Package ‘perry’

February 20, 2015

Type Package

Title Resampling-based prediction error estimation for regression models

Version 0.2.0

Date 2013-03-12

Depends R (>= 2.14.1), ggplot2 (>= 0.9.2), parallel, robustbase

Imports stats

Description Tools that allow developers to write functions for prediction error estimation with minimal programming effort and assist users with model selection in regression problems.

License GPL (>= 2)

LazyLoad yes

Author Andreas Alfons [aut, cre]

Maintainer Andreas Alfons <alfons@ese.eur.nl>

NeedsCompilation no

Repository CRAN

Date/Publication 2013-03-13 11:10:58

R topics documented:

perry-package *Resampling-based prediction error estimation for regression models*

Description

Tools that allow developers to write functions for prediction error estimation with minimal programming effort and assist users with model selection in regression problems.

Details

Package:	perry
Type:	Package
Version:	0.2.0
Date:	2013-03-12
Depends:	R (>= 2.14.1), ggplot2 (>= 0.9.2), parallel, robustbase
Imports:	stats
License:	GPL (>= 2)
LazyLoad:	yes

Index:

```
accessors               Access or set information on resampling-based
                        prediction error results
aggregate.perry         Aggregate resampling-based prediction error
                        results
bootControl             Control object for bootstrap samples
bootSamples             Bootstrap samples
cost                    Prediction loss
cvFolds                 Cross-validation folds
foldControl             Control object for cross-validation folds
fortify.perry           Convert resampling-based prediction error
                        results into a data frame for plotting
```

`perry`	`Resampling-based prediction error for fitted models`
`perry-deprecated`	`Deprecated functions in package 'perry'`
`perry-package`	`Resampling-based prediction error estimation for regression models`
`perryFit`	`Resampling-based prediction error for model evaluation`
`perryPlot`	`Plot resampling-based prediction error results`
`perryReshape`	`Reshape resampling-based prediction error results`
`perrySelect`	`Model selection via resampling-based prediction error measures`
`perrySplits`	`Data splits for resampling-based prediction error measures`
`perryTuning`	`Resampling-based prediction error for tuning parameter selection`
`randomSplits`	`Random data splits`
`reperry`	`Recompute resampling-based prediction error measures`
`splitControl`	`Control object for random data splits`
`subset.perry`	`Subsetting resampling-based prediction error results`
`summary.perry`	`Summarize resampling-based prediction error results`

Author(s)

Andreas Alfons [aut, cre]

Maintainer: Andreas Alfons <alfons@ese.eur.nl>

`accessors` *Access or set information on resampling-based prediction error results*

Description

Retrieve or set the names of resampling-based prediction error results, retrieve or set the identifiers of the models, or retrieve the number of prediction error results or included models.

Usage

```
peNames(x)

peNames(x) <- value

fits(x)

fits(x) <- value
```

```
npe(x)

nfits(x)
```

Arguments

x	an object inheriting from class "perry" or "perrySelect" that contains prediction error results.
value	a vector of replacement values.

Value

peNames returns the names of the prediction error results. The replacement function thereby returns them invisibly.

fits returns the identifiers of the models for objects inheriting from class "perrySelect" and NULL for objects inheriting from class "perry". The replacement function thereby returns those values invisibly.

npe returns the number of prediction error results.

nfits returns the number of models included in objects inheriting from class "perrySelect" and NULL for objects inheriting from class "perry".

Author(s)

Andreas Alfons

See Also

perryFit, perrySelect, perryTuning

Examples

```
data("coleman")
set.seed(1234)  # set seed for reproducibility

## set up folds for cross-validation
folds <- cvFolds(nrow(coleman), K = 5, R = 10)

## compare raw and reweighted LTS estimators for
## 50% and 75% subsets

# 50% subsets
fit50 <- ltsReg(Y ~ ., data = coleman, alpha = 0.5)
cv50 <- perry(fit50, splits = folds, fit = "both",
    cost = rtmspe, trim = 0.1)

# 75% subsets
fit75 <- ltsReg(Y ~ ., data = coleman, alpha = 0.75)
cv75 <- perry(fit75, splits = folds, fit = "both",
    cost = rtmspe, trim = 0.1)
```

```
# combine results into one object
cv <- perrySelect("0.5" = cv50, "0.75" = cv75)
cv

# "perry" object
npe(cv50)
nfits(cv50)
peNames(cv50)
peNames(cv50) <- c("improved", "initial")
fits(cv50)
cv50

# "perrySelect" object
npe(cv)
nfits(cv)
peNames(cv)
peNames(cv) <- c("improved", "initial")
fits(cv)
fits(cv) <- 1:2
cv
```

aggregate.perry *Aggregate resampling-based prediction error results*

Description

Compute summary statistics of resampling-based prediction error results.

Usage

```
 ## S3 method for class 'perry'
aggregate(x, FUN = mean, select = NULL,
   ...)

 ## S3 method for class 'perrySelect'
aggregate(x, FUN = mean,
   select = NULL, ...)

 ## S3 method for class 'perryTuning'
aggregate(x, ...)
```

Arguments

x	an object inheriting from class "perry" or "perrySelect" that contains prediction error results (note that the latter includes objects of class "perryTuning").
FUN	a function to compute the summary statistics.
select	a character, integer or logical vector indicating the columns of prediction error results for which to compute the summary statistics.

`...` for the "perryTuning" method, additional arguments to be passed to the "perrySelect" method. Otherwise additional arguments to be passed to FUN.

Value

The "perry" method returns a vector or matrix of aggregated prediction error results, depending on whether FUN returns a single value or a vector.

For the other methods, a data frame containing the aggregated prediction error results for each model is returned. In the case of the "perryTuning" method, the data frame contains the combinations of tuning parameters rather than a column describing the models.

Note

Duplicate indices in subset or select are removed such that all models and prediction error results are unique.

Author(s)

Andreas Alfons

See Also

perryFit, perrySelect, perryTuning, aggregate

Examples

```
data("coleman")
set.seed(1234)  # set seed for reproducibility

## set up folds for cross-validation
folds <- cvFolds(nrow(coleman), K = 5, R = 10)

## compare raw and reweighted LTS estimators for
## 50% and 75% subsets

# 50% subsets
fit50 <- ltsReg(Y ~ ., data = coleman, alpha = 0.5)
cv50 <- perry(fit50, splits = folds, fit = "both",
    cost = rtmspe, trim = 0.1)

# 75% subsets
fit75 <- ltsReg(Y ~ ., data = coleman, alpha = 0.75)
cv75 <- perry(fit75, splits = folds, fit = "both",
    cost = rtmspe, trim = 0.1)

# combine results into one object
cv <- perrySelect("0.5" = cv50, "0.75" = cv75)
cv

# summary of the results with the 50% subsets
aggregate(cv50, summary)
```

```
# summary of the combined results
aggregate(cv, summary)
```

`bootControl` *Control object for bootstrap samples*

Description

Generate an object that controls how to draw bootstrap samples and which bootstrap estimator of prediction error to compute.

Usage

```
  bootControl(R = 1, type = c("0.632", "out-of-bag"),
    grouping = NULL)
```

Arguments

`R` an integer giving the number of bootstrap samples.

`type` a character string specifying a bootstrap estimator. Possible values are `"0.632"` (the default), or `"out-of-bag"`.

`grouping` a factor specifying groups of observations.

Value

An object of class `"bootSamples"` with the following components:

`R` an integer giving the number of bootstrap samples.

`type` a character string specifying the type of bootstrap estimator.

`grouping` if supplied, a factor specifying groups of observations. The groups will then be resampled rather than individual observations such that all observations within a group belong either to the bootstrap sample or the test data.

Author(s)

Andreas Alfons

References

Efron, B. (1983) Estimating the error rate of a prediction rule: improvement on cross-validation. *Journal of the American Statistical Association*, **78**(382), 316–331.

See Also

`perrySplits`, `bootSamples`, `foldControl`, `splitControl`

Examples

```
set.seed(1234)  # set seed for reproducibility
perrySplits(20, bootControl())
perrySplits(20, bootControl(R = 10))
```

---

bootSamples *Bootstrap samples*

Description

Draw bootstrap samples of observations or groups of observations and specify which bootstrap estimator of prediction error to compute.

Usage

```
bootSamples(n, R = 1, type = c("0.632", "out-of-bag"),
  grouping = NULL)
```

Arguments

n	an integer giving the number of observations for which to draw bootstrap samples. This is ignored if grouping is supplied in order to respect the group structure of the data in the bootstrap samples.
R	an integer giving the number of bootstrap samples.
type	a character string specifying a bootstrap estimator. Possible values are "0.632" (the default), or "out-of-bag".
grouping	a factor specifying groups of observations. If supplied, the groups are resampled rather than individual observations such that all observations within a group belong either to the bootstrap sample or the test data.

Value

An object of class "bootSamples" with the following components:

n	an integer giving the number of observations or groups.
R	an integer giving the number of bootstrap samples.
subsets	an integer matrix in which each column contains the indices of the observations or groups in the corresponding bootstrap sample.
grouping	a list giving the indices of the observations belonging to each group. This is only returned if a grouping factor has been supplied.

Note

This is a simple wrapper function for perrySplits with a control object generated by bootControl.

Author(s)

Andreas Alfons

References

Efron, B. (1983) Estimating the error rate of a prediction rule: improvement on cross-validation. *Journal of the American Statistical Association*, **78**(382), 316–331.

See Also

perrySplits, bootControl, cvFolds, randomSplits

Examples

```
set.seed(1234)  # set seed for reproducibility
bootSamples(20)
bootSamples(20, R = 10)
```

---

cost *Prediction loss*

Description

Compute the prediction loss of a model.

Usage

```
mspe(y, yHat, includeSE = FALSE)

rmspe(y, yHat, includeSE = FALSE)

mape(y, yHat, includeSE = FALSE)

tmspe(y, yHat, trim = 0.25, includeSE = FALSE)

rtmspe(y, yHat, trim = 0.25, includeSE = FALSE)
```

Arguments

y	a numeric vector or matrix giving the observed values.
yHat	a numeric vector or matrix of the same dimensions as y giving the fitted values.
trim	a numeric value giving the trimming proportion (the default is 0.25).
includeSE	a logical indicating whether standard errors should be computed as well.

Details

mspe and rmspe compute the mean squared prediction error and the root mean squared prediction error, respectively. In addition, mape returns the mean absolute prediction error, which is somewhat more robust.

Robust prediction loss based on trimming is implemented in tmspe and rtmspe. To be more precise, tmspe computes the trimmed mean squared prediction error and rtmspe computes the root trimmed mean squared prediction error. A proportion of the largest squared differences of the observed and fitted values are thereby trimmed.

Standard errors can be requested via the includeSE argument. Note that standard errors for tmspe are based on a winsorized standard deviation. Furthermore, standard errors for rmspe and rtmspe are computed from the respective standard errors of mspe and tmspe via the delta method.

Value

If standard errors are not requested, a numeric value giving the prediction loss is returned.

Otherwise a list is returned, with the first component containing the prediction loss and the second component the corresponding standard error.

Author(s)

Andreas Alfons

References

Tukey, J.W. and McLaughlin, D.H. (1963) Less vulnerable confidence and significance procedures for location based on a single sample: Trimming/winsorization. *Sankhya: The Indian Journal of Statistics, Series A*, **25**(3), 331–352

Oehlert, G.W. (1992) A note on the delta method. *The American Statistician*, **46**(1), 27–29.

See Also

perryFit, perryTuning

Examples

```
# fit an MM-regression model
data("coleman")
fit <- lmrob(Y~., data=coleman)

# compute the prediction loss from the fitted values
# (hence the prediction loss is underestimated in this simple
# example since all observations are used to fit the model)
mspe(coleman$Y, predict(fit))
rmspe(coleman$Y, predict(fit))
mape(coleman$Y, predict(fit))
tmspe(coleman$Y, predict(fit), trim = 0.1)
rtmspe(coleman$Y, predict(fit), trim = 0.1)

# include standard error
```

```
mspe(coleman$Y, predict(fit), includeSE = TRUE)
rmspe(coleman$Y, predict(fit), includeSE = TRUE)
mape(coleman$Y, predict(fit), includeSE = TRUE)
tmspe(coleman$Y, predict(fit), trim = 0.1, includeSE = TRUE)
rtmspe(coleman$Y, predict(fit), trim = 0.1, includeSE = TRUE)
```

`cvFolds` *Cross-validation folds*

Description

Split observations or groups of observations into K folds to be used for (repeated) K-fold cross-validation. K should thereby be chosen such that all folds are of approximately equal size.

Usage

```
cvFolds(n, K = 5, R = 1,
  type = c("random", "consecutive", "interleaved"),
  grouping = NULL)
```

Arguments

`n` an integer giving the number of observations to be split into folds. This is ignored if `grouping` is supplied in order to split groups of observations into folds.

`K` an integer giving the number of folds into which the observations should be split (the default is five). Setting `K` equal to the number of observations or groups yields leave-one-out cross-validation.

`R` an integer giving the number of replications for repeated K-fold cross-validation. This is ignored for for leave-one-out cross-validation and other non-random splits of the data.

`type` a character string specifying the type of folds to be generated. Possible values are `"random"` (the default), `"consecutive"` or `"interleaved"`.

`grouping` a factor specifying groups of observations. If supplied, the data are split according to the groups rather than individual observations such that all observations within a group belong to the same fold.

Value

An object of class `"cvFolds"` with the following components:

`n` an integer giving the number of observations or groups.

`K` an integer giving the number of folds.

`R` an integer giving the number of replications.

`subsets` an integer matrix in which each column contains a permutation of the indices of the observations or groups.

`which` an integer vector giving the fold for each permuted observation or group.

`grouping` a list giving the indices of the observations belonging to each group. This is only returned if a grouping factor has been supplied.

Note

This is a simple wrapper function for `perrySplits` with a control object generated by `foldControl`.

Author(s)

Andreas Alfons

See Also

`perrySplits`, `foldControl`, `randomSplits`, `bootSamples`

Examples

```
set.seed(1234)  # set seed for reproducibility
cvFolds(20, K = 5, type = "random")
cvFolds(20, K = 5, type = "consecutive")
cvFolds(20, K = 5, type = "interleaved")
cvFolds(20, K = 5, R = 10)
```

`foldControl` *Control object for cross-validation folds*

Description

Generate an object that controls how to split n observations or groups of observations into K folds to be used for (repeated) K-fold cross-validation. K should thereby be chosen such that all folds are of approximately equal size.

Usage

```
  foldControl(K = 5, R = 1,
    type = c("random", "consecutive", "interleaved"),
    grouping = NULL)
```

Arguments

`K` an integer giving the number of folds into which the observations should be split (the default is five).

`R` an integer giving the number of replications for repeated K-fold cross-validation.

`type` a character string specifying the type of folds to be generated. Possible values are `"random"` (the default), `"consecutive"` or `"interleaved"`.

`grouping` a factor specifying groups of observations.

Value

An object of class "foldControl" with the following components:

K an integer giving the number of folds. A value of K equal to the number of observations or groups yields leave-one-out cross-validation.

R an integer giving the number of replications. This will be ignored for for leave-one-out cross-validation and other non-random splits of the data.

type a character string specifying the type of folds.

grouping if supplied, a factor specifying groups of observations. The data will then be split according to the groups rather than individual observations such that all observations within a group belong to the same fold.

Author(s)

Andreas Alfons

See Also

perrySplits, cvFolds, splitControl, bootControl

Examples

```
set.seed(1234)  # set seed for reproducibility
perrySplits(20, foldControl(K = 5))
perrySplits(20, foldControl(K = 5, R = 10))
```

fortify.perry *Convert resampling-based prediction error results into a data frame for plotting*

Description

Extract all necessary information for plotting from resampling-based prediction error results and store it in a data frame.

Usage

```
 ## S3 method for class 'perry'
fortify(model, data, select = NULL,
   reps = model$splits$R > 1, seFactor = NA, ...)

 ## S3 method for class 'perrySelect'
fortify(model, data,
   subset = NULL, select = NULL,
   reps = model$splits$R > 1, seFactor = model$seFactor,
   ...)

 ## S3 method for class 'perryTuning'
fortify(model, data, ...)
```

Arguments

`model` an object inheriting from class `"perry"` or `"perrySelect"` that contains prediction error results.

`data` currently ignored.

`subset` a character, integer or logical vector indicating the subset of models to be converted.

`select` a character, integer or logical vector indicating the columns of prediction error results to be converted.

`reps` a logical indicating whether to convert the results from all replications (`TRUE`) or the aggregated results (`FALSE`). The former is suitable for box plots or smooth density plots, while the latter is suitable for dot plots or line plots (see `perryPlot`).

`seFactor` a numeric value giving the multiplication factor of the standard error for displaying error bars in dot plots or line plots. Error bars in those plots can be suppressed by setting this to `NA`.

`...` for the `"perryTuning"` method, additional arguments to be passed down to the `"perrySelect"` method. For the other methods, additional arguments are currently ignored.

Value

A data frame containing the columns listed below, as well as additional information stored in the attribute `"facets"` (default faceting formula for the plots).

`Fit` a vector or factor containing the identifiers of the models.

`Name` a factor containing the names of the predictor error results (not returned in case of only one column of prediction error results with the default name).

`PE` the estimated prediction errors.

`Lower` the lower end points of the error bars (only returned if `reps` is `FALSE`).

`Upper` the upper end points of the error bars (only returned if `reps` is `FALSE`).

Note

Duplicate indices in `subset` or `select` are removed such that all models and prediction error results are unique.

Author(s)

Andreas Alfons

See Also

`fortify`, `perryPlot`, `perryFit`, `perrySelect`, `perryTuning`

Examples

```
data("coleman")
set.seed(1234)  # set seed for reproducibility

## set up folds for cross-validation
folds <- cvFolds(nrow(coleman), K = 5, R = 10)

## compare LS, MM and LTS regression

# perform cross-validation for an LS regression model
fitLm <- lm(Y ~ ., data = coleman)
cvLm <- perry(fitLm, splits = folds,
    cost = rtmspe, trim = 0.1)

# perform cross-validation for an MM regression model
fitLmrob <- lmrob(Y ~ ., data = coleman, k.max = 500)
cvLmrob <- perry(fitLmrob, splits = folds,
    cost = rtmspe, trim = 0.1)

# perform cross-validation for an LTS regression model
fitLts <- ltsReg(Y ~ ., data = coleman)
cvLts <- perry(fitLts, splits = folds,
    cost = rtmspe, trim = 0.1)

# combine results into one object
cv <- perrySelect(LS = cvLm, MM = cvLmrob, LTS = cvLts)
cv

## convert MM regression results to data frame for plotting
# all replications for box plot
cvLmrobBox <- fortify(cvLmrob, reps = TRUE)
perryPlot(cvLmrobBox)
# aggregated results for dot plot
cvLmrobDot <- fortify(cvLmrob, reps = FALSE, seFactor = 1)
perryPlot(cvLmrobDot)

## convert combined results to data frame for plotting
# all replications for box plot
cvBox <- fortify(cv, reps = TRUE)
perryPlot(cvBox)
# aggregated results for dot plot
cvDot <- fortify(cv, reps = FALSE, seFactor = 1)
perryPlot(cvDot)
```

---

perry *Resampling-based prediction error for fitted models*

Description

Estimate the prediction error of a fitted model via (repeated) K-fold cross-validation, (repeated) random splitting (also known as random subsampling or Monte Carlo cross-validation), or the bootstrap. Methods are available for least squares fits computed with `lm` as well as for the following robust alternatives: MM-type models computed with `lmrob` and least trimmed squares fits computed with `ltsReg`.

Usage

```
 perry(object, ...)

 ## S3 method for class 'lm'
perry(object, splits = foldControl(),
   cost = rmspe, ncores = 1, cl = NULL, seed = NULL, ...)

 ## S3 method for class 'lmrob'
perry(object, splits = foldControl(),
   cost = rtmspe, ncores = 1, cl = NULL, seed = NULL, ...)

 ## S3 method for class 'lts'
perry(object, splits = foldControl(),
   fit = c("reweighted", "raw", "both"), cost = rtmspe,
   ncores = 1, cl = NULL, seed = NULL, ...)
```

Arguments

`object` the fitted model for which to estimate the prediction error.

`splits` an object of class `"cvFolds"` (as returned by `cvFolds`) or a control object of class `"foldControl"` (see `foldControl`) defining the folds of the data for (repeated) K-fold cross-validation, an object of class `"randomSplits"` (as returned by `randomSplits`) or a control object of class `"splitControl"` (see `splitControl`) defining random data splits, or an object of class `"bootSamples"` (as returned by `bootSamples`) or a control object of class `"bootControl"` (see `bootControl`) defining bootstrap samples.

`fit` a character string specifying for which fit to estimate the prediction error. Possible values are `"reweighted"` (the default) for the prediction error of the reweighted fit, `"raw"` for the prediction error of the raw fit, or `"both"` for the prediction error of both fits.

`cost` a cost function measuring prediction loss. It should expect the observed values of the response to be passed as the first argument and the predicted values as the second argument, and must return either a non-negative scalar value, or a list with the first component containing the prediction error and the second component containing the standard error. The default is to use the root mean squared prediction error for the `"lm"` method and the root trimmed mean squared prediction error for the `"lmrob"` and `"lts"` methods (see `cost`).

`ncores` a positive integer giving the number of processor cores to be used for parallel computing (the default is 1 for no parallelization). If this is set to `NA`, all available processor cores are used.

`cl` a **parallel** cluster for parallel computing as generated by `makeCluster`. If supplied, this is preferred over `ncores`.

`seed` optional initial seed for the random number generator (see `.Random.seed`). Note that also in case of parallel computing, resampling is performed on the manager process rather than the worker processes. On the parallel worker processes, random number streams are used and the seed is set via `clusterSetRNGStream`.

`...` for the generic function, additional arguments to be passed down to methods. For the methods, additional arguments to be passed to the prediction loss function `cost`.

Value

An object of class `"perry"` with the following components:

`pe` a numeric vector containing the estimated prediction errors. For the `"lm"` and `"lmrob"` methods, this is a single numeric value. For the `"lts"` method, this contains one value for each of the requested fits. In case of more than one replication, those are average values over all replications.

`se` a numeric vector containing the estimated standard errors of the prediction loss. For the `"lm"` and `"lmrob"` methods, this is a single numeric value. For the `"lts"` method, this contains one value for each of the requested fits.

`reps` a numeric matrix containing the estimated prediction errors from all replications. For the `"lm"` and `"lmrob"` methods, this is a matrix with one column. For the `"lts"` method, this contains one column for each of the requested fits. However, this is only returned in case of more than one replication.

`splits` an object giving the data splits used to estimate the prediction error.

`y` the response.

`yHat` a list containing the predicted values from all replications.

`call` the matched function call.

Note

The `perry` methods extract the data from the fitted model and call `perryFit` to perform resampling-based prediction error estimation.

Author(s)

Andreas Alfons

See Also

`perryFit`

Examples

```
## load data and fit an LS regression model
data("mtcars")
fit <- lm(mpg ~ wt + cyl, data=mtcars)

## perform cross-validation
```

```
# K-fold CV
perry(fit, foldControl(K = 5, R = 10), seed = 1234)
# leave-one-out CV
perry(fit, foldControl(K = nrow(mtcars)))

## perform random splitting
perry(fit, splitControl(m = 6, R = 10), seed = 1234)

## perform bootstrap prediction error estimation
# 0.632 estimator
perry(fit, bootControl(R = 10, type = "0.632"), seed = 1234)
# out-of-bag estimator
perry(fit, bootControl(R = 10, type = "out-of-bag"), seed = 1234)
```

---

perry-deprecated *Deprecated functions in package* **perry**

Description

These functions are provided for compatibility with older versions only, and may be defunct as soon as the next release.

Usage

```
repCV(object, K = 5, R = 1,
  foldType = c("random", "consecutive", "interleaved"),
  grouping = NULL, folds = NULL, ...)

repRS(object, m, R = 1, grouping = NULL, splits = NULL,
  ...)

bootPE(object, R = 1,
  bootType = c("0.632", "out-of-bag"), grouping = NULL,
  samples = NULL, ...)
```

Arguments

`object` the fitted model for which to estimate the prediction error.

`K` an integer giving the number of folds into which the observations should be split (the default is five). Setting `K` equal to the number of observations or groups yields leave-one-out cross-validation.

`m` an integer giving the number of observations or groups of observations to be used as test data.

`R` an integer giving the number of replications. In `repCV`, this is ignored for for leave-one-out cross-validation and other non-random splits of the data.

`foldType` a character string specifying the type of folds to be generated. Possible values are `"random"` (the default), `"consecutive"` or `"interleaved"`.

`bootType` a character string specifying a bootstrap estimator. Possible values are `"0.632"` (the default), or `"out-of-bag"`.

`grouping` a factor specifying groups of observations. If supplied, the data are split according to the groups rather than individual observations such that all observations within a group belong either to the training or test data.

`folds` an object of class `"cvFolds"` (as returned by `cvFolds`) or a control object of class `"foldControl"` (see `foldControl`) defining the folds of the data for (repeated) K-fold cross-validation. If supplied, this is preferred over the arguments for generating cross-validation folds.

`splits` an object of class `"randomSplits"` (as returned by `randomSplits`) or a control object of class `"splitControl"` (see `splitControl`) defining random data splits. If supplied, this is preferred over the arguments for generating random data splits.

`samples` an object of class `"bootSamples"` (as returned by `bootSamples`) or a control object of class `"bootControl"` (see `bootControl`) defining bootstrap samples. If supplied, this is preferred over the arguments for generating bootstrap samples.

`...` additional arguments to be passed down to `perry`.

Details

`repCV`, `repRS` and `bootPE` are wrapper functions for `perry` that perform (repeated) K-fold cross-validation, (repeated) random splitting (also known as random subsampling or Monte Carlo cross-validation) and the bootstrap, respectively.

Author(s)

Andreas Alfons

See Also

`Deprecated`

`perryFit` *Resampling-based prediction error for model evaluation*

Description

Estimate the prediction error of a model via (repeated) K-fold cross-validation, (repeated) random splitting (also known as random subsampling or Monte Carlo cross-validation), or the bootstrap. It is thereby possible to supply an object returned by a model fitting function, a model fitting function itself, or an unevaluated function call to a model fitting function.

Usage

```
perryFit(object, ...)

 ## Default S3 method:
perryFit(object, data = NULL,
   x = NULL, y, splits = foldControl(),
   predictFun = predict, predictArgs = list(),
   cost = rmspe, costArgs = list(), names = NULL,
   envir = parent.frame(), ncores = 1, cl = NULL,
   seed = NULL, ...)

 ## S3 method for class 'function'
perryFit(object, formula,
   data = NULL, x = NULL, y, args = list(),
   splits = foldControl(), predictFun = predict,
   predictArgs = list(), cost = rmspe, costArgs = list(),
   names = NULL, envir = parent.frame(), ncores = 1,
   cl = NULL, seed = NULL, ...)

 ## S3 method for class 'call'
perryFit(object, data = NULL, x = NULL,
   y, splits = foldControl(), predictFun = predict,
   predictArgs = list(), cost = rmspe, costArgs = list(),
   names = NULL, envir = parent.frame(), ncores = 1,
   cl = NULL, seed = NULL, ...)
```

Arguments

`object` the fitted model for which to estimate the prediction error, a function for fitting a model, or an unevaluated function call for fitting a model (see `call` for the latter). In the case of a fitted model, the object is required to contain a component `call` that stores the function call used to fit the model, which is typically the case for objects returned by model fitting functions.

`formula` a `formula` describing the model.

`data` a data frame containing the variables required for fitting the models. This is typically used if the model in the function call is described by a `formula`.

`x` a numeric matrix containing the predictor variables. This is typically used if the function call for fitting the models requires the predictor matrix and the response to be supplied as separate arguments.

`y` a numeric vector or matrix containing the response.

`args` a list of additional arguments to be passed to the model fitting function.

`splits` an object of class `"cvFolds"` (as returned by `cvFolds`) or a control object of class `"foldControl"` (see `foldControl`) defining the folds of the data for (repeated) K-fold cross-validation, an object of class `"randomSplits"` (as returned by `randomSplits`) or a control object of class `"splitControl"` (see `splitControl`) defining random data splits, or an object of class `"bootSamples"` (as returned by bootSamples) or a control object of class "bootControl" (see bootControl) defining bootstrap samples.

predictFun a function to compute predictions for the test data. It should expect the fitted model to be passed as the first argument and the test data as the second argument, and must return either a vector or a matrix containing the predicted values. The default is to use the predict method of the fitted model.

predictArgs a list of additional arguments to be passed to predictFun.

cost a cost function measuring prediction loss. It should expect the observed values of the response to be passed as the first argument and the predicted values as the second argument, and must return either a non-negative scalar value, or a list with the first component containing the prediction error and the second component containing the standard error. The default is to use the root mean squared prediction error (see cost).

costArgs a list of additional arguments to be passed to the prediction loss function cost.

names an optional character vector giving names for the arguments containing the data to be used in the function call (see “Details”).

envir the environment in which to evaluate the function call for fitting the models (see eval).

ncores a positive integer giving the number of processor cores to be used for parallel computing (the default is 1 for no parallelization). If this is set to NA, all available processor cores are used.

cl a **parallel** cluster for parallel computing as generated by makeCluster. If supplied, this is preferred over ncores.

seed optional initial seed for the random number generator (see .Random.seed). Note that also in case of parallel computing, resampling is performed on the manager process rather than the worker processes. On the parallel worker processes, random number streams are used and the seed is set via clusterSetRNGStream for reproducibility in case the model fitting function involves randomness.

... additional arguments to be passed down.

Details

(Repeated) K-fold cross-validation is performed in the following way. The data are first split into K previously obtained blocks of approximately equal size (given by folds). Each of the K data blocks is left out once to fit the model, and predictions are computed for the observations in the left-out block with predictFun. Thus a prediction is obtained for each observation. The response variable and the obtained predictions for all observations are then passed to the prediction loss function cost to estimate the prediction error. For repeated K-fold cross-validation (as indicated by splits), this process is replicated and the estimated prediction errors from all replications are returned.

(Repeated) random splitting is performed similarly. In each replication, the data are split into a training set and a test set at random. Then the training data is used to fit the model, and predictions are computed for the test data. Hence only the response values from the test data and the corresponding predictions are passed to the prediction loss function cost.

For the bootstrap estimator, each bootstrap sample is used as training data to fit the model. The out-of-bag estimator uses the observations that do not enter the bootstrap sample as test data and computes the prediction loss function cost for those out-of-bag observations. The 0.632 estimator is computed as a linear combination of the out-of-bag estimator and the prediction loss of the fitted values of the model computed from the full sample.

In any case, if the response is a vector but predictFun returns a matrix, the prediction error is computed for each column. A typical use case for this behavior would be if predictFun returns predictions from an initial model fit and stepwise improvements thereof.

If formula or data are supplied, all variables required for fitting the models are added as one argument to the function call, which is the typical behavior of model fitting functions with a formula interface. In this case, the accepted values for names depend on the method. For the function method, a character vector of length two should supplied, with the first element specifying the argument name for the formula and the second element specifying the argument name for the data (the default is to use c("formula", "data")). Note that names for both arguments should be supplied even if only one is actually used. For the other methods, which do not have a formula argument, a character string specifying the argument name for the data should be supplied (the default is to use "data").

If x is supplied, on the other hand, the predictor matrix and the response are added as separate arguments to the function call. In this case, names should be a character vector of length two, with the first element specifying the argument name for the predictor matrix and the second element specifying the argument name for the response (the default is to use c("x", "y")). It should be noted that the formula or data arguments take precedence over x.

Value

An object of class "perry" with the following components:

pe a numeric vector containing the respective estimated prediction errors. In case of more than one replication, those are average values over all replications.

se a numeric vector containing the respective estimated standard errors of the prediction loss.

reps a numeric matrix in which each column contains the respective estimated prediction errors from all replications. This is only returned in case of more than one replication.

splits an object giving the data splits used to estimate the prediction error.

y the response.

yHat a list containing the predicted values from all replications.

call the matched function call.

Author(s)

Andreas Alfons

See Also

perrySelect, perryTuning, cvFolds, randomSplits, bootSamples, cost

Examples

```
data("coleman")
set.seed(1234)  # set seed for reproducibility

## via model fit
# fit an MM regression model
fit <- lmrob(Y ~ ., data=coleman)
# perform cross-validation
perryFit(fit, data = coleman, y = coleman$Y,
    splits = foldControl(K = 5, R = 10),
    cost = rtmspe, costArgs = list(trim = 0.1),
    seed = 1234)

## via model fitting function
# perform cross-validation
# note that the response is extracted from 'data' in
# this example and does not have to be supplied
perryFit(lmrob, formula = Y ~ ., data = coleman,
    splits = foldControl(K = 5, R = 10),
    cost = rtmspe, costArgs = list(trim = 0.1),
    seed = 1234)

## via function call
# set up function call
call <- call("lmrob", formula = Y ~ .)
# perform cross-validation
perryFit(call, data = coleman, y = coleman$Y,
    splits = foldControl(K = 5, R = 10),
    cost = rtmspe, costArgs = list(trim = 0.1),
    seed = 1234)
```

perryPlot *Plot resampling-based prediction error results*

Description

Plot results of resampling-based prediction error measures.

Usage

```
 perryPlot(x, ...)

 ## S3 method for class 'perry'
perryPlot(x,
   method = c("box", "density", "dot"), select = NULL,
   seFactor = NA, ...)

 ## S3 method for class 'perrySelect'
perryPlot(x,
   method = c("box", "density", "dot", "line"),
   subset = NULL, select = NULL, seFactor = x$seFactor,
   ...)

 ## Default S3 method:
perryPlot(x,
   method = c("box", "density", "dot", "line"), mapping,
   facets = attr(x, "facets"), ...)

 ## S3 method for class 'perry'
autoplot(object, ...)

 ## S3 method for class 'perrySelect'
autoplot(object, ...)

 ## S3 method for class 'perry'
plot(x, ...)

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

Arguments

`x` an object inheriting from class `"perry"` or `"perrySelect"` that contains prediction error results, or a data frame containing all necessary information for plotting (as generated by the corresponding `fortify` method).

`object` an object inheriting from class `"perry"` or `"perrySelect"`.

`method` a character string specifying the type of plot. Possible values are `"box"` to create a box plot, `"density"` to create a smooth density plot, `"dot"` to create a dot plot, or `"line"` to plot the (average) results for each model as a connected line (for objects inheriting from class `"perrySelect"`). Note that the first two plots are only meaningful in case of repeated resampling. The default is to use `"box"` in case of repeated resampling and `"dot"` otherwise. In any case, partial string matching allows supply abbreviations of the accepted values.

`subset` a character, integer or logical vector indicating the subset of models for which to plot the prediction error results.

`select` a character, integer or logical vector indicating the columns of prediction error results to be plotted.

`seFactor` a numeric value giving the multiplication factor of the standard error for displaying error bars in dot plots or line plots. Error bars in those plots can be suppressed by setting this to `NA`.

`mapping` an aesthetic mapping to override the default behavior (see `aes` or `aes_string`)

`facets` a faceting formula to override the default behavior. If supplied, `facet_wrap` or `facet_grid` is called depending on whether the formula is one-sided or two-sided.

... for the generic function perryPlot, additional arguments to be passed down to methods. For the "perry" and "perrySelect" methods of perryPlot, additional arguments to be passed down to the default method. For the default method of perryPlot, additional arguments to be passed down to geom_boxplot, geom_density, geom_pointrange or geom_line. For the methods of plot, additional arguments to be passed down to perryPlot.

Details

For objects with multiple columns of prediction error results, conditional plots are produced.

Value

An object of class "ggplot" (see ggplot).

Note

Duplicate indices in subset or select are removed such that all models and prediction error results are unique.

Author(s)

Andreas Alfons

See Also

perryFit, perrySelect, perryTuning, ggplot, autoplot, plot

Examples

```
data("coleman")
set.seed(1234)  # set seed for reproducibility

## set up folds for cross-validation
folds <- cvFolds(nrow(coleman), K = 5, R = 10)

## compare LS, MM and LTS regression

# perform cross-validation for an LS regression model
fitLm <- lm(Y ~ ., data = coleman)
cvLm <- perry(fitLm, splits = folds,
    cost = rtmspe, trim = 0.1)

# perform cross-validation for an MM regression model
fitLmrob <- lmrob(Y ~ ., data = coleman, k.max = 500)
cvLmrob <- perry(fitLmrob, splits = folds,
    cost = rtmspe, trim = 0.1)

# perform cross-validation for an LTS regression model
fitLts <- ltsReg(Y ~ ., data = coleman)
cvLts <- perry(fitLts, splits = folds,
    cost = rtmspe, trim = 0.1)
```

```
# combine results into one object
cv <- perrySelect(LS = cvLm, MM = cvLmrob, LTS = cvLts)
cv

# plot results for the MM regression model
plot(cvLmrob, method = "box")
plot(cvLmrob, method = "density")
plot(cvLmrob, method = "dot", seFactor = 1)

# plot combined results
plot(cv, method = "box")
plot(cv, method = "density")
plot(cv, method = "dot", seFactor = 1)
```

perryReshape *Reshape resampling-based prediction error results*

Description

Reshape resampling-based prediction error results into an object of class "perrySelect" with only one column of results.

Usage

```
perryReshape(x, selectBest = c("min", "hastie"),
  seFactor = 1, ...)
```

Arguments

`x` an object inheriting from class "perry" or "perrySelect" that contains prediction error results.

`selectBest` a character string specifying a criterion for selecting the best model. Possible values are "min" (the default) or "hastie". The former selects the model with the smallest prediction error. The latter is useful for nested models or for models with a tuning parameter controlling the complexity of the model (e.g., penalized regression). It selects the most parsimonious model whose prediction error is no larger than seFactor standard errors above the prediction error of the best overall model. Note that the models are thereby assumed to be ordered from the most parsimonious one to the most complex one. In particular a one-standard-error rule is frequently applied.

`seFactor` a numeric value giving a multiplication factor of the standard error for the selection of the best model. This is ignored if selectBest is "min".

`...` additional arguments to be passed down.

Value

An object of class `"perrySelect"` with the following components:

`splits` an object giving the data splits used to estimate the prediction error.

`best` an integer giving the index of the model with the best prediction performance.

`pe` a data frame containing the estimated prediction errors for the models. In case of more than one resampling replication, those are average values over all replications.

`se` a data frame containing the estimated standard errors of the prediction loss for the models.

`selectBest` a character string specifying the criterion used for selecting the best model.

`seFactor` a numeric value giving the multiplication factor of the standard error used for the selection of the best model.

`reps` a data frame containing the estimated prediction errors for the models from all replications. This is only returned in case of more than one resampling replication.

Author(s)

Andreas Alfons

References

Hastie, T., Tibshirani, R. and Friedman, J. (2009) *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2nd edition.

See Also

`perryFit`, `perrySelect`, `perryTuning`

Examples

```
data("coleman")

# perform cross-validation for an LTS regression model
fit <- ltsReg(Y ~ ., data = coleman)
folds <- foldControl(K = 5, R = 10)
cv <- perry(fit, splits = folds, fit = "both",
    cost = rtmspe, trim = 0.1, seed = 1234)

# compare original and reshaped object
cv
perryReshape(cv)
```

perrySelect *Model selection via resampling-based prediction error measures*

Description

Combine resampling-based prediction error results for various models into one object and select the model with the best prediction performance.

Usage

```
perrySelect(..., .list = list(...), .reshape = FALSE,
  .selectBest = c("min", "hastie"), .seFactor = 1)
```

Arguments

`...`	objects inheriting from class `"perry"` or `"perrySelect"` that contain prediction error results.
`.list`	a list of objects inheriting from class `"perry"` or `"perrySelect"`. If supplied, this is preferred over objects supplied via the ... argument.
`.reshape`	a logical indicating whether objects with more than one column of prediction error results should be reshaped to have only one column (see “Details”).
`.selectBest`	a character string specifying a criterion for selecting the best model. Possible values are `"min"` (the default) or `"hastie"`. The former selects the model with the smallest prediction error. The latter is useful for nested models or for models with a tuning parameter controlling the complexity of the model (e.g., penalized regression). It selects the most parsimonious model whose prediction error is no larger than `.seFactor` standard errors above the prediction error of the best overall model. Note that the models are thereby assumed to be ordered from the most parsimonious one to the most complex one. In particular a one-standard-error rule is frequently applied.
`.seFactor`	a numeric value giving a multiplication factor of the standard error for the selection of the best model. This is ignored if `.selectBest` is `"min"`.

Details

Keep in mind that objects inheriting from class `"perry"` or `"perrySelect"` may contain multiple columns of prediction error results. This is the case if the response is univariate but the function to compute predictions (usually the `predict` method of the fitted model) returns a matrix.

The `.reshape` argument determines how to handle such objects. If `.reshape` is `FALSE`, all objects are required to have the same number of columns and the best model for each column is selected. A typical use case for this behavior would be if the investigated models contain prediction error results for a raw and a reweighted fit. It might then be of interest to researchers to compare the best model for the raw estimators with the best model for the reweighted estimators.

If `.reshape` is `TRUE`, objects with more than one column of results are first transformed with `perryReshape` to have only one column. Then the best overall model is selected.

It should also be noted that the argument names of `.list`, `.reshape`, `.selectBest` and `.seFacor` start with a dot to avoid conflicts with the argument names used for the objects containing prediction error results.

Value

An object of class `"perrySelect"` with the following components:

`pe` a data frame containing the estimated prediction errors for the models. In case of more than one resampling replication, those are average values over all replications.

`se` a data frame containing the estimated standard errors of the prediction loss for the models.

`reps` a data frame containing the estimated prediction errors for the models from all replications. This is only returned in case of more than one resampling replication.

`splits` an object giving the data splits used to estimate the prediction error of the models.

`y` the response.

`yHat` a list containing the predicted values for the models. Each list component is again a list containing the corresponding predicted values from all replications.

`best` an integer vector giving the indices of the models with the best prediction performance.

`selectBest` a character string specifying the criterion used for selecting the best model.

`seFactor` a numeric value giving the multiplication factor of the standard error used for the selection of the best model.

Note

To ensure comparability, the prediction errors for all models are required to be computed from the same data splits.

Author(s)

Andreas Alfons

References

Hastie, T., Tibshirani, R. and Friedman, J. (2009) *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2nd edition.

See Also

`perryFit`, `perryTuning`

Examples

```
data("coleman")
set.seed(1234)  # set seed for reproducibility

## set up folds for cross-validation
folds <- cvFolds(nrow(coleman), K = 5, R = 10)

## compare LS, MM and LTS regression

# perform cross-validation for an LS regression model
fitLm <- lm(Y ~ ., data = coleman)
cvLm <- perry(fitLm, splits = folds,
    cost = rtmspe, trim = 0.1)

# perform cross-validation for an MM regression model
fitLmrob <- lmrob(Y ~ ., data = coleman)
cvLmrob <- perry(fitLmrob, splits = folds,
    cost = rtmspe, trim = 0.1)

# perform cross-validation for an LTS regression model
fitLts <- ltsReg(Y ~ ., data = coleman)
cvLts <- perry(fitLts, splits = folds,
    cost = rtmspe, trim = 0.1)

# compare cross-validation results
perrySelect(LS = cvLm, MM = cvLmrob, LTS = cvLts)
```

perrySplits *Data splits for resampling-based prediction error measures*

Description

Split observations or groups of observations into segments to be used for (repeated) K-fold cross-validation, (repeated) random splitting (also known as random subsampling or Monte Carlo cross-validation), or the bootstrap.

Usage

```
perrySplits(n, control)
```

Arguments

`n` an integer giving the number of observations to be split.

`control` a control object of class "`foldControl`" (as generated by `foldControl`), "`splitControl`" (as generated by `splitControl`) or "`bootControl`" (as generated by `bootControl`).

Value

For the foldControl method, an object of class "cvFolds" giving folds for (repeated) K-fold cross-validation (see cvFolds).

For the splitControl method, an object of class "randomSplits" giving random data splits (see randomSplits).

For the bootControl method, an object of class "bootSamples" giving bootstrap samples (see bootSamples).

Note

Users may prefer the wrapper functions cvFolds, randomSplits and bootSamples.

Author(s)

Andreas Alfons

See Also

foldControl, splitControl, bootControl, cvFolds, randomSplits, bootSamples

Examples

```
set.seed(1234)  # set seed for reproducibility

## data folds for K-fold cross-validation
perrySplits(20, foldControl(K = 5))
perrySplits(20, foldControl(K = 5, R = 10))

## random data splits
perrySplits(20, splitControl(m = 5))
perrySplits(20, splitControl(m = 5, R = 10))

## bootstrap samples
perrySplits(20, bootControl())
perrySplits(20, bootControl(R = 10))
```

---

perryTuning *Resampling-based prediction error for tuning parameter selection*

Description

Select tuning parameters of a model by estimating the respective prediction errors via (repeated) K-fold cross-validation, (repeated) random splitting (also known as random subsampling or Monte Carlo cross-validation), or the bootstrap. It is thereby possible to supply a model fitting function or an unevaluated function call to a model fitting function.

Usage

```
 perryTuning(object, ...)

 ## S3 method for class 'function'
perryTuning(object, formula,
   data = NULL, x = NULL, y, tuning = list(),
   args = list(), splits = foldControl(),
   predictFun = predict, predictArgs = list(),
   cost = rmspe, costArgs = list(),
   selectBest = c("min", "hastie"), seFactor = 1,
   final = FALSE, names = NULL, envir = parent.frame(),
   ncores = 1, cl = NULL, seed = NULL, ...)

 ## S3 method for class 'call'
perryTuning(object, data = NULL,
   x = NULL, y, tuning = list(), splits = foldControl(),
   predictFun = predict, predictArgs = list(),
   cost = rmspe, costArgs = list(),
   selectBest = c("min", "hastie"), seFactor = 1,
   final = FALSE, names = NULL, envir = parent.frame(),
   ncores = 1, cl = NULL, seed = NULL, ...)
```

Arguments

`object` a function or an unevaluated function call for fitting a model (see `call` for the latter).

`formula` a `formula` describing the model.

`data` a data frame containing the variables required for fitting the models. This is typically used if the model in the function call is described by a `formula`.

`x` a numeric matrix containing the predictor variables. This is typically used if the function call for fitting the models requires the predictor matrix and the response to be supplied as separate arguments.

`y` a numeric vector or matrix containing the response.

`tuning` a list of arguments giving the tuning parameter values to be evaluated. The names of the list components should thereby correspond to the argument names of the tuning parameters. For each tuning parameter, a vector of values can be supplied. The prediction error is then estimated for all possible combinations of tuning parameter values.

`args` a list of additional arguments to be passed to the model fitting function.

`splits` an object of class `"cvFolds"` (as returned by `cvFolds`) or a control object of class `"foldControl"` (see `foldControl`) defining the folds of the data for (repeated) K-fold cross-validation, an object of class `"randomSplits"` (as returned by `randomSplits`) or a control object of class `"splitControl"` (see `splitControl`) defining random data splits, or an object of class `"bootSamples"` (as returned by `bootSamples`) or a control object of class `"bootControl"` (see `bootControl`) defining bootstrap samples.

`predictFun` a function to compute predictions for the test data. It should expect the fitted model to be passed as the first argument and the test data as the second argument, and must return either a vector or a matrix containing the predicted values. The default is to use the `predict` method of the fitted model.

`predictArgs` a list of additional arguments to be passed to `predictFun`.

`cost` a cost function measuring prediction loss. It should expect the observed values of the response to be passed as the first argument and the predicted values as the second argument, and must return either a non-negative scalar value, or a list with the first component containing the prediction error and the second component containing the standard error. The default is to use the root mean squared prediction error (see `cost`).

`costArgs` a list of additional arguments to be passed to the prediction loss function `cost`.

`selectBest` a character string specifying a criterion for selecting the best model. Possible values are `"min"` (the default) or `"hastie"`. The former selects the model with the smallest prediction error. The latter is useful for models with a tuning parameter controlling the complexity of the model (e.g., penalized regression). It selects the most parsimonious model whose prediction error is no larger than `seFactor` standard errors above the prediction error of the best overall model. Note that the models are thereby assumed to be ordered from the most parsimonious one to the most complex one. In particular a one-standard-error rule is frequently applied.

`seFactor` a numeric value giving a multiplication factor of the standard error for the selection of the best model. This is ignored if `selectBest` is `"min"`.

`final` a logical indicating whether to fit the final model with the optimal combination of tuning parameters.

`names` an optional character vector giving names for the arguments containing the data to be used in the function call (see “Details”).

`envir` the `environment` in which to evaluate the function call for fitting the models (see `eval`).

`ncores` a positive integer giving the number of processor cores to be used for parallel computing (the default is 1 for no parallelization). If this is set to `NA`, all available processor cores are used.

`cl` a **parallel** cluster for parallel computing as generated by `makeCluster`. If supplied, this is preferred over `ncores`.

`seed` optional initial seed for the random number generator (see `.Random.seed`). Note that also in case of parallel computing, resampling is performed on the manager process rather than the worker processes. On the parallel worker processes, random number streams are used and the seed is set via `clusterSetRNGStream` for reproducibility in case the model fitting function involves randomness.

`...` additional arguments to be passed down.

Details

(Repeated) K-fold cross-validation is performed in the following way. The data are first split into K previously obtained blocks of approximately equal size (given by `folds`). Each of the K data blocks is left out once to fit the model, and predictions are computed for the observations in the left-out block with predictFun. Thus a prediction is obtained for each observation. The response variable and the obtained predictions for all observations are then passed to the prediction loss function cost to estimate the prediction error. For repeated K-fold cross-validation (as indicated by splits), this process is replicated and the estimated prediction errors from all replications are returned.

(Repeated) random splitting is performed similarly. In each replication, the data are split into a training set and a test set at random. Then the training data is used to fit the model, and predictions are computed for the test data. Hence only the response values from the test data and the corresponding predictions are passed to the prediction loss function cost.

For the bootstrap estimator, each bootstrap sample is used as training data to fit the model. The out-of-bag estimator uses the observations that do not enter the bootstrap sample as test data and computes the prediction loss function cost for those out-of-bag observations. The 0.632 estimator is computed as a linear combination of the out-of-bag estimator and the prediction loss of the fitted values of the model computed from the full sample.

In any case, if the response is a vector but predictFun returns a matrix, the prediction error is computed for each column. A typical use case for this behavior would be if predictFun returns predictions from an initial model fit and stepwise improvements thereof.

If formula or data are supplied, all variables required for fitting the models are added as one argument to the function call, which is the typical behavior of model fitting functions with a formula interface. In this case, the accepted values for names depend on the method. For the function method, a character vector of length two should supplied, with the first element specifying the argument name for the formula and the second element specifying the argument name for the data (the default is to use c("formula", "data")). Note that names for both arguments should be supplied even if only one is actually used. For the call method, which does not have a formula argument, a character string specifying the argument name for the data should be supplied (the default is to use "data").

If x is supplied, on the other hand, the predictor matrix and the response are added as separate arguments to the function call. In this case, names should be a character vector of length two, with the first element specifying the argument name for the predictor matrix and the second element specifying the argument name for the response (the default is to use c("x", "y")). It should be noted that the formula or data arguments take precedence over x.

Value

If tuning is an empty list, perryFit is called to return an object of class "perry".

Otherwise an object of class "perryTuning" (which inherits from class "perrySelect") with the following components is returned:

pe — a data frame containing the estimated prediction errors for all combinations of tuning parameter values. In case of more than one replication, those are average values over all replications.

se — a data frame containing the estimated standard errors of the prediction loss for all combinations of tuning parameter values.

reps — a data frame containing the estimated prediction errors from all replications for all combinations of tuning parameter values. This is only returned in case of more than one replication.

`splits` an object giving the data splits used to estimate the prediction error.

`y` the response.

`yHat` a list containing the predicted values for all combinations of tuning parameter values. Each list component is again a list containing the corresponding predicted values from all replications.

`best` an integer vector giving the indices of the optimal combinations of tuning parameters.

`selectBest` a character string specifying the criterion used for selecting the best model.

`seFactor` a numeric value giving the multiplication factor of the standard error used for the selection of the best model.

`tuning` a data frame containing the grid of tuning parameter values for which the prediction error was estimated.

`finalModel` the final model fit with the optimal combination of tuning parameters. This is only returned if argument `final` is `TRUE`.

`call` the matched function call.

Note

The same data splits are used for all combinations of tuning parameter values for maximum comparability.

If a final model with the optimal combination of tuning parameters is computed, class `"perryTuning"` inherits the `coef()`, `fitted()`, `predict()` and `residuals()` methods from its component `finalModel`.

Author(s)

Andreas Alfons

References

Hastie, T., Tibshirani, R. and Friedman, J. (2009) *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* Springer, 2nd edition.

See Also

`perryFit`, `perrySelect`, `cvFolds`, `randomSplits`, `bootSamples`, `cost`

Examples

```
data("coleman")

## evaluate MM regression models tuned for 85% and 95% efficiency
tuning <- list(tuning.psi = c(3.443689, 4.685061))

## via model fitting function
# perform cross-validation
# note that the response is extracted from 'data' in
# this example and does not have to be supplied
perryTuning(lmrob, formula = Y ~ ., data = coleman,
    tuning = tuning, splits = foldControl(K = 5, R = 10),
    cost = rtmspe, costArgs = list(trim = 0.1), seed = 1234)

## via function call
# set up function call
call <- call("lmrob", formula = Y ~ .)
# perform cross-validation
perryTuning(call, data = coleman, y = coleman$Y,
    tuning = tuning, splits = foldControl(K = 5, R = 10),
    cost = rtmspe, costArgs = list(trim = 0.1), seed = 1234)
```

---

randomSplits *Random data splits*

Description

Split observations or groups of observations into training and test data to be used for (repeated) random splitting (also known as random subsampling or Monte Carlo cross-validation).

Usage

```
randomSplits(n, m, R = 1, grouping = NULL)
```

Arguments

`n` an integer giving the number of observations to be split into training and test data. This is ignored if `grouping` is supplied in order to split groups of observations into folds.

`m` an integer giving the number of observations or groups of observations to be used as test data.

`R` an integer giving the number of random data splits.

`grouping` a factor specifying groups of observations. If supplied, the data are split according to the groups rather than individual observations such that all observations within a group belong either to the training or test data.

Value

An object of class `"randomSplits"` with the following components:

`n` an integer giving the number of observations or groups.

`m` an integer giving the number of observations or groups in the test data.

`R` an integer giving the number of random data splits.

`subsets` an integer matrix in which each column contains the indices of the observations or groups in the test data of the corresponding random data split.

`grouping` a list giving the indices of the observations belonging to each group. This is only returned if a grouping factor has been supplied.

Note

This is a simple wrapper function for perrySplits with a control object generated by splitControl.

Author(s)

Andreas Alfons

See Also

perrySplits, splitControl, cvFolds, bootSamples

Examples

```
set.seed(1234)  # set seed for reproducibility
randomSplits(20, m = 5)
randomSplits(20, m = 5, R = 10)
```

---

reperry *Recompute resampling-based prediction error measures*

Description

Recompute prediction error measures for previously obtained objects that contain resampling-based prediction error results. This is useful for computing a different measure of prediction loss.

Usage

```
  reperry(object, ...)

  ## S3 method for class 'perry'
 reperry(object, cost = rmspe, ...)

  ## S3 method for class 'perrySelect'
 reperry(object, cost = rmspe, ...)
```

Arguments

`object` an object inheriting from class "perry" or "perrySelect" that contains prediction error results.

`cost` a cost function measuring prediction loss. It should expect the observed values of the response to be passed as the first argument and the predicted values as the second argument, and must return either a non-negative scalar value, or a list with the first component containing the prediction error and the second component containing the standard error. The default is to use the root mean squared prediction error (see cost).

`...` for the generic function, additional arguments to be passed down to methods. For the methods, additional arguments to be passed to the prediction loss function cost.

Value

An object similar to `object` containing the results for the new measure of prediction loss.

Author(s)

Andreas Alfons

See Also

`perryFit`, `perryTuning`, `perrySelect`

Examples

```
data("coleman")
set.seed(1234)  # set seed for reproducibility

## set up folds for cross-validation
folds <- cvFolds(nrow(coleman), K = 5, R = 10)

## compare raw and reweighted LTS estimators for 50% and 75%
## subsets based on their RTMSPE with 25% trimming

# 50% subsets
fit50 <- ltsReg(Y ~ ., data = coleman, alpha = 0.5)
cv50 <- perry(fit50, splits = folds, fit = "both",
    cost = rtmspe, trim = 0.25)

# 75% subsets
fit75 <- ltsReg(Y ~ ., data = coleman, alpha = 0.75)
cv75 <- perry(fit75, splits = folds, fit = "both",
    cost = rtmspe, trim = 0.25)

# combine results into one object
cv <- perrySelect("0.5" = cv50, "0.75" = cv75)
cv

## recompute the RTMSPE with 10% trimming
reperry(cv50, cost = rtmspe, trim = 0.1)
reperry(cv, cost = rtmspe, trim = 0.1)
```

`splitControl` *Control object for random data splits*

Description

Generate an object that controls how to split n observations or groups of observations into training and test data to be used for (repeated) random splitting (also known as random subsampling or Monte Carlo cross-validation).

Usage

```
splitControl(m, R = 1, grouping = NULL)
```

Arguments

m an integer giving the number of observations or groups of observations to be used as test data.

R an integer giving the number of random data splits.

grouping a factor specifying groups of observations.

Value

An object of class "splitControl" with the following components:

m an integer giving the number of observations or groups of observations to be used as test data.

R an integer giving the number of random data splits.

grouping if supplied, a factor specifying groups of observations. The data will then be split according to the groups rather than individual observations such that all observations within a group belong either to the training or test data.

Author(s)

Andreas Alfons

See Also

perrySplits, randomSplits, foldControl, bootControl

Examples

```
set.seed(1234)  # set seed for reproducibility
perrySplits(20, splitControl(m = 5))
perrySplits(20, splitControl(m = 5, R = 10))
```

subset.perry *Subsetting resampling-based prediction error results*

Description

Extract subsets of resampling-based prediction error results.

Usage

```
 ## S3 method for class 'perry'
subset(x, select = NULL, ...)

 ## S3 method for class 'perrySelect'
subset(x, subset = NULL,
   select = NULL, ...)
```

Arguments

x an object inheriting from class "perry" or "perrySelect" that contains prediction error results.

subset a character, integer or logical vector indicating the subset of models for which to keep the prediction error results.

select a character, integer or logical vector indicating the prediction error results to be extracted.

... currently ignored.

Value

An object similar to x containing just the selected results.

Note

Duplicate indices in subset or select are removed such that all models and prediction error results are unique.

Author(s)

Andreas Alfons

See Also

perryFit, perrySelect, perryTuning, subset

Examples

```
data("coleman")
set.seed(1234)  # set seed for reproducibility

## set up folds for cross-validation
folds <- cvFolds(nrow(coleman), K = 5, R = 10)

## compare raw and reweighted LTS estimators for
## 50% and 75% subsets

# 50% subsets
fit50 <- ltsReg(Y ~ ., data = coleman, alpha = 0.5)
cv50 <- perry(fit50, splits = folds, fit = "both",
    cost = rtmspe, trim = 0.1)

# 75% subsets
fit75 <- ltsReg(Y ~ ., data = coleman, alpha = 0.75)
cv75 <- perry(fit75, splits = folds, fit = "both",
    cost = rtmspe, trim = 0.1)

# combine results into one object
cv <- perrySelect("0.5" = cv50, "0.75" = cv75)
cv

# extract reweighted LTS results with 50% subsets
subset(cv50, select = "reweighted")
subset(cv, subset = c(TRUE, FALSE), select = "reweighted")
```

`summary.perry` *Summarize resampling-based prediction error results*

Description

Produce a summary of resampling-based prediction error results.

Usage

```
## S3 method for class 'perry'
summary(object, ...)

## S3 method for class 'perrySelect'
summary(object, ...)

## S3 method for class 'perryTuning'
summary(object, ...)
```

Arguments

`object`	an object inheriting from class `"perry"` or `"perrySelect"` that contains prediction error results (note that the latter includes objects of class `"perryTuning"`).
`...`	currently ignored.

Value

An object of class `"summary.perry"`, `"summary.perrySelect"` or `"summary.perryTuning"`, depending on the class of `object`.

Author(s)

Andreas Alfons

See Also

perryFit, perrySelect, perryTuning, summary

Examples

```
data("coleman")
set.seed(1234)  # set seed for reproducibility

## set up folds for cross-validation
folds <- cvFolds(nrow(coleman), K = 5, R = 10)

## compare raw and reweighted LTS estimators for
## 50% and 75% subsets

# 50% subsets
fit50 <- ltsReg(Y ~ ., data = coleman, alpha = 0.5)
cv50 <- perry(fit50, splits = folds, fit = "both",
    cost = rtmspe, trim = 0.1)

# 75% subsets
fit75 <- ltsReg(Y ~ ., data = coleman, alpha = 0.75)
cv75 <- perry(fit75, splits = folds, fit = "both",
    cost = rtmspe, trim = 0.1)

# combine results into one object
cv <- perrySelect("0.5" = cv50, "0.75" = cv75)
cv

# summary of the results with the 50% subsets
summary(cv50)
# summary of the combined results
summary(cv)
```

Index