Package 'cvTools'

July 22, 2025

Type Package

Title Cross-Validation Tools for Regression Models

Version 0.3.3

Date 2024-03-13

Depends R (>= 2.11.0), lattice, robustbase

Imports stats

Description Tools that allow developers to write functions for cross-validation with minimal programming effort and assist users with model selection.

License GPL (>= 2)

LazyLoad yes

Author Andreas Alfons [aut, cre] (ORCID: <https://orcid.org/0000-0002-2513-3788>)

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

Encoding UTF-8

RoxygenNote 7.2.3

NeedsCompilation no

Repository CRAN

Date/Publication 2024-03-13 12:40:04 UTC

Contents

`cvTools-package` *Cross-Validation Tools for Regression Models*

Description

Tools that allow developers to write functions for cross-validation with minimal programming effort and assist users with model selection.

Details

The DESCRIPTION file:

Package:	cvTools
Type:	Package
Title:	Cross-Validation Tools for Regression Models
Version:	0.3.3
Date:	2024-03-13
Depends:	R (>= 2.11.0), lattice, robustbase
Imports:	stats
Description:	Tools that allow developers to write functions for cross-validation with minimal programming effort and assi
License:	GPL (>= 2)
LazyLoad:	yes
Authors@R:	person("Andreas", "Alfons", email = "alfons@ese.eur.nl", role = c("aut", "cre"), comment = c(ORCID = "00
Author:	Andreas Alfons [aut, cre] (<https://orcid.org/0000-0002-2513-3788>)
Maintainer:	Andreas Alfons <alfons@ese.eur.nl>
Encoding:	UTF-8
RoxygenNote:	7.2.3

Index of help topics:

```
accessors               Access or set information on cross-validation
                        results
aggregate.cv            Aggregate cross-validation results
bwplot.cv               Box-and-whisker plots of cross-validation
                        results
```

cost	Prediction loss
cvFit	Cross-validation for model evaluation
cvFolds	Cross-validation folds
cvReshape	Reshape cross-validation results
cvSelect	Model selection based on cross-validation
cvTool	Low-level function for cross-validation
cvTools-package	Cross-Validation Tools for Regression Models
cvTuning	Cross-validation for tuning parameter selection
densityplot.cv	Kernel density plots of cross-validation results
dotplot.cv	Dot plots of cross-validation results
plot.cv	Plot cross-validation results
repCV	Cross-validation for linear models
subset.cv	Subsetting cross-validation results
summary.cv	Summarize cross-validation results
xyplot.cv	X-Y plots of cross-validation results

Author(s)

Andreas Alfons [aut, cre] (<https://orcid.org/0000-0002-2513-3788>)

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

`accessors` *Access or set information on cross-validation results*

Description

Retrieve or set the names of cross-validation results, retrieve or set the identifiers of the models, or retrieve the number of cross-validation results or included models.

Usage

```
cvNames(x)

cvNames(x) <- value

fits(x)

fits(x) <- value

ncv(x)

nfits(x)
```

Arguments

x	an object inheriting from class "cv" or "cvSelect" that contains cross-validation results.
value	a vector of replacement values.

Value

cvNames returns the names of the cross-validation results. The replacement function thereby returns them invisibly.

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

ncv returns the number of cross-validation results.

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

Author(s)

Andreas Alfons

See Also

cvFit, cvSelect, cvTuning

Examples

```
library("robustbase")
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
fitLts50 <- ltsReg(Y ~ ., data = coleman, alpha = 0.5)
cvFitLts50 <- cvLts(fitLts50, cost = rtmspe, folds = folds,
    fit = "both", trim = 0.1)

# 75% subsets
fitLts75 <- ltsReg(Y ~ ., data = coleman, alpha = 0.75)
cvFitLts75 <- cvLts(fitLts75, cost = rtmspe, folds = folds,
    fit = "both", trim = 0.1)

# combine results into one object
cvFitsLts <- cvSelect("0.5" = cvFitLts50, "0.75" = cvFitLts75)
cvFitsLts
```

```
# "cv" object
ncv(cvFitLts50)
nfits(cvFitLts50)
cvNames(cvFitLts50)
cvNames(cvFitLts50) <- c("improved", "initial")
fits(cvFitLts50)
cvFitLts50

# "cvSelect" object
ncv(cvFitsLts)
nfits(cvFitsLts)
cvNames(cvFitsLts)
cvNames(cvFitsLts) <- c("improved", "initial")
fits(cvFitsLts)
fits(cvFitsLts) <- 1:2
cvFitsLts
```

`aggregate.cv` *Aggregate cross-validation results*

Description

Compute summary statistics of results from repeated K-fold cross-validation.

Usage

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

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

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

Arguments

`x`	an object inheriting from class `"cv"` or `"cvSelect"` that contains cross-validation results (note that the latter includes objects of class `"cvTuning"`).
`FUN`	a function to compute the summary statistics.
`select`	a character, integer or logical vector indicating the columns of cross-validation results for which to compute the summary statistics.
`...`	for the `"cvTuning"` method, additional arguments to be passed to the `"cvSelect"` method. Otherwise additional arguments to be passed to `FUN`.

Value

The "cv" method returns a vector or matrix of aggregated cross-validation results, depending on whether FUN returns a single value or a vector.

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

Author(s)

Andreas Alfons

See Also

cvFit, cvSelect, cvTuning, aggregate

Examples

```
library("robustbase")
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
fitLts50 <- ltsReg(Y ~ ., data = coleman, alpha = 0.5)
cvFitLts50 <- cvLts(fitLts50, cost = rtmspe, folds = folds,
    fit = "both", trim = 0.1)

# 75% subsets
fitLts75 <- ltsReg(Y ~ ., data = coleman, alpha = 0.75)
cvFitLts75 <- cvLts(fitLts75, cost = rtmspe, folds = folds,
    fit = "both", trim = 0.1)

# combine results into one object
cvFitsLts <- cvSelect("0.5" = cvFitLts50, "0.75" = cvFitLts75)
cvFitsLts

# summary of the results with the 50% subsets
aggregate(cvFitLts50, summary)
# summary of the combined results
aggregate(cvFitsLts, summary)
```

bwplot.cv *Box-and-whisker plots of cross-validation results*

Description

Produce box-and-whisker plots of results from repeated K-fold cross-validation.

Usage

```
## S3 method for class 'cv'
bwplot(x, data, select = NULL, ...)

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

Arguments

x	an object inheriting from class "cv" or "cvSelect" that contains cross-validation results.
data	currently ignored.
select	a character, integer or logical vector indicating the columns of cross-validation results to be plotted.
...	additional arguments to be passed to the "formula" method of bwplot.
subset	a character, integer or logical vector indicating the subset of models for which to plot the cross-validation results.

Details

For objects with multiple columns of repeated cross-validation results, conditional box-and-whisker plots are produced.

Value

An object of class "trellis" is returned invisibly. The update method can be used to update components of the object and the print method (usually called by default) will plot it on an appropriate plotting device.

Author(s)

Andreas Alfons

See Also

cvFit, cvSelect, cvTuning, plot, densityplot, xyplot, dotplot

Examples

```
library("robustbase")
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)
cvFitLm <- cvLm(fitLm, cost = rtmspe,
    folds = folds, trim = 0.1)

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

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

# combine results into one object
cvFits <- cvSelect(LS = cvFitLm, MM = cvFitLmrob, LTS = cvFitLts)
cvFits

# plot results for the MM regression model
bwplot(cvFitLmrob)
# plot combined results
bwplot(cvFits)
```

`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.
includeSE	a logical indicating whether standard errors should be computed as well.
trim	a numeric value giving the trimming proportion (the default is 0.25).

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

cvFit, cvTuning

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)
```

`cvFit` *Cross-validation for model evaluation*

Description

Estimate the prediction error of a model via (repeated) K-fold cross-validation. 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

```
cvFit(object, ...)

## Default S3 method:
cvFit(
  object,
  data = NULL,
  x = NULL,
  y,
  cost = rmspe,
  K = 5,
  R = 1,
  foldType = c("random", "consecutive", "interleaved"),
  grouping = NULL,
  folds = NULL,
  names = NULL,
```

```
  predictArgs = list(),
  costArgs = list(),
  envir = parent.frame(),
  seed = NULL,
  ...
)

## S3 method for class '`function`'
cvFit(
  object,
  formula,
  data = NULL,
  x = NULL,
  y,
  args = list(),
  cost = rmspe,
  K = 5,
  R = 1,
  foldType = c("random", "consecutive", "interleaved"),
  grouping = NULL,
  folds = NULL,
  names = NULL,
  predictArgs = list(),
  costArgs = list(),
  envir = parent.frame(),
  seed = NULL,
  ...
)

## S3 method for class 'call'
cvFit(
  object,
  data = NULL,
  x = NULL,
  y,
  cost = rmspe,
  K = 5,
  R = 1,
  foldType = c("random", "consecutive", "interleaved"),
  grouping = NULL,
  folds = NULL,
  names = NULL,
  predictArgs = list(),
  costArgs = list(),
  envir = parent.frame(),
  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.
...	additional arguments to be passed down.
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.
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).
K	an integer giving the number of folds into which the data should be split (the default is five). Keep in mind that this should be chosen such that all folds are of approximately equal size. 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.
foldType	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.
folds	an object of class "cvFolds" giving the folds of the data for cross-validation (as returned by cvFolds). If supplied, this is preferred over the arguments for generating cross-validation folds.
names	an optional character vector giving names for the arguments containing the data to be used in the function call (see "Details").
predictArgs	a list of additional arguments to be passed to the predict method of the fitted models.
costArgs	a list of additional arguments to be passed to the prediction loss function cost.
envir	the environment in which to evaluate the function call for fitting the models (see eval).
seed	optional initial seed for the random number generator (see .Random.seed).
formula	a formula describing the model.
args	a list of additional arguments to be passed to the model fitting function.

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. 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 the `predict` method of the fitted model. 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 cross-validation, this process is replicated and the estimated prediction errors from all replications as well as their average are included in the returned object.

Furthermore, if the response is a vector but the `predict` method of the fitted models returns a matrix, the prediction error is computed for each column. A typical use case for this behavior would be if the `predict` method 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 `"cv"` 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.
`cv`	a numeric vector containing the respective estimated prediction errors. For repeated cross-validation, 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 for repeated cross-validation.
`seed`	the seed of the random number generator before cross-validation was performed.
`call`	the matched function call.

Author(s)

Andreas Alfons

See Also

cvTool, cvSelect, cvTuning, cvFolds, cost

Examples

```
library("robustbase")
data("coleman")

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

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

---

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.

Author(s)

Andreas Alfons

See Also

cvFit, cvSelect, cvTuning

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)
```

cvReshape *Reshape cross-validation results*

Description

Reshape cross-validation results into an object of class "cvSelect" with only one column of results.

Usage

```
cvReshape(x, ...)

## S3 method for class 'cv'
cvReshape(x, selectBest = c("min", "hastie"), seFactor = 1, ...)

## S3 method for class 'cvSelect'
cvReshape(x, selectBest = c("min", "hastie"), seFactor = 1, ...)
```

Arguments

x	an object inheriting from class "cv" or "cvSelect" that contains cross-validation results.
...	additional arguments to be passed down.
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".

Value

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

n	an integer giving the number of observations.
K	an integer giving the number of folds used in cross-validation.
R	an integer giving the number of replications used in cross-validation.
best	an integer giving the index of the model with the best prediction performance.
cv	a data frame containing the estimated prediction errors for the models. For repeated cross-validation, 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 if repeated cross-validation was performed.

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

cvFit, cvSelect, cvTuning

Examples

```
library("robustbase")
data("coleman")

# perform cross-validation for an LTS regression model
fitLts <- ltsReg(Y ~ ., data = coleman)
cvFitLts <- cvLts(fitLts, cost = rtmspe, K = 5, R = 10,
    fit = "both", trim = 0.1, seed = 1234)
# compare original and reshaped object
cvFitLts
cvReshape(cvFitLts)
```

---

cvSelect *Model selection based on cross-validation*

Description

Combine cross-validation results for various models into one object and select the model with the best prediction performance.

Usage

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

Arguments

`...`	objects inheriting from class `"cv"` or `"cvSelect"` that contain cross-validation results.
`.reshape`	a logical indicating whether objects with more than one column of cross-validation 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 `"cv"` or `"cvSelect"` may contain multiple columns of cross-validation results. This is the case if the response is univariate but 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 cross-validation 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 `cvReshape` to have only one column. Then the best overall model is selected.

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

Value

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

`n`	an integer giving the number of observations.
`K`	an integer vector giving the number of folds used in cross-validation for the respective model.
`R`	an integer vector giving the number of replications used in cross-validation for the respective model.
`best`	an integer vector giving the indices of the models with the best prediction performance.

cv	a data frame containing the estimated prediction errors for the models. For models for which repeated cross-validation was performed, 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 from all replications for those models for which repeated cross-validation was performed. This is only returned if repeated cross-validation was performed for at least one of the models.

Note

Even though the function allows to compare cross-validation results obtained with a different number of folds or a different number of replications, such comparisons should be made with care. Hence warnings are issued in those cases. For maximum comparability, the same data folds should be used in cross-validation for all models to be compared.

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

cvFit, cvTuning

Examples

```
library("robustbase")
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)
cvFitLm <- cvLm(fitLm, cost = rtmspe,
    folds = folds, trim = 0.1)
```

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

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

# compare cross-validation results
cvSelect(LS = cvFitLm, MM = cvFitLmrob, LTS = cvFitLts)
```

`cvTool` *Low-level function for cross-validation*

Description

Basic function to estimate the prediction error of a model via (repeated) K-fold cross-validation. The model is thereby specified by an unevaluated function call to a model fitting function.

Usage

```
cvTool(
  call,
  data = NULL,
  x = NULL,
  y,
  cost = rmspe,
  folds,
  names = NULL,
  predictArgs = list(),
  costArgs = list(),
  envir = parent.frame()
)
```

Arguments

`call`	an unevaluated function call for fitting a model (see `call`).
`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.

`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`).
`folds`	an object of class `"cvFolds"` giving the folds of the data for cross-validation (as returned by `cvFolds`).
`names`	an optional character vector giving names for the arguments containing the data to be used in the function call (see "Details").
`predictArgs`	a list of additional arguments to be passed to the `predict` method of the fitted models.
`costArgs`	a list of additional arguments to be passed to the prediction loss function `cost`.
`envir`	the `environment` in which to evaluate the function call for fitting the models (see `eval`).

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 the `predict` method of the fitted model. 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 cross-validation (as indicated by `folds`), this process is replicated and the estimated prediction errors from all replications are returned.

Furthermore, if the response is a vector but the `predict` method of the fitted models returns a matrix, the prediction error is computed for each column. A typical use case for this behavior would be if the `predict` method returns predictions from an initial model fit and stepwise improvements thereof.

If `data` is 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, a character string specifying the argument name can be passed via `names` (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 `data` takes precedence over `x` if both are supplied.

Value

If only one replication is requested and the prediction loss function `cost` also returns the standard error, a list is returned, with the first component containing the estimated prediction errors and the second component the corresponding estimated standard errors.

Otherwise the return value is a numeric matrix in which each column contains the respective estimated prediction errors from all replications.

Author(s)

Andreas Alfons

See Also

cvFit, cvTuning, cvFolds, cost

Examples

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

# set up function call for an MM regression model
call <- call("lmrob", formula = Y ~ .)
# set up folds for cross-validation
folds <- cvFolds(nrow(coleman), K = 5, R = 10)

# perform cross-validation
cvTool(call, data = coleman, y = coleman$Y, cost = rtmspe,
    folds = folds, costArgs = list(trim = 0.1))
```

---

cvTuning *Cross-validation for tuning parameter selection*

Description

Select tuning parameters of a model by estimating the respective prediction errors via (repeated) K-fold cross-validation. It is thereby possible to supply a model fitting function or an unevaluated function call to a model fitting function.

Usage

```
cvTuning(object, ...)

## S3 method for class '`function`'
cvTuning(
  object,
  formula,
  data = NULL,
  x = NULL,
  y,
  tuning = list(),
  args = list(),
  cost = rmspe,
```

```
  K = 5,
  R = 1,
  foldType = c("random", "consecutive", "interleaved"),
  grouping = NULL,
  folds = NULL,
  names = NULL,
  predictArgs = list(),
  costArgs = list(),
  selectBest = c("min", "hastie"),
  seFactor = 1,
  envir = parent.frame(),
  seed = NULL,
  ...
)

## S3 method for class 'call'
cvTuning(
  object,
  data = NULL,
  x = NULL,
  y,
  tuning = list(),
  cost = rmspe,
  K = 5,
  R = 1,
  foldType = c("random", "consecutive", "interleaved"),
  grouping = NULL,
  folds = NULL,
  names = NULL,
  predictArgs = list(),
  costArgs = list(),
  selectBest = c("min", "hastie"),
  seFactor = 1,
  envir = parent.frame(),
  seed = NULL,
  ...
)
```

Arguments

object	a function or an unevaluated function call for fitting a model (see call for the latter).
...	additional arguments to be passed down.
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. Cross-validation is then applied over the grid of all possible combinations of tuning parameter values.

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

`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`).

`K` an integer giving the number of folds into which the data should be split (the default is five). Keep in mind that this should be chosen such that all folds are of approximately equal size. 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.

`foldType` 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.

`folds` an object of class `"cvFolds"` giving the folds of the data for cross-validation (as returned by `cvFolds`). If supplied, this is preferred over the arguments for generating cross-validation folds.

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

`predictArgs` a list of additional arguments to be passed to the `predict` method of the fitted models.

`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"`.

envir	the environment in which to evaluate the function call for fitting the models (see eval).
seed	optional initial seed for the random number generator (see .Random.seed).

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. 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 the predict method of the fitted model. 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 cross-validation, this process is replicated and the estimated prediction errors from all replications as well as their average are included in the returned object.

Furthermore, if the response is a vector but the predict method of the fitted models returns a matrix, the prediction error is computed for each column. A typical use case for this behavior would be if the predict method 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, cvFit is called to return an object of class "cv".

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

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.
tuning	a data frame containing the grid of tuning parameter values for which the prediction error was estimated.
best	an integer vector giving the indices of the optimal combinations of tuning parameters.

cv	a data frame containing the estimated prediction errors for all combinations of tuning parameter values. For repeated cross-validation, 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.
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 from all replications for all combinations of tuning parameter values. This is only returned for repeated cross-validation.
seed	the seed of the random number generator before cross-validation was performed.
call	the matched function call.

Note

The same cross-validation folds are used for all combinations of tuning parameter values for maximum comparability.

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

cvTool, cvFit, cvSelect, cvFolds, cost

Examples

```
library("robustbase")
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
cvTuning(lmrob, formula = Y ~ ., data = coleman, tuning = tuning,
    cost = rtmspe, K = 5, R = 10, costArgs = list(trim = 0.1),
    seed = 1234)

## via function call
```

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

`densityplot.cv` *Kernel density plots of cross-validation results*

Description

Produce kernel density plots of results from repeated K-fold cross-validation.

Usage

```
## S3 method for class 'cv'
densityplot(x, data, select = NULL, ...)

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

Arguments

`x`	an object inheriting from class `"cv"` or `"cvSelect"` that contains cross-validation results.
`data`	currently ignored.
`select`	a character, integer or logical vector indicating the columns of cross-validation results to be plotted.
`...`	additional arguments to be passed to the `"formula"` method of `densityplot`.
`subset`	a character, integer or logical vector indicating the subset of models for which to plot the cross-validation results.

Details

For objects with multiple columns of repeated cross-validation results, conditional kernel density plots are produced.

Value

An object of class `"trellis"` is returned invisibly. The `update` method can be used to update components of the object and the `print` method (usually called by default) will plot it on an appropriate plotting device.

Author(s)

Andreas Alfons

See Also

cvFit, cvSelect, cvTuning, plot, bwplot, xyplot, dotplot

Examples

```
library("robustbase")
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)
cvFitLm <- cvLm(fitLm, cost = rtmspe,
    folds = folds, trim = 0.1)

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

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

# combine results into one object
cvFits <- cvSelect(LS = cvFitLm, MM = cvFitLmrob, LTS = cvFitLts)
cvFits

# plot results for the MM regression model
densityplot(cvFitLmrob)
# plot combined results
densityplot(cvFits)
```

---

dotplot.cv *Dot plots of cross-validation results*

Description

Produce dot plots of (average) results from (repeated) K-fold cross-validation.

Usage

```
## S3 method for class 'cv'
dotplot(x, data, select = NULL, seFactor = NA, ...)

## S3 method for class 'cvSelect'
dotplot(x, data, subset = NULL, select = NULL, seFactor = x$seFactor, ...)
```

Arguments

x	an object inheriting from class "cvSelect" that contains cross-validation results.
data	currently ignored.
select	a character, integer or logical vector indicating the columns of cross-validation results to be plotted.
seFactor	a numeric value giving the multiplication factor of the standard error for displaying error bars. Error bars can be suppressed by setting this to NA.
...	additional arguments to be passed to the "formula" method of dotplot.
subset	a character, integer or logical vector indicating the subset of models for which to plot the cross-validation results.

Details

For objects with multiple columns of repeated cross-validation results, conditional dot plots are produced.

Value

An object of class "trellis" is returned invisibly. The update method can be used to update components of the object and the print method (usually called by default) will plot it on an appropriate plotting device.

Author(s)

Andreas Alfons

See Also

cvFit, cvSelect, cvTuning, plot, xyplot, bwplot, densityplot

Examples

```
library("robustbase")
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)
cvFitLm <- cvLm(fitLm, cost = rtmspe,
    folds = folds, trim = 0.1)

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

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

# combine and plot results
cvFits <- cvSelect(LS = cvFitLm, MM = cvFitLmrob, LTS = cvFitLts)
cvFits
dotplot(cvFits)
```

`plot.cv` *Plot cross-validation results*

Description

Plot results from (repeated) K-fold cross-validation.

Usage

```
## S3 method for class 'cv'
plot(x, method = c("bwplot", "densityplot", "xyplot", "dotplot"), ...)

## S3 method for class 'cvSelect'
plot(x, method = c("bwplot", "densityplot", "xyplot", "dotplot"), ...)
```

Arguments

`x`	an object inheriting from class `"cv"` or `"cvSelect"` that contains cross-validation results.
`method`	a character string specifying the type of plot. For the `"cv"` method, possible values are `"bwplot"` to create a box-and-whisker plot via `bwplot` (the default), or `"densityplot"` to create a kernel density plot via `densityplot`. Note that those plots are only meaningful for results from repeated cross-validation. For the `"cvSelect"` method, there are two additional possibilities: `"xyplot"` to plot the (average) results for each model via `xyplot`, or `"dotplot"` to create a similar dot plot via `dotplot`. The default is to use `"bwplot"` for results from repeated

cross-validation and `"xyplot"` otherwise. In any case, partial string matching allows supply abbreviations of the accepted values.

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

Details

For objects with multiple columns of cross-validation results, conditional plots are produced.

Value

An object of class `"trellis"` is returned invisibly. The `update` method can be used to update components of the object and the `print` method (usually called by default) will plot it on an appropriate plotting device.

Author(s)

Andreas Alfons

See Also

`cvFit`, `cvSelect`, `cvTuning`, `bwplot`, `densityplot`, `xyplot`, `dotplot`

Examples

```
library("robustbase")
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)
cvFitLm <- cvLm(fitLm, cost = rtmspe,
    folds = folds, trim = 0.1)

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

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

# combine results into one object
cvFits <- cvSelect(LS = cvFitLm, MM = cvFitLmrob, LTS = cvFitLts)
cvFits
```

```
# plot results for the MM regression model
plot(cvFitLmrob, method = "bw")
plot(cvFitLmrob, method = "density")

# plot combined results
plot(cvFits, method = "bw")
plot(cvFits, method = "density")
plot(cvFits, method = "xy")
plot(cvFits, method = "dot")
```

---

repCV *Cross-validation for linear models*

Description

Estimate the prediction error of a linear model via (repeated) K-fold cross-validation. Cross-validation functions 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

```
repCV(object, ...)

## S3 method for class 'lm'
repCV(
  object,
  cost = rmspe,
  K = 5,
  R = 1,
  foldType = c("random", "consecutive", "interleaved"),
  grouping = NULL,
  folds = NULL,
  seed = NULL,
  ...
)

## S3 method for class 'lmrob'
repCV(
  object,
  cost = rtmspe,
  K = 5,
  R = 1,
  foldType = c("random", "consecutive", "interleaved"),
  grouping = NULL,
  folds = NULL,
```

```
  seed = NULL,
  ...
)

## S3 method for class 'lts'
repCV(
  object,
  cost = rtmspe,
  K = 5,
  R = 1,
  foldType = c("random", "consecutive", "interleaved"),
  grouping = NULL,
  folds = NULL,
  fit = c("reweighted", "raw", "both"),
  seed = NULL,
  ...
)

cvLm(
  object,
  cost = rmspe,
  K = 5,
  R = 1,
  foldType = c("random", "consecutive", "interleaved"),
  grouping = NULL,
  folds = NULL,
  seed = NULL,
  ...
)

cvLmrob(
  object,
  cost = rtmspe,
  K = 5,
  R = 1,
  foldType = c("random", "consecutive", "interleaved"),
  grouping = NULL,
  folds = NULL,
  seed = NULL,
  ...
)

cvLts(
  object,
  cost = rtmspe,
  K = 5,
  R = 1,
  foldType = c("random", "consecutive", "interleaved"),
```

```
  grouping = NULL,
  folds = NULL,
  fit = c("reweighted", "raw", "both"),
  seed = NULL,
  ...
)
```

Arguments

object	an object returned from a model fitting function. Methods are implemented for objects of class "lm" computed with lm, objects of class "lmrob" computed with lmrob, and object of class "lts" computed with ltsReg.
...	additional arguments to be passed to the prediction loss function cost.
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).
K	an integer giving the number of folds into which the data should be split (the default is five). Keep in mind that this should be chosen such that all folds are of approximately equal size. 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.
foldType	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.
folds	an object of class "cvFolds" giving the folds of the data for cross-validation (as returned by cvFolds). If supplied, this is preferred over the arguments for generating cross-validation folds.
seed	optional initial seed for the random number generator (see .Random.seed).
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.

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. 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 the predict method of the fitted model. 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 cross-validation, this process is replicated and the estimated prediction errors from all replications as well as their average are included in the returned object.

Value

An object of class `"cv"` 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.
`cv`	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 the case of repeated cross-validation, 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 for repeated cross-validation.
`seed`	the seed of the random number generator before cross-validation was performed.
`call`	the matched function call.

Note

The `repCV` methods are simple wrapper functions that extract the data from the fitted model and call `cvFit` to perform cross-validation. In addition, `cvLm`, `cvLmrob` and `cvLts` are aliases for the respective methods.

Author(s)

Andreas Alfons

See Also

`cvFit`, `cvFolds`, `cost`, `lm`, `lmrob`, `ltsReg`

Examples

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

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

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

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

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

`subset.cv` *Subsetting cross-validation results*

Description

Extract subsets of results from (repeated) K-fold cross-validation.

Usage

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

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

Arguments

`x`	an object inheriting from class `"cv"` or `"cvSelect"` that contains cross-validation results.
`select`	a character, integer or logical vector indicating the columns of cross-validation results to be extracted.
`...`	currently ignored.
`subset`	a character, integer or logical vector indicating the subset of models for which to keep the cross-validation results.

Value

An object similar to `x` containing just the selected results.

Author(s)

Andreas Alfons

See Also

cvFit, cvSelect, cvTuning, subset

Examples

```
library("robustbase")
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
fitLts50 <- ltsReg(Y ~ ., data = coleman, alpha = 0.5)
cvFitLts50 <- cvLts(fitLts50, cost = rtmspe, folds = folds,
    fit = "both", trim = 0.1)

# 75% subsets
fitLts75 <- ltsReg(Y ~ ., data = coleman, alpha = 0.75)
cvFitLts75 <- cvLts(fitLts75, cost = rtmspe, folds = folds,
    fit = "both", trim = 0.1)

# combine results into one object
cvFitsLts <- cvSelect("0.5" = cvFitLts50, "0.75" = cvFitLts75)
cvFitsLts

# extract reweighted LTS results with 50% subsets
subset(cvFitLts50, select = "reweighted")
subset(cvFitsLts, subset = c(TRUE, FALSE), select = "reweighted")
```

---

summary.cv *Summarize cross-validation results*

Description

Produce a summary of results from (repeated) K-fold cross-validation.

Usage

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

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

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

Arguments

`object`	an object inheriting from class `"cv"` or `"cvSelect"` that contains cross-validation results (note that the latter includes objects of class `"cvTuning"`).
`...`	currently ignored.

Value

An object of class `"summary.cv"`, `"summary.cvSelect"` or `"summary.cvTuning"`, depending on the class of `object`.

Author(s)

Andreas Alfons

See Also

`cvFit`, `cvSelect`, `cvTuning`, `summary`

Examples

```
library("robustbase")
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
fitLts50 <- ltsReg(Y ~ ., data = coleman, alpha = 0.5)
cvFitLts50 <- cvLts(fitLts50, cost = rtmspe, folds = folds,
    fit = "both", trim = 0.1)

# 75% subsets
fitLts75 <- ltsReg(Y ~ ., data = coleman, alpha = 0.75)
cvFitLts75 <- cvLts(fitLts75, cost = rtmspe, folds = folds,
    fit = "both", trim = 0.1)

# combine results into one object
cvFitsLts <- cvSelect("0.5" = cvFitLts50, "0.75" = cvFitLts75)
cvFitsLts

# summary of the results with the 50% subsets
summary(cvFitLts50)
# summary of the combined results
```

```
summary(cvFitsLts)
```

`xyplot.cv` *X-Y plots of cross-validation results*

Description

Plot the (average) results from (repeated) K-fold cross-validation on the y-axis against the respective models on the x-axis.

Usage

```
## S3 method for class 'cv'
xyplot(x, data, select = NULL, seFactor = NA, ...)

## S3 method for class 'cvSelect'
xyplot(x, data, subset = NULL, select = NULL, seFactor = x$seFactor, ...)

## S3 method for class 'cvTuning'
xyplot(x, data, subset = NULL, select = NULL, seFactor = x$seFactor, ...)
```

Arguments

`x`	an object inheriting from class `"cvSelect"` that contains cross-validation results (note that this includes objects of class `"cvTuning"`).
`data`	currently ignored.
`select`	a character, integer or logical vector indicating the columns of cross-validation results to be plotted.
`seFactor`	a numeric value giving the multiplication factor of the standard error for displaying error bars. Error bars can be suppressed by setting this to `NA`.
`...`	additional arguments to be passed to the `"formula"` method of `xyplot`.
`subset`	a character, integer or logical vector indicating the subset of models for which to plot the cross-validation results.

Details

For objects with multiple columns of repeated cross-validation results, conditional plots are produced.

In most situations, the default behavior is to represent the cross-validation results for each model by a vertical line segment (i.e., to call the default method of `xyplot` with `type = "h"`). However, the behavior is different for objects of class `"cvTuning"` with only one numeric tuning parameter. In that situation, the cross-validation results are plotted against the values of the tuning parameter as a connected line (i.e., by using `type = "b"`).

The default behavior can of course be overridden by supplying the `type` argument (a full list of accepted values can be found in the help file of `panel.xyplot`).

Value

An object of class "trellis" is returned invisibly. The update method can be used to update components of the object and the print method (usually called by default) will plot it on an appropriate plotting device.

Author(s)

Andreas Alfons

See Also

cvFit, cvSelect, cvTuning, plot, dotplot, bwplot, densityplot

Examples

```
library("robustbase")
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)
cvFitLm <- cvLm(fitLm, cost = rtmspe,
    folds = folds, trim = 0.1)

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

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

# combine and plot results
cvFits <- cvSelect(LS = cvFitLm, MM = cvFitLmrob, LTS = cvFitLts)
cvFits
xyplot(cvFits)
```

Index