluster nodes. The default extracts automatically the libraries, functions and global variables of the, potentially user-defined, functions fit.fun, complexity and aggregation.fun, see function extract.fun. Can be set to NULL, e.g. if no libraries, functions and variables are needed. Alternatively, use argument load.all. See Details. |
| load.vars | a named list with global variables required for computation on cluster nodes. See Details. Relict, global variabels can now be passed as list element variables of argument load.list. |
| load.all | logical. If set to TRUE, all variables, functions and libraries of the current global environment are loaded on cluster nodes. See Details. |
| trace | logical. If TRUE, output about the current execution step is printed (if running parallel: printed on nodes, that means not visible in master R process, see Details). |
| debug | if TRUE, information concerning export of variables is given. |
| peperr.lib.loc | location of package **peperr** if not in standard library search path (.libPaths()), to be specified for loading **peperr** onto the cluster nodes. |
| RNG | type of RNG. "fixed" requires a specified seed. "RNGstream" and "SPRNG" use default seeds, if not specified. See Details. |
| seed | seed to allow reproducibility of results. Only considered if argument RNG is not "none". See Details. |
| lb | if TRUE and a compute cluster is used, computation of slaves is executed load balanced. See Details. |

| | |
|---|---|
| sr | if TRUE, intermediate results are saved. If execution is interrupted, they can be restored by setting argument sr.restore to TRUE. See documentation of package **snowfall** for details |
| sr.name | if sr is set to TRUE and more than one computation runs simultaneously, unique names need to be used. |
| sr.restore | if sr is set to TRUE, an interrupted computation is restarted. |

## Details

Validation of new model fitting approaches requires the proper use of resampling techniques for prediction error estimation. Especially in high-dimensional data situations the computational demand might be huge. peperr accelerates computation through automatically parallelisation of the resampling procedure, if a compute cluster is available. A noticeable speed-up is reached even when using a dual-core processor.

Resampling based prediction error estimation requires for each split in training and test data the following steps: a) selection of model complexity (if desired), using the training data set, b) fitting the model with the selected (or a given) complexity on the training set and c) measurement of prediction error on the corresponding test set.

Functions for fitting the model, determination of model complexity, if required by the fitting procedure, and aggregating the prediction error are passed as arguments fit.fun, complexity and aggregation.fun. Already available functions are

for model fit: fit.CoxBoost, fit.coxph, fit.LASSO, fit.rsf_mtry

to determine complexity: complexity.mincv.CoxBoost, complexity.ipec.CoxBoost, complexity.LASSO, complexity.ipec.rsf_mtry

to aggregate prediction error: aggregation.pmpec, aggregation.brier, aggregation.misclass

Function peperr is especially designed for evaluation of newly developed model fitting routines. For that, own routines can be passed as arguments to the peperr call. They are incorporated as follows (also compare existing functions, as named above):

1. Model fitting techniques, which require selection of one or more complexity parameters, often provide routines based on cross-validation or similar to determine this parameter. If this routine is already at hand, the complexity function needed for the peperr call is not more than a wrapper around that, which consists of providing the data in the required way, calling the routine and return the selected complexity value(s).
2. For a given model fitting routine the fitting function, which is passed to the peperr call as argument fit.fun, is not more than a wrapper around that. Explicitly, response and matrix of covariates have to be transformed to the required form, if necessary, the routine is called with the passed complexity value, if required, and the fitted prediction model is returned.
3. Prediction error is estimated using a fitted model and a data set, by any kind of comparison of the true and the predicted response values. In case of survival response, apparent error (type apparent), which means that the prediction error is estimated in the same data set as used for model fitting, and no-information error (type noinf), which calculates the prediction error in permuted data, have to be provided. Note that the aggregation function returns the error with an additional attribute called addattr. The evaluation time points have to be stored there to allow later access.

4. In case of survival response, the user may additionally provide a function for partial log likelihood calculation, if he uses an own function for model fit, called `PLL.class`. If prediction error curves are used for aggregation (`aggregation.pmpec`), a predictProb method has to be provided, i.e. for each model of class `class` `predictProb.class`, see there.

Concerning parallelisation, there are three possibilities to run `peperr`:

- Start R on commandline with sfCluster and preferred options, for example number of cpus. Leave the three arguments `parallel`, `clustertype` and `nodes` unchanged.
- Use any other cluster solution supported by **snowfall**, i.e. LAM/MPI, socket, PVM, NWS (set argument `clustertype`). Argument `parallel` has to be set to TRUE and number of cpus can be chosen by argument `nodes`)
- If no cluster is used, R works sequentially. Keep `parallel=NULL`. No parallelisation takes place and therefore no speed up can be obtained.

In general, if `parallel=NULL`, all information concerning the cluster set-up is taken from commandline, else, it can be specified using the three arguments `parallel`, `clustertype`, `nodes`, and, if necessary, `clusterhosts`.

sfCluster is a Unix tool for flexible and comfortable managment of parallel R processes. It is available at www.imbi.uni-freiburg.de/parallel, together with detailed information. However, **peperr** is usable with any other cluster solution supported by **snowfall**, i.e. sfCluster has not to be installed to use package **peperr**. Note that this may require cluster handling by the user, e.g. manually shut down with 'lamhalt' on commandline for `type="MPI"`. But, using a socket cluster (argument `parallel=TRUE` and `clustertype="SOCK"`), does not require any extra installation.

Note that the run time cannot speed up anymore if the number of nodes is chosen higher than the number of passed training/test samples plus one, as parallelisation takes place in the resampling procedure and one additional run is used for computation on the full sample.

If not running in sequential mode, a specified number of R processes called nodes is spawned for parallel execution of the resampling procedure (see above). This requires to provide all variables, functions and libraries necessary for computation on each of these R processes, so explicitly all variables, functions and libraries required by the, potentially user-defined, functions `fit.fun`, `complexity` and `aggregation.fun`. The simplest possibility is to load the whole content of the global environment on each node and all loaded libraries. This is done by setting argument `load.all=TRUE`. This is not the default, as a huge amount of data is potentially loaded to each node unnecessarily. Function `extract.fun` is provided to extract the functions and libraries needed, automatically called at each call of function `peperr`. Note that all required libraries have to be located in the standard library search path (obtained by `.libPaths()`). Another alternative is to load required data manually on the slaves, using **snowfall** functions `sfLibrary`, `sfExport` and `sfExportAll`. Then, argument `noclusterstart` has to be switched to TRUE. Additionally, argument `load.list` could be set to NULL, to avoid potentially overwriting of functions and variables loaded to the cluster nodes automatically.

Note that a `set.seed` call before calling function `peperr` is not sufficient to allow reproducibility of results when running in parallel mode, as the slave R processes are not affected as they are own R instances. `peperr` provides two possibilities to make results reproducible:

- Use `RNG="RNGstream"` or `RNG="SPRNG"`. Independent parallel random number streams are initialized on the cluster nodes, using function `sfClusterSetupRNG` of package **snowfall**. A

seed can be specified using argument seed, else the default values are taken. A set.seed call on the master is required additionally and argument lb=FALSE, see below.

- If RNG="fixed", a seed has to be specified. This can be either an integer or a vector of length number of samples +2. In the second case, the first entry is used for the main R process, the next number of samples ones for each sample run (in parallel execution mode on slave R processes) and the last one for computation on full sample (as well on slave R process in parallel execution mode). Passing integer x is equivalent to passing vector x+(0:(number of samples+1)). This procedure allows reproducibility in any case, i.e. also if the number of parallel processes changes as well as in sequential execution.

Load balancing (argument lb) means, that a slave gets a new job immediately after the previous is finished. This speeds up computation, but may change the order of jobs. Due to that, results are only reproducible, if RNG="fixed" is used.

## Value

Object of class peperr

| | |
|---|---|
| indices | list of resampling indices. |
| complexity | passed complexity. If argument complexity not specified, 0. |
| selected.complexity | selected complexity for the full data set, if complexity was passed as function. Else equal to value complexity. |
| response | passed response. |
| full.model.fit | List, one entry per complexity value. Fitted model of the full data set by passed fit.fun. |
| full.apparent | full apparent error of the full data set. Matrix: One row per complexity value. In case of survival response, columns correspond to evaluation timepoints, which are returned in value attribute. |
| noinf.error | No information error of the full data set, i. e. evaluation in permuted data. Matrix: One row per complexity value. Columns correspond to evaluation timepoints, which are returned in attribute. |
| attribute | if response is survival: Evaluation time points. Passed in args.aggregation or automatically determined by aggregation function. Otherwise, if available, extra attribute returned by aggregation function, else NULL, see Details. |
| sample.error | list. Each entry contains matrix of prediction error for one resampling test sample. One row per complexity value. |
| sample.complexity | vector of complexity values. Equals value complexity, if complexity value was passed explicitly, otherwise by function complexity selected complexity value for each resampling sample. If argument complexity not specified, 0. |
| sample.lipec | only, if response is survival. Lebesgue integrated prediction error curve for each sample. List with one entry per sample, each a matrix with one row per complexity value. |
| sample.pll | only, if response is survival and PLL.class function available. Predictive partial log likelihood for each sample. List with one entry per sample, each a matrix with one row per complexity value. |

`null.model.fit` only, if response is survival or binary. Fit of null model, i.e. fit without information of covariates. In case of survival response Kaplan-Meier, else logistic regression model.

`null.model` only, if response is survival or binary. Vector or scalar: Prediction error of the null model, in case of survival response at each evaluation time point.

`sample.null.model` list. Prediction error of the null model for one resampling test sample. Matrix, one row per complexity value.

## Author(s)

Christine Porzelius <cp@fdm.uni-freiburg.de>, Harald Binder

## References

Binder, H. and Schumacher, M. (2008) Adapting prediction error estimates for biased complexity selection in high-dimensional bootstrap samples. Statistical Applications in Genetics and Molecular Biology, 7:1.

Porzelius, C., Binder, H., Schumacher, M. (2008) Parallelised prediction error estimation for evaluation of high-dimensional models. Manuscript.

## See Also

`perr`, `resample.indices`, `extract.fun`

## Examples

```
# Generate survival data with 10 informative covariates
## Not run:
n <- 200
p <- 100
beta <- c(rep(1,10),rep(0,p-10))
x <- matrix(rnorm(n*p),n,p)
real.time <- -(log(runif(n)))/(10*exp(drop(x %*% beta)))
cens.time <- rexp(n,rate=1/10)
status <- ifelse(real.time <= cens.time,1,0)
time <- ifelse(real.time <= cens.time,real.time,cens.time)

# A: R runs sequential or R is started on commandline with desired options
# (for example using sfCluster: sfCluster -i --cpus=5)
# Example A1:
# Obtain prediction error estimate fitting a Cox proportional hazards model
# using CoxBoost
# through 10 bootstrap samples
# with fixed complexity 50 and 75
# and aggregate using prediction error curves (default setting)

peperr.object1 <- peperr(response=Surv(time, status), x=x,
   fit.fun=fit.CoxBoost, complexity=c(50, 75),
   indices=resample.indices(n=length(time), method="sub632", sample.n=10))
peperr.object1
```

```
# Diagnostic plots
plot(peperr.object1)

# Extraction of prediction error curves (.632+ prediction error estimate),
# blue line corresponds to complexity 50,
# red one to complexity 75
plot(peperr.object1$attribute,
   perr(peperr.object1)[1,], type="l", col="blue",
   xlab="Evaluation time points", ylab="Prediction error")
lines(peperr.object1$attribute,
   perr(peperr.object1)[2,], col="red")

# Example A2:
# As Example A1, but
# with complexity selected through a cross-validation procedure
# and extra argument 'penalty' passed to fit function and complexity function
peperr.object2 <- peperr(response=Surv(time, status), x=x,
   fit.fun=fit.CoxBoost, args.fit=list(penalty=100),
   complexity=complexity.mincv.CoxBoost, args.complexity=list(penalty=100),
   indices=resample.indices(n=length(time), method="sub632", sample.n=10),
   trace=TRUE)
peperr.object2

# Diagnostic plots
plot(peperr.object2)

# Example A3:
# As Example A2, but
# with extra argument 'times', specifying the evaluation times passed to aggregation.fun
# and seed, for reproducibility of results
# Note: set.seed() is required additional to argument 'seed',
# as function 'resample.indices' is used in peperr call.
set.seed(123)
peperr.object3 <- peperr(response=Surv(time, status), x=x,
   fit.fun=fit.CoxBoost, args.fit=list(penalty=100),
   complexity=complexity.mincv.CoxBoost, args.complexity=list(penalty=100),
   indices=resample.indices(n=length(time), method="sub632", sample.n=10),
   args.aggregation=list(times=seq(0, quantile(time, probs=0.9), length.out=100)),
   trace=TRUE, RNG="fixed", seed=321)
peperr.object3

# Diagnostic plots
plot(peperr.object3)

# B: R is started sequential, desired cluster options are given as arguments
# Example B1:
# As example A1, but using a socket cluster and 3 CPUs
peperr.object4 <- peperr(response=Surv(time, status), x=x,
   fit.fun=fit.CoxBoost, complexity=c(50, 75),
   indices=resample.indices(n=length(time), method="sub632", sample.n=10),
   parallel=TRUE, clustertype="SOCK", cpus=3)
```

```
## End(Not run)
```

---

perr *Prediction error estimates*

---

## Description

Extracts prediction error estimates from peperr objects.

## Usage

```
perr(peperrobject,
    type = c("632p", "632", "apparent", "NoInf", "resample", "nullmodel"))
```

## Arguments

| | |
|---|---|
| peperrobject | peperr object obtained by call to function peperr. |
| type | "632p" for the .632+ prediction error estimate (default), "632" for the .632 prediction error estimate. "apparent", "NoInf", "resample" and "nullmodel" return the apparent error, the no-information error, the mean sample error and the nullmodel fit, see Details. |

## Details

The .632 and the .632+ prediction error estimates are weighted combinations of the apparent error and bootstrap cross-validation error estimate, for survival data at given time points.

## Value

If type="632p" or type="632": Prediction error: Matrix, with one row per complexity value.

If type="apparent": Apparent error of the full data set. Matrix: One row per complexity value. In case of survival response, columns correspond to evaluation timepoints, which are given in attribute addattr.

If type="NoInf": No-information error of the full data set, i. e. evaluation in permuted data. Matrix: One row per complexity value. Columns correspond to evaluation timepoints, which are given in attribute addattr.

If type="resample": Matrix. Mean prediction error of resampling test samples, one row per complexity value.

If type="nullmodel": Vector or scalar: Null model prediction error, i.e. of fit without information of covariates. In case of survival response Kaplan-Meier estimate at each time point, if response is binary logistic regression model, else not available.

## References

Binder, H. and Schumacher, M. (2008) Adapting prediction error estimates for biased complexity selection in high-dimensional bootstrap samples. Statistical Applications in Genetics and Molecular Biology, 7:1.

Gerds, T. and Schumacher, M. (2007) Efron-type measures of prediction error for survival analysis. Biometrics, 63, 1283–1287.

Schumacher, M. and Binder, H., and Gerds, T. (2007) Assessment of Survival Prediction Models in High-Dimensional Settings. Bioinformatics, 23, 1768-1774.

## See Also

peperr, ipec

## Examples

```
## Not run:
n <- 200
p <- 100
beta <- c(rep(1,10),rep(0,p-10))
x <- matrix(rnorm(n*p),n,p)
real.time <- -(log(runif(n)))/(10*exp(drop(x %*% beta)))
cens.time <- rexp(n,rate=1/10)
status <- ifelse(real.time <= cens.time,1,0)
time <- ifelse(real.time <= cens.time,real.time,cens.time)

# Example:
# Obtain prediction error estimate fitting a Cox proportional hazards model
# using CoxBoost
# through 10 bootstrap samples
# with fixed complexity 50 and 75
# and aggregate using prediction error curves
peperr.object <- peperr(response=Surv(time, status), x=x,
   fit.fun=fit.CoxBoost, complexity=c(50, 75),
   indices=resample.indices(n=length(time), method="sub632", sample.n=10))
# 632+ estimate for both complexity values at each time point
perr(peperr.object)


## End(Not run)
```

---

PLL *Generic function for extracting the predictive partial log-likelihood*

---

## Description

Generic function for extracting th predictive partial log-likelihood from a fitted survival model.

### Usage

```
PLL(object, newdata, newtime, newstatus, ...)
```

### Arguments

| | |
|---|---|
| object | fitted model of class class. |
| newdata | n_new*p matrix of covariates. |
| newtime | n_new-vector of censored survival times. |
| newstatus | n_new-vector of event status, coded with 0 for censoring and 1, if an event occurred. |
| ... | additional arguments, for example complexity value, if necessary. |

### Details

The predictive partial log-likelihood measures the prediction performance of each model fitted in a boostrap sample, using the data not in this sample. Multiplying by (-2) leads to a deviance-like measure, which means that small values indicate good prediction performance.

peperr requires function PLL.class in case of survival response, for each model fit of class class. At the time, PLL.CoxBoost is available.

### Value

Vector of length n_new

---

## PLL.CoxBoost *Predictive partial log-likelihood for CoxBoost model fit*

---

### Description

Extracts the predictive partial log-likelihood from a CoxBoost model fit.

### Usage

```
## S3 method for class 'CoxBoost'
PLL(object, newdata, newtime, newstatus, complexity, ...)
```

### Arguments

| | |
|---|---|
| object | fitted model of class CoxBoost. |
| newdata | n_new*p matrix of covariates. |
| newtime | n_new-vector of censored survival times. |
| newstatus | n_new-vector of survival status, coded with 0 and .1 |
| complexity | complexity, either one value, which is number of boosting steps, or a list containing number of boosting steps in argument stepno. |
| ... | additional arguments, not used. |

## Details

Used by function peperr, if function fit.CoxBoost is used for model fit.

## Value

Vector of length n_new

---

PLL.coxph *Predictive partial log-likelihood for Cox poportional hazards model*

---

## Description

Extracts the predictive partial log-likelihood from a coxph model fit.

## Usage

```
## S3 method for class 'coxph'
PLL(object, newdata, newtime, newstatus, complexity, ...)
```

## Arguments

| | |
|---|---|
| object | fitted model of class coxph. |
| newdata | n_new*p matrix of covariates. |
| newtime | n_new-vector of censored survival times. |
| newstatus | n_new-vector of survival status, coded with 0 and .1 |
| complexity | not used. |
| ... | additional arguments, not used. |

## Details

Used by function peperr, if function fit.coxph is used for model fit.

## Value

Vector of length n_new

`plot.peperr` *Plot method for peperr object*

## Description

Plots, allowing to get a first impression of the prediction error estimates and to check complexity selection in bootstrap samples.

## Usage

```
## S3 method for class 'peperr'
plot(x, y, ...)
```

## Arguments

| | |
|---|---|
| `x` | `peperr` object. |
| `y` | not used. |
| `...` | additional arguments, not used. |

## Details

The plots provide a simple and fast overview of the results of the estimation of the prediction error through resampling. Which plots are shown depends on if complexity was selected, i.e., a function was passed in the `peperr` call for `complexity`, or explicitly passed. In case of survival response, prediction error curves are shown. In case of binary response, where one complexity value is passed explicitly, no plot is available. Especially in the case that complexity is selected in each bootstrap sample, these diagnostic plots help to check whether the resampling procedure works adequately and to detect specific problems due to high-dimensional data structures.

## Examples

```
## Not run:
n <- 200
p <- 100
beta <- c(rep(1,10),rep(0,p-10))
x <- matrix(rnorm(n*p),n,p)
real.time <- -(log(runif(n)))/(10*exp(drop(x %*% beta)))
cens.time <- rexp(n,rate=1/10)
status <- ifelse(real.time <= cens.time,1,0)
time <- ifelse(real.time <= cens.time,real.time,cens.time)

peperr.object1 <- peperr(response=Surv(time, status), x=x,
   fit.fun=fit.CoxBoost, complexity=c(50, 75),
   indices=resample.indices(n=length(time), method="sub632", sample.n=10))
plot(peperr.object1)

peperr.object2 <- peperr(response=Surv(time, status), x=x,
   fit.fun=fit.CoxBoost, args.fit=list(penalty=100),
```

```
   complexity=complexity.mincv.CoxBoost, args.complexity=list(penalty=100),
   indices=resample.indices(n=length(time), method="sub632", sample.n=10),
   trace=TRUE)
plot(peperr.object2)

peperr.object3 <- peperr(response=Surv(time, status), x=x,
   fit.fun=fit.CoxBoost, args.fit=list(penalty=100),
   complexity=complexity.mincv.CoxBoost, args.complexity=list(penalty=100),
   indices=resample.indices(n=length(time), method="sub632", sample.n=10),
   args.aggregation=list(times=seq(0, quantile(time, probs=0.9), length.out=100)),
   trace=TRUE)
plot(peperr.object3)

## End(Not run)
```

---

pmpec *Calculate prediction error curves*

---

## Description

Calculation of prediction error curve from a survival response and predicted probabilities of survival.

## Usage

```
pmpec(object, response=NULL, x=NULL, times, model.args=NULL,
    type=c("PErr","NoInf"), external.time=NULL, external.status=NULL,
    data=NULL)
```

## Arguments

| | |
|---|---|
| object | fitted model of a class for which the interface function predictProb.class is available. |
| response | Either a survival object (with Surv(time, status), where time is an n-vector of censored survival times and status an n-vector containing event status, coded with 0 and 1) or a matrix with columns time containing survival times and status containing integers, where 0 indicates censoring, 1 the interesting event and larger numbers other competing risks. |
| x | n*p matrix of covariates. |
| times | vector of time points at which the prediction error is to be estimated. |
| model.args | named list of additional arguments, e.g. complexity value, which are to be passed to predictProb function. |
| type | type of output: Estimated prediction error (default) or no information error (prediction error obtained by permuting the data). |
| external.time | optional vector of time points, used for censoring distribution. |
| external.status | optional vector of status values, used for censoring distribution. |

`data` Data frame containing `n`-vector of observed times ('time'), `n`-vector of event status ('status') and `n*p` matrix of covariates (remaining entries). Alternatively to `response` and `x`, for compatibility to **pec**.

## Details

Prediction error of survival data is measured by the Brier score, which considers the squared difference of the true event status at a given time point and the predicted event status by a risk prediction model at that time. A prediction error curve is the weighted mean Brier score as a function of time at time points in `times` (see References).

`pmpec` requires a `predictProb` method for the class of the fitted model, i.e. for a model of class `class` `predictProb.class`.

`pmpec` is implemented to behave similar to function `pec` of package **pec**, which provides several `predictProb` methods.

In bootstrap framework, `data` contains only a part of the full data set. For censoring distribution, the full data should be used to avoid extreme variance in case of small data sets. For that, the observed times and status values can be passed as argument `external.time` and `external.status`.

## Value

Vector of prediction error estimates at each time point given in `time`.

## Author(s)

Harald Binder

## References

Gerds, A. and Schumacher, M. (2006) Consistent estimation of the expected Brier score in general survival models with right-censored event times. Biometrical Journal, 48, 1029–1040.

Schoop, R. (2008) Predictive accuracy of failure time models with longitudinal covariates. PhD thesis, University of Freiburg. http://www.freidok.uni-freiburg.de/volltexte/4995/.

## See Also

`predictProb`, **pec**

---

# `predictProb` *Generic function for extracting predicted survival probabilities*

---

## Description

Generic function for extraction of predicted survival probabilities from a fitted survival model conforming to the interface required by `pmpec`.

### Usage

```
predictProb(object, response, x, ...)
```

### Arguments

| | |
|---|---|
| object | a fitted survival model. |
| response | Either a survival object (with Surv(time, status), where time is an n-vector of censored survival times and status an n-vector containing event status, coded with 0 and 1) or a matrix with columns time containing survival times and status containing integers, where 0 indicates censoring, 1 the interesting event and larger numbers other competing risks. In case of binary response, vector with entries 0 and 1. |
| x | n*p matrix of covariates. |
| ... | additional arguments, for example model complexity or, in case of survival response, argument times, a vector containing evaluation times. |

### Details

pmpec requires a predictProb.class function for each model fit of class class. It extracts the predicted probability of survival from this model.

See existing predictProb functions, at the time predictProb.CoxBoost, predictProb.coxph and predictProb.survfit.

If desired predictProb function for class class is not available in **peperr**, but implemented in package **pec** as predictSurvProb.class, it can easily be transformed as predictProb method.

### Value

Matrix with predicted probabilities for each evaluation time point in times (columns) and each new observation (rows).

---

## predictProb.CoxBoost *Extract predicted survival probabilities from a CoxBoost fit*

---

### Description

Extracts predicted survival probabilities from survival model fitted by CoxBoost, providing an interface as required by pmpec.

### Usage

```
## S3 method for class 'CoxBoost'
predictProb(object, response, x, times, complexity, ...)
```

### Arguments

| | |
|---|---|
| object | a fitted model of class CoxBoost. |
| response | survival object (with Surv(time, status), where time is an n-vector of censored survival times and status an n-vector containing survival status, coded with 0 and 1. |
| x | n*p matrix of covariates. |
| times | vector of evaluation time points. |
| complexity | complexity value. |
| ... | additional arguments, currently not used. |

### Value

Matrix with probabilities for each evaluation time point in times (columns) and each new observation (rows).

---

## predictProb.coxph *Extract predicted survival probabilities from a coxph object*

---

### Description

Extracts predicted survival probabilities for survival models fitted by Cox proportional hazards model, providing an interface as required by pmpec.

### Usage

```
## S3 method for class 'coxph'
predictProb(object, response, x, times, ...)
```

### Arguments

| | |
|---|---|
| object | a fitted model of class coxph. |
| response | survival object (with Surv(time, status), where time is an n-vector of censored survival times and status an n-vector containing survival status, coded with 0 and 1. |
| x | n*p matrix of covariates. |
| times | vector of evaluation time points. |
| ... | additional arguments, currently not used. |

### Value

Matrix with probabilities for each evaluation time point in times(columns) and each new observation (rows).

## predictProb.survfit *Extract predicted survival probabilities from a survfit object*

### Description

Extracts predicted survival probabilities for survival models fitted by `survfit`, providing an interface as required by `pmpec`.

### Usage

```
## S3 method for class 'survfit'
predictProb(object, response, x, times, train.data, ...)
```

### Arguments

| | |
|---|---|
| `object` | a fitted model of class `survfit`. |
| `response` | survival object (with `Surv(time, status)`, where time is an `n`-vector of censored survival times and status an `n`-vector containing survival status, coded with 0 and 1. |
| `x` | `n*p` matrix of covariates. |
| `times` | vector of evaluation time points. |
| `train.data` | not used. |
| `...` | additional arguments, currently not used. |

### Value

Matrix with probabilities for each evaluation time point in `times`(columns) and each new observation (rows).

## resample.indices *Generation of indices for resampling Procedure*

### Description

Generates training and test set indices for use in resampling estimation of prediction error, e.g. cross-validation or bootstrap (with and without replacement).

### Usage

```
resample.indices(n, sample.n = 100, method = c("no", "cv" ,"boot", "sub632"))
```

## Arguments

| | |
|---|---|
| n | number of observations of the full data set. |
| sample.n | the number of bootstrap samples in case of method="boot" and the number of cross-validation subsets in case of method="cv", e.g. 10 for 10-fold cross-validation. Not considered if method="no", where number of samples is one (the full data set) by definition. |
| method | by default, the training set indices are the same as the test set indices, i.e. the model is assessed in the same data as fitted ("no"). "cv": Cross-validation, "boot": Bootstrap (with replacement), "sub632": Boostrap without replacement, also called subsampling. In the latter case, the number of observations in each sample equals round(0.632 * n), see Details. |

## Details

As each bootstrap sample should be taken as if new data, complexity selection should be carried out in each bootstrap sample. Binder and Schumacher show that when bootstrap samples are drawn with replacement, often too complex models are obtained in high-dimensional data settings. They recommend to draw bootstrap samples without replacement, each of size round(0.632 * n), which equals the expected number of unique observations in one bootstrap sample drawn with replacement, to avoid biased complexity selection and improve predictive power of the resulting models.

## Value

A list containing two lists of length sample.n:

| | |
|---|---|
| sample.index | contains in each element the indices of observations of one training set. |
| not.in.sample | contains in each element the indices of observations of one test set, corresponding to the training set in listelement sample.index. |

## References

Binder, H. and Schumacher, M. (2008) Adapting prediction error estimates for biased complexity selection in high-dimensional bootstrap samples. Statistical Applications in Genetics and Molecular Biology, 7:1.

## See Also

peperr

## Examples

```
# generate dataset: 100 patients, 20 covariates
data <- matrix(rnorm(2000), nrow=100)

# generate indices for training and test data for 10-fold cross-validation
indices <- resample.indices(n=100, sample.n = 10, method = "cv")

# create training and test data via indices
```

```
trainingsample1 <- data[indices$sample.index[[1]],]
testsample1 <- data[indices$not.in.sample[[1]],]
```

# Index