Package: nestedcv (via r-universe)

November 5, 2024

Title Nested Cross-Validation with 'glmnet' and 'caret'

Version 0.7.12

Maintainer Myles Lewis <`myles.lewis@qmul.ac.uk`>

BugReports `https://github.com/myles-lewis/nestedcv/issues`

URL `https://github.com/myles-lewis/nestedcv`

Description Implements nested k*l-fold cross-validation for lasso and elastic-net regularised linear models via the 'glmnet' package and other machine learning models via the 'caret' package <doi:10.1093/bioadv/vbad048>. Cross-validation of 'glmnet' alpha mixing parameter and embedded fast filter functions for feature selection are provided. Described as double cross-validation by Stone (1977) <doi:10.1111/j.2517-6161.1977.tb01603.x>. Also implemented is a method using outer CV to measure unbiased model performance metrics when fitting Bayesian linear and logistic regression shrinkage models using the horseshoe prior over parameters to encourage a sparse model as described by Piironen & Vehtari (2017) <doi:10.1214/17-EJS1337SI>.

Language en-gb

License MIT + file LICENSE

Encoding UTF-8

Imports caret, data.table, doParallel, foreach, ggplot2, glmnet, matrixStats, matrixTests, methods, parallel, pROC, Rfast, RhpcBLASctl, rlang, ROCR

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

Suggests Boruta, CORElearn, fastshap (>= 0.1.0), gbm, ggbeeswarm, ggpubr, hsstan, mda, mlbench, pbapply, pls, randomForest, ranger, RcppEigen, rmarkdown, knitr, SuperLearner

VignetteBuilder knitr

Config/pak/sysreqs make libicu-dev

Repository https://myles-lewis.r-universe.dev

RemoteUrl https://github.com/myles-lewis/nestedcv

RemoteRef HEAD

RemoteSha 9c5da9bbe29062474b4a24cca653b388151c500f

Contents

`barplot_var_stability` *Barplot variable stability*

Description

Produces a ggplot2 plot of stability (as SEM) of variable importance across models trained and tested across outer CV folds. Optionally overlays directionality for binary response or regression outcomes.

Usage

```
barplot_var_stability(
  x,
  final = TRUE,
  top = NULL,
  direction = 0,
  dir_labels = NULL,
  scheme = c("royalblue", "red"),
  breaks = NULL,
  percent = TRUE,
  level = 1,
  sort = TRUE
)
```

Arguments

`x`	a `nestcv.glmnet` or `nestcv.train` fitted object
`final`	Logical whether to restrict variables to only those which ended up in the final fitted model or to include all variables selected across all outer folds.
`top`	Limits number of variables plotted. Set to `NULL` to plot all variables.
`direction`	Integer controlling plotting of directionality for binary or regression models. `0` means no directionality is shown, `1` means directionality is overlaid as a colour, `2` means directionality is reflected in the sign of variable importance. Not available for multiclass caret models.
`dir_labels`	Character vector for controlling the legend when `direction = 1`
`scheme`	Vector of 2 colours for directionality when `direction = 1`
`breaks`	Vector of continuous breaks for legend colour/size
`percent`	Logical for `nestcv.glmnet` objects only, whether to scale coefficients to percentage of the largest coefficient in each model. If set to `FALSE`, model coefficients are shown and `direction` is ignored.
`level`	For multinomial `nestcv.glmnet` models only, either an integer specifying which level of outcome is being examined, or the level can be specified as a character value.
`sort`	Logical whether to sort by mean variable importance. Passed to `var_stability()`.

Value

A ggplot2 plot

See Also

`var_stability()`

`boot_filter` *Bootstrap for filter functions*

Description

Randomly samples predictors and averages the ranking to give an ensemble measure of predictor variable importance.

Usage

```
boot_filter(y, x, filterFUN, B = 50, nfilter = NULL, type = "index", ...)
```

Arguments

`y`	Response vector
`x`	Matrix of predictors
`filterFUN`	Filter function, e.g. `ttest_filter()`.
`B`	Number of times to bootstrap
`nfilter`	Number of predictors to return
`type`	Type of vector returned. Default "index" returns indices, "full" returns full output.
`...`	Optional arguments passed to the function specified by `filterFUN`

Value

Integer vector of indices of filtered parameters (`type = "index"`) or if `type = "full"` a matrix of rankings from each bootstrap is returned.

See Also

`boot_ttest()`

`boot_ttest` *Bootstrap univariate filters*

Description

Randomly samples predictors and averages the ranking from filtering functions including `ttest_filter()`, `wilcoxon_filter()`, `anova_filter()`, `correl_filter()` and `lm_filter()` to give an ensemble measure of best predictors by repeated random sampling subjected to a statistical test.

Usage

```
boot_ttest(y, x, B = 50, ...)

boot_wilcoxon(y, x, B = 50, ...)

boot_anova(y, x, B = 50, ...)

boot_correl(y, x, B = 50, ...)

boot_lm(y, x, B = 50, ...)
```

Arguments

`y`	Response vector
`x`	Matrix of predictors
`B`	Number of times to bootstrap
`...`	Optional arguments passed to the filter function

Value

Integer vector of indices of filtered parameters (`type = "index"`), or if `type = "full"`, a matrix of rankings from each bootstrap is returned.

See Also

`ttest_filter()`, `wilcoxon_filter()`, `anova_filter()`, `correl_filter()`, `lm_filter()` and `boot_filter()`

`boruta_filter` *Boruta filter*

Description

Filter using Boruta algorithm.

Usage

```
boruta_filter(
  y,
  x,
  select = c("Confirmed", "Tentative"),
  type = c("index", "names", "full"),
  ...
)
```

Arguments

`y`	Response vector
`x`	Matrix of predictors
`select`	Which type of features to retain. Options include "Confirmed" and/or "Tentative".
`type`	Type of vector returned. Default "index" returns indices, "names" returns predictor names, "full" returns a named vector of variable importance.
`...`	Other arguments passed to `Boruta::Boruta()`

Details

Boruta works differently from other filters in that it does not rank variables by variable importance, but tries to determine relevant features and divides features into Rejected, Tentative or Confirmed.

Value

Integer vector of indices of filtered parameters (type = "index") or character vector of names (type = "names") of filtered parameters. If `type` is `"full"` full output from `Boruta` is returned.

`boxplot_expression` *Boxplot expression levels of model predictors*

Description

Boxplots to show range of model predictors to identify exceptional predictors with excessively low or high values.

Usage

```
boxplot_expression(x, scheme = NULL, palette = "Dark 3", ...)
```

Arguments

`x`	a "nestedcv" object
`scheme`	colour scheme
`palette`	palette name (one of `hcl.pals()`) which is passed to hcl.colors
`...`	other arguments passed to boxplot.

Value

No return value

Author(s)

Myles Lewis

See Also

nestcv.glmnet

`class_balance` *Check class balance in training folds*

Description

Check class balance in training folds

Usage

```
class_balance(object)

## Default S3 method:
class_balance(object)

## S3 method for class 'nestcv.train'
class_balance(object)
```

Arguments

`object`	Object of class `nestedcv.glmnet`, `nestcv.train` or `outercv`

Value

Invisibly a table of the response classes in the training folds

---

`coef.cva.glmnet` *Extract coefficients from a cva.glmnet object*

Description

Extracts model coefficients from a fitted `cva.glmnet()` object.

Usage

```
## S3 method for class 'cva.glmnet'
coef(object, ...)
```

Arguments

`object`	Fitted `cva.glmnet` object.
`...`	Other arguments passed to `coef.glmnet()` e.g. `s` the value of lambda at which coefficients are required.

Value

Sparse matrix containing coefficients from a `cv.glmnet` model

`coef.nestcv.glmnet` *Extract coefficients from nestcv.glmnet object*

Description

Extracts coefficients from the final fit of a `"nestcv.glmnet"` object.

Usage

```
## S3 method for class 'nestcv.glmnet'
coef(object, s = object$final_param["lambda"], ...)
```

Arguments

`object`	Object of class `"nestcv.glmnet"`
`s`	Value of penalty parameter lambda. Default is the mean of lambda values selected across each outer fold.
`...`	Other arguments passed to `glmnet::coef.glmnet()`

Value

Vector or list of coefficients ordered with the intercept first, followed by highest absolute value to lowest.

`collinear` *Filter to reduce collinearity in predictors*

Description

This function identifies predictors with r^2 above a given cut-off and produces an index of predictors to be removed. The function takes a matrix or data.frame of predictors, and the columns need to be ordered in terms of importance - first column of any pair that are correlated is retained and subsequent columns which correlate above the cut-off are flagged for removal.

Usage

```
collinear(x, rsq_cutoff = 0.9, rsq_method = "pearson", verbose = FALSE)
```

Arguments

`x`	A matrix or data.frame of values. The order of columns is used to determine which columns to retain, so the columns in `x` should be sorted with the most important columns first.
`rsq_cutoff`	Value of cut-off for r-squared
`rsq_method`	character string indicating which correlation coefficient is to be computed. One of "pearson" (default), "kendall", or "spearman". See `cor()`.
`verbose`	Boolean whether to print details

Value

Integer vector of the indices of columns in `x` to remove due to collinearity

---

`combo_filter` *Combo filter*

Description

Filter combining univariate (t-test or anova) filtering and reliefF filtering in equal measure.

Usage

```
combo_filter(y, x, nfilter, type = c("index", "names", "full"), ...)
```

Arguments

`y`	Response vector
`x`	Matrix or dataframe of predictors
`nfilter`	Number of predictors to return, using 1/2 from `ttest_filter` or `anova_filter` and 1/2 from `relieff_filter`. Since `unique` is applied, the final number returned may be less than `nfilter`.
`type`	Type of vector returned. Default "index" returns indices, "names" returns predictor names, "full" returns full output.
`...`	Optional arguments passed via relieff_filter to CORElearn::attrEval

Value

Integer vector of indices of filtered parameters (type = "index") or character vector of names (type = "names") of filtered parameters. If `type` is `"full"` a list containing full outputs from either ttest_filter or anova_filter and relieff_filter is returned.

`correls2` *Correlation between a vector and a matrix*

Description

Fast Pearson/Spearman correlation where `y` is vector, `x` is matrix, adapted from stats::cor.test.

Usage

```
correls2(y, x, method = "pearson", use = "complete.obs")
```

Arguments

`y`	Numerical vector
`x`	Matrix
`method`	Type of correlation, either "pearson" or "spearman".
`use`	Optional character string giving a method for computing covariances in the presence of missing values. See cor

Details

For speed, p-values for Spearman's test are computed by asymptotic t approximation, equivalent to cor.test with `exact = FALSE`.

Value

Matrix with columns containing the correlation statistic, either Pearson r or Spearman rho, and p-values for each column of `x` correlated against vector `y`

cva.glmnet — *Cross-validation of alpha for glmnet*

Description

Performs k-fold cross-validation for glmnet, including alpha mixing parameter.

Usage

```
cva.glmnet(x, y, nfolds = 10, alphaSet = seq(0.1, 1, 0.1), foldid = NULL, ...)
```

Arguments

`x`	Matrix of predictors
`y`	Response vector
`nfolds`	Number of folds (default 10)
`alphaSet`	Sequence of alpha values to cross-validate
`foldid`	Optional vector of values between 1 and `nfolds` identifying what fold each observation is in.
`...`	Other arguments passed to glmnet::cv.glmnet

Value

Object of S3 class "cva.glmnet", which is a list of the cv.glmnet objects for each value of alpha and `alphaSet`.

`fits`	List of fitted glmnet::cv.glmnet objects
`alphaSet`	Sequence of alpha values used
`alpha_cvm`	The mean cross-validated error - a vector of length `length(alphaSet)`.
`best_alpha`	Value of alpha giving lowest `alpha_cvm`.
`which_alpha`	Index of `alphaSet` with lowest `alpha_cvm`

Author(s)

Myles Lewis

See Also

glmnet::cv.glmnet, glmnet::glmnet

`cv_coef` *Coefficients from outer CV glmnet models*

Description

Extracts coefficients from outer CV glmnet models from a `nestcv.glmnet` fitted object.

Usage

```
cv_coef(x, level = 1)
```

Arguments

`x`	a `nestcv.glmnet` fitted object
`level`	For multinomial models only, either an integer specifying which level of outcome is being examined, or the level can be specified as a character value

Value

matrix of coefficients from outer CV glmnet models plus the final glmnet model. Coefficients for variables which are not present in a particular outer CV fold model are set to 0.

See Also

`cv_varImp()`

`cv_varImp` *Extract variable importance from outer CV caret models*

Description

Extracts variable importance or coefficients from outer CV glmnet models from a `nestcv.train` fitted object.

Usage

```
cv_varImp(x)
```

Arguments

`x`	a `nestcv.train` fitted object

Details

Note that `caret::varImp()` may require the model package to be fully loaded in order to function. During the fitting process `caret` often only loads the package by namespace.

Value

matrix of variable importance from outer CV fold caret models as well as the final model. Variable importance for variables which are not present in a particular outer CV fold model is set to 0.

See Also

`cv_coef()`

`glmnet_coefs` *glmnet coefficients*

Description

Convenience function for retrieving coefficients from a glmnet::cv.glmnet model at a specified lambda. Sparsity is removed and non-intercept coefficients are ranked by absolute value.

Usage

```
glmnet_coefs(fit, s, ...)
```

Arguments

`fit`	A glmnet::cv.glmnet fitted model object.
`s`	Value of lambda. See glmnet::coef.glmnet and glmnet::predict.cv.glmnet
`...`	Other arguments passed to glmnet::coef.glmnet

Value

Vector or list of coefficients ordered with the intercept first, followed by highest absolute value to lowest.

`glmnet_filter` *glmnet filter*

Description

Filter using sparsity of elastic net regression using glmnet to calculate variable importance.

Usage

```
glmnet_filter(
  y,
  x,
  family = NULL,
  force_vars = NULL,
  nfilter = NULL,
  method = c("mean", "nonzero"),
  type = c("index", "names", "full"),
  ...
)
```

Arguments

`y`	Response vector
`x`	Matrix of predictors
`family`	Either a character string representing one of the built-in families, or else a `glm()` family object. See `glmnet::glmnet()`. If not specified, the function tries to set this automatically to one of either "gaussian", "binomial" or "multinomial".
`force_vars`	Vector of column names `x` which have no shrinkage and are always included in the model.
`nfilter`	Number of predictors to return
`method`	String indicating method of determining variable importance. "mean" (the default) uses the mean absolute coefficients across the range of lambdas; "nonzero" counts the number of times variables are retained in the model across all values of lambda.
`type`	Type of vector returned. Default "index" returns indices, "names" returns predictor names, "full" returns full output.
`...`	Other arguments passed to glmnet::glmnet

Details

The glmnet elastic net mixing parameter alpha can be varied to include a larger number of predictors. Default alpha = 1 is pure LASSO, resulting in greatest sparsity, while alpha = 0 is pure ridge regression, retaining all predictors in the regression model. Note, the `family` argument is commonly needed, see glmnet::glmnet.

Value

Integer vector of indices of filtered parameters (type = "index") or character vector of names (type = "names") of filtered parameters. If `type` is `"full"` a named vector of variable importance is returned.

See Also

glmnet::glmnet

`innercv_preds` *Inner CV predictions*

Description

Obtain predictions on held-out test inner CV folds

Usage

```
innercv_preds(x)

## S3 method for class 'nestcv.glmnet'
innercv_preds(x)

## S3 method for class 'nestcv.train'
innercv_preds(x)
```

Arguments

`x`	a `nestcv.glmnet` or `nestcv.train` fitted object

Value

Dataframe with columns `testy` and `predy`, and for binomial and multinomial models additional columns containing probabilities or log likelihood values.

`innercv_roc` *Build ROC curve from left-out folds from inner CV*

Description

Build ROC (receiver operating characteristic) curve from left-out folds from inner CV. Object can be plotted using `plot()` or passed to functions `pROC::auc()` etc.

Usage

```
innercv_roc(x, direction = "<", ...)
```

Arguments

`x`	a `nestcv.glmnet` or `nestcv.train` fitted object
`direction`	Set ROC directionality pROC::roc
`...`	Other arguments passed to pROC::roc

Value

"roc" object, see pROC::roc

Examples

```
## Example binary classification problem with P >> n
x <- matrix(rnorm(150 * 2e+04), 150, 2e+04)  # predictors
y <- factor(rbinom(150, 1, 0.5))  # binary response

## Partition data into 2/3 training set, 1/3 test set
trainSet <- caret::createDataPartition(y, p = 0.66, list = FALSE)

## t-test filter using whole dataset
filt <- ttest_filter(y, x, nfilter = 100)
filx <- x[, filt]

## Train glmnet on training set only using filtered predictor matrix
library(glmnet)
fit <- cv.glmnet(filx[trainSet, ], y[trainSet], family = "binomial")
plot(fit)

## Predict response on test partition
predy <- predict(fit, newx = filx[-trainSet, ], s = "lambda.min", type = "class")
predy <- as.vector(predy)
predyp <- predict(fit, newx = filx[-trainSet, ], s = "lambda.min", type = "response")
predyp <- as.vector(predyp)
output <- data.frame(testy = y[-trainSet], predy = predy, predyp = predyp)

## Results on test partition
## shows bias since univariate filtering was applied to whole dataset
predSummary(output)

## Nested CV
fit2 <- nestcv.glmnet(y, x, family = "binomial", alphaSet = 1,
                      filterFUN = ttest_filter,
                      filter_options = list(nfilter = 100),
                      n_outer_folds = 3)
summary(fit2)

## ROC plots
library(pROC)
testroc <- roc(output$testy, output$predyp, direction = "<")
inroc <- innercv_roc(fit2)
plot(fit2$roc)
lines(inroc, col = 'blue')
lines(testroc, col = 'red')
legend('bottomright', legend = c("Nested CV", "Left-out inner CV folds",
                                 "Test partition, non-nested filtering"),
       col = c("black", "blue", "red"), lty = 1, lwd = 2, bty = "n")
```

innercv_summary — *Summarise performance on inner CV test folds*

Description

Calculates performance metrics on inner CV held-out test folds: confusion matrix, accuracy and balanced accuracy for classification; ROC AUC for binary classification; RMSE, R^2 and mean absolute error (MAE) for regression.

Usage

```
innercv_summary(x)
```

Arguments

x	a `nestcv.glmnet` or `nestcv.train` object

Value

Returns performance metrics from outer training folds, see predSummary.

See Also

predSummary

Examples

```
data(iris)
x <- iris[, 1:4]
y <- iris[, 5]

fit <- nestcv.glmnet(y, x,
                     family = "multinomial",
                     alpha = 1,
                     n_outer_folds = 3)
summary(fit)
innercv_summary(fit)
```

lines.prc *Add precision-recall curve to a plot*

Description

Adds a precision-recall curve to a base graphics plot. It accepts an S3 object of class 'prc', see `prc()`.

Usage

```
## S3 method for class 'prc'
lines(x, ...)
```

Arguments

`x`	An object of class 'prc'
`...`	Optional graphical arguments passed to `lines()`

Value

No return value

See Also

`prc()` `plot.prc()`

lm_filter *Linear model filter*

Description

Linear models are fitted on each predictor, with inclusion of variable names listed in `force_vars` in the model. Predictors are ranked by Akaike information criteria (AIC) value, or can be filtered by the p-value on the estimate of the coefficient for that predictor in its model.

Usage

```
lm_filter(
  y,
  x,
  force_vars = NULL,
  nfilter = NULL,
  p_cutoff = 0.05,
  rsq_cutoff = NULL,
  rsq_method = "pearson",
  type = c("index", "names", "full"),
  keep_factors = TRUE,
  method = 0L,
  mc.cores = 1
)
```

Arguments

`y`	Numeric or integer response vector
`x`	Matrix of predictors. If `x` is a data.frame it will be turned into a matrix. But note that factors will be reduced to numeric values, but a full design matrix is not generated, so if factors have 3 or more levels, it is recommended to convert `x` into a design (model) matrix first.
`force_vars`	Vector of column names `x` which are incorporated into the linear model.
`nfilter`	Number of predictors to return. If `NULL` all predictors with p-values < `p_cutoff` are returned.
`p_cutoff`	p-value cut-off. P-values are calculated by t-statistic on the estimated coefficient for the predictor being tested.
`rsq_cutoff`	r^2 cutoff for removing predictors due to collinearity. Default `NULL` means no collinearity filtering. Predictors are ranked based on AIC from a linear model. If 2 or more predictors are collinear, the first ranked predictor by AIC is retained, while the other collinear predictors are removed. See `collinear()`.
`rsq_method`	character string indicating which correlation coefficient is to be computed. One of "pearson" (default), "kendall", or "spearman". See `collinear()`.
`type`	Type of vector returned. Default "index" returns indices, "names" returns predictor names, "full" returns a matrix of p values.
`keep_factors`	Logical affecting factors with 3 or more levels. Dataframes are coerced to a matrix using data.matrix. Binary factors are converted to numeric values 0/1 and analysed as such. If `keep_factors` is `TRUE` (the default), factors with 3 or more levels are not filtered and are retained. If `keep_factors` is `FALSE`, they are removed.
`method`	Integer determining linear model method. See `RcppEigen::fastLmPure()`
`mc.cores`	Number of cores for parallelisation using `parallel::mclapply()`.

Details

This filter is based on the model `y` ~ `xvar` + `force_vars` where `y` is the response vector, `xvar` are variables in columns taken sequentially from `x` and `force_vars` are optional covariates extracted from `x`. It uses `RcppEigen::fastLmPure()` with `method` = `0` as default since it is rank-revealing. `method` = `3` is significantly faster but can give errors in estimation of p-value with variables of zero variance. The algorithm attempts to detect these and set their stats to `NA`. `NA` in `x` are not tolerated.

Parallelisation is available via `mclapply()`. This is provided mainly for the use case of the filter being used as standalone. Nesting parallelisation inside of parallelised `nestcv.glmnet()` or `nestcv.train()` loops is not recommended.

Value

Integer vector of indices of filtered parameters (`type = "index"`) or character vector of names (`type = "names"`) of filtered parameters in order of linear model AIC. Any variables in `force_vars` which are incorporated into all models are listed first. If `type = "full"` a matrix of AIC value, sigma (residual standard error, see summary.lm), coefficient, t-statistic and p-value for each tested predictor is returned.

`mcc` *Matthews correlation coefficient*

Description

Calculates Matthews correlation coefficient (MCC) which is in essence a correlation coefficient between the observed and predicted binary classifications. It has also been generalised to multi-class classification.

Usage

```
mcc(cm)

mcc_multi(cm)
```

Arguments

`cm`	A contingency table or matrix of predicted vs observed classes with reference classes in columns and predicted classes in rows.

Details

Use `mcc()` for 2x2 tables (binary classification). `mcc_multi()` is for multi-class classification with k x k tables and is calculated using Gorodkin's method.

Value

Returns a value between -1 and +1. A coefficient of +1 represents a perfect prediction, 0 no better than random prediction and -1 indicates total disagreement between prediction and observation.

References

Gorodkin, J. (2004). *Comparing two K-category assignments by a K-category correlation coefficient*. Computational Biology and Chemistry. 28 (5): 367–374.

metrics *Model performance metrics*

Description

Returns model metrics from nestedcv models. Extended metrics including

Usage

```
metrics(object, extra = FALSE, innerCV = FALSE, positive = 2)
```

Arguments

`object`	A 'nestcv.glmnet', 'nestcv.train', 'nestcv.SuperLearner' or 'outercv' object.
`extra`	Logical whether additional performance metrics are gathered for classification models: area under precision recall curve (PR.AUC, binary classification only), Cohen's kappa, F1 score, Matthews correlation coefficient (MCC).
`innerCV`	Whether to calculate metrics for inner CV folds. Only available for 'nestcv.glmnet' and 'nestcv.train' objects.
`positive`	For binary classification, either an integer 1 or 2 for the level of response factor considered to be 'positive' or 'relevant', or a character value for that factor. This affects the F1 score. See `caret::confusionMatrix()`.

Details

Area under precision recall curve is estimated by trapezoidal estimation using `MLmetrics::PRAUC()`.

For multi-class classification models, Matthews correlation coefficient is calculated using Gorodkin's method. Multi-class F1 score (macro F1) is calculated as the arithmetic mean of the class-wise F1 scores.

Value

A named numeric vector of performance metrics.

References

Gorodkin, J. (2004). *Comparing two K-category assignments by a K-category correlation coefficient*. Computational Biology and Chemistry. 28 (5): 367–374.

See Also

`mcc()`

`model.hsstan` *hsstan model for cross-validation*

Description

This function applies a cross-validation (CV) procedure for training Bayesian models with hierarchical shrinkage priors using the `hsstan` package. The function allows the option of embedded filtering of predictors for feature selection within the CV loop. Within each training fold, an optional filtering of predictors is performed, followed by fitting of an `hsstsan` model. Predictions on the testing folds are brought back together and error estimation/ accuracy determined. The default is 10-fold CV. The function is implemented within the `nestedcv` package. The `hsstan` models do not require tuning of meta-parameters and therefore only a single CV procedure is needed to evaluate performance. This is implemented using the `outer` CV procedure in the `nestedcv` package. Supports binary outcome (logistic regression) or continuous outcome. Multinomial models are currently not supported.

Usage

```
model.hsstan(y, x, unpenalized = NULL, ...)
```

Arguments

`y`	Response vector. For classification this should be a factor.
`x`	Matrix of predictors
`unpenalized`	Vector of column names `x` which are always retained into the model (i.e. not penalized). Default `NULL` means the parameters for all predictors will be drawn from a hierarchical prior distribution, i.e. will be penalized. Note: if filtering of predictors is specified, then the vector of `unpenalized` predictors should also be passed to the filter function using the `filter_options$force_vars` argument. Filters currently implementing this option are the `partial_ttest_filter` for binary outcomes and the `lm_filter` for continuous outcomes.
`...`	Optional arguments passed to `hsstan`

Details

Caution should be used when setting the number of cores available for parallelisation. The default setting in `hsstan` is to use 4 cores to parallelise the Markov chains of the Bayesian inference procedure. This can be switched off either by adding argument `cores = 1` (passed on to `rstan`) or setting `options(mc.cores = 1)`.

Argument `cv.cores` in `outercv()` controls parallelisation over the outer CV folds. On unix/mac setting `cv.cores` to >1 will induce nested parallelisation which will generate an error, unless parallelisation of the chains is disabled using `cores = 1` or setting `options(mc.cores = 1)`.

Nested parallelisation is feasible if `cv.cores` is >1 and `multicore_fork = FALSE` is set as this uses cluster based parallelisation instead. Beware that large numbers of processes will be spawned. If we are performing 10-fold cross-validation with 4 chains and set `cv.cores = 10` then 40 processes will be invoked simultaneously.

Value

An object of class `hsstan`

Author(s)

Athina Spiliopoulou

See Also

`outercv()` `hsstan::hsstan()`

Examples

```
# Cross-validation is used to apply univariate filtering of predictors.
# only one CV split is needed (outercv) as the Bayesian model does not
# require learning of meta-parameters.

# control number of cores used for parallelisation over chains
oldopt <- options(mc.cores = 2)

# load iris dataset and simulate a continuous outcome
data(iris)
dt <- iris[, 1:4]
colnames(dt) <- c("marker1", "marker2", "marker3", "marker4")
dt <- as.data.frame(apply(dt, 2, scale))
dt$outcome.cont <- -3 + 0.5 * dt$marker1 + 2 * dt$marker2 + rnorm(nrow(dt), 0, 2)

library(hsstan)
# unpenalised covariates: always retain in the prediction model
uvars <- "marker1"
# penalised covariates: coefficients are drawn from hierarchical shrinkage
# prior
pvars <- c("marker2", "marker3", "marker4") # penalised covariates
# run cross-validation with univariate filter and hsstan
# dummy sampling for fast execution of example
# recommend 4 chains, warmup 1000, iter 2000 in practice
res.cv.hsstan <- outercv(y = dt$outcome.cont, x = dt[, c(uvars, pvars)],
                         model = "model.hsstan",
                         filterFUN = lm_filter,
                         filter_options = list(force_vars = uvars,
                                               nfilter = 2,
                                               p_cutoff = NULL,
                                               rsq_cutoff = 0.9),
                         n_outer_folds = 3,
                         chains = 2,
                         cv.cores = 1,
                         unpenalized = uvars, warmup = 100, iter = 200)
# view prediction performance based on testing folds
res.cv.hsstan$summary
# view coefficients for the final model
res.cv.hsstan$final_fit
# view covariates selected by the univariate filter
```

```
res.cv.hsstan$final_vars

# use hsstan package to examine the Bayesian model
sampler.stats(res.cv.hsstan$final_fit)
print(projsel(res.cv.hsstan$final_fit), digits = 4)  # adding marker2
options(oldopt)  # reset configuation

# Here adding `marker2` improves the model fit: substantial decrease of
# KL-divergence from the full model to the submodel. Adding `marker3` does
# not improve the model fit: no decrease of KL-divergence from the full model
# to the submodel.
```

`nestcv.glmnet` *Nested cross-validation with glmnet*

Description

This function enables nested cross-validation (CV) with glmnet including tuning of elastic net alpha parameter. The function also allows the option of embedded filtering of predictors for feature selection nested within the outer loop of CV. Predictions on the outer test folds are brought back together and error estimation/ accuracy determined. The default is 10x10 nested CV.

Usage

```
nestcv.glmnet(
  y,
  x,
  family = c("gaussian", "binomial", "poisson", "multinomial", "cox", "mgaussian"),
  filterFUN = NULL,
  filter_options = NULL,
  balance = NULL,
  balance_options = NULL,
  modifyX = NULL,
  modifyX_useY = FALSE,
  modifyX_options = NULL,
  outer_method = c("cv", "LOOCV"),
  n_outer_folds = 10,
  n_inner_folds = 10,
  outer_folds = NULL,
  pass_outer_folds = FALSE,
  alphaSet = seq(0.1, 1, 0.1),
  min_1se = 0,
  keep = TRUE,
  outer_train_predict = FALSE,
  weights = NULL,
  penalty.factor = rep(1, ncol(x)),
  cv.cores = 1,
  finalCV = TRUE,
  na.option = "omit",
  verbose = FALSE,
  ...
)
```

Arguments

`y`	Response vector or matrix. Matrix is only used for `family = 'mgaussian'` or `'cox'`.
`x`	Matrix of predictors. Dataframes will be coerced to a matrix as is necessary for glmnet.
`family`	Either a character string representing one of the built-in families, or else a `glm()` family object. Passed to glmnet::cv.glmnet and glmnet::glmnet
`filterFUN`	Filter function, e.g. ttest_filter or relieff_filter. Any function can be provided and is passed `y` and `x`. Must return a numeric vector with indices of filtered predictors.
`filter_options`	List of additional arguments passed to the filter function specified by `filterFUN`.
`balance`	Specifies method for dealing with imbalanced class data. Current options are `"randomsample"` or `"smote"`. See `randomsample()` and `smote()`
`balance_options`	List of additional arguments passed to the balancing function
`modifyX`	Character string specifying the name of a function to modify `x`. This can be an imputation function for replacing missing values, or a more complex function which alters or even adds columns to `x`. The required return value of this function depends on the `modifyX_useY` setting.
`modifyX_useY`	Logical value whether the `x` modifying function makes use of response training data from `y`. If `FALSE` then the `modifyX` function simply needs to return a modified `x` object, which will be coerced to a matrix as required by `glmnet`. If `TRUE` then the `modifyX` function must return a model type object on which `predict()` can be called, so that train and test partitions of `x` can be modified independently.
`modifyX_options`	List of additional arguments passed to the `x` modifying function
`outer_method`	String of either `"cv"` or `"LOOCV"` specifying whether to do k-fold CV or leave one out CV (LOOCV) for the outer folds
`n_outer_folds`	Number of outer CV folds
`n_inner_folds`	Number of inner CV folds
`outer_folds`	Optional list containing indices of test folds for outer CV. If supplied, `n_outer_folds` is ignored.
`pass_outer_folds`	Logical indicating whether the same outer folds are used for fitting of the final model when final CV is applied. Note this can only be applied when `n_outer_folds` and `n_inner_folds` are the same and no balancing is applied.
`alphaSet`	Vector of alphas to be tuned

`min_1se`	Value from 0 to 1 specifying choice of optimal lambda from 0=lambda.min to 1=lambda.1se
`keep`	Logical indicating whether inner CV predictions are retained for calculating left-out inner CV fold accuracy etc. See argument `keep` in glmnet::cv.glmnet.
`outer_train_predict`	Logical whether to save predictions on outer training folds to calculate performance on outer training folds.
`weights`	Weights applied to each sample. Note `weights` and `balance` cannot be used at the same time. Weights are only applied in glmnet and not in filters.
`penalty.factor`	Separate penalty factors can be applied to each coefficient. Can be 0 for some variables, which implies no shrinkage, and that variable is always included in the model. Default is 1 for all variables. See glmnet::glmnet. Note this works separately from filtering. For some `nestedcv` filter functions you might need to set `force_vars` to avoid filtering out features.
`cv.cores`	Number of cores for parallel processing of the outer loops. NOTE: this uses `parallel::mclapply` on unix/mac and `parallel::parLapply` on windows.
`finalCV`	Logical whether to perform one last round of CV on the whole dataset to determine the final model parameters. If set to `FALSE`, the median of hyperparameters from outer CV folds are used for the final model. Performance metrics are independent of this last step. If set to `NA`, final model fitting is skipped altogether, which gives a useful speed boost if performance metrics are all that is needed.
`na.option`	Character value specifying how `NA`s are dealt with. `"omit"` (the default) is equivalent to `na.action = na.omit`. `"omitcol"` removes cases if there are `NA` in 'y', but columns (predictors) containing `NA` are removed from 'x' to preserve cases. Any other value means that `NA` are ignored (a message is given).
`verbose`	Logical whether to print messages and show progress
`...`	Optional arguments passed to glmnet::cv.glmnet

Details

glmnet does not tolerate missing values, so `na.option = "omit"` is the default.

Value

An object with S3 class "nestcv.glmnet"

`call`	the matched call
`output`	Predictions on the left-out outer folds
`outer_result`	List object of results from each outer fold containing predictions on left-out outer folds, best lambda, best alpha, fitted glmnet coefficients, list object of inner fitted cv.glmnet and number of filtered predictors at each fold.
`outer_method`	the `outer_method` argument
`n_inner_folds`	number of inner folds
`outer_folds`	List of indices of outer test folds
`dimx`	dimensions of `x`

`xsub`	subset of `x` containing all predictors used in both outer CV folds and the final model
`y`	original response vector
`yfinal`	final response vector (post-balancing)
`final_param`	Final mean best lambda and alpha from each fold
`final_fit`	Final fitted glmnet model
`final_coef`	Final model coefficients and mean expression. Variables with coefficients shrunk to 0 are removed.
`final_vars`	Column names of filtered predictors entering final model. This is useful for subsetting new data for predictions.
`roc`	ROC AUC for binary classification where available.
`summary`	Overall performance summary. Accuracy and balanced accuracy for classification. ROC AUC for binary classification. RMSE for regression.

Author(s)

Myles Lewis

Examples

```
## Example binary classification problem with P >> n
x <- matrix(rnorm(150 * 2e+04), 150, 2e+04)  # predictors
y <- factor(rbinom(150, 1, 0.5))  # binary response

## Partition data into 2/3 training set, 1/3 test set
trainSet <- caret::createDataPartition(y, p = 0.66, list = FALSE)

## t-test filter using whole dataset
filt <- ttest_filter(y, x, nfilter = 100)
filx <- x[, filt]

## Train glmnet on training set only using filtered predictor matrix
library(glmnet)
fit <- cv.glmnet(filx[trainSet, ], y[trainSet], family = "binomial")
plot(fit)

## Predict response on test partition
predy <- predict(fit, newx = filx[-trainSet, ], s = "lambda.min", type = "class")
predy <- as.vector(predy)
predyp <- predict(fit, newx = filx[-trainSet, ], s = "lambda.min", type = "response")
predyp <- as.vector(predyp)
output <- data.frame(testy = y[-trainSet], predy = predy, predyp = predyp)

## Results on test partition
## shows bias since univariate filtering was applied to whole dataset
predSummary(output)

## Nested CV
## n_outer_folds reduced to speed up example
```

```
fit2 <- nestcv.glmnet(y, x, family = "binomial", alphaSet = 1,
                      n_outer_folds = 3,
                      filterFUN = ttest_filter,
                      filter_options = list(nfilter = 100),
                      cv.cores = 2)
summary(fit2)
plot_lambdas(fit2, showLegend = "bottomright")

## ROC plots
library(pROC)
testroc <- roc(output$testy, output$predyp, direction = "<")
inroc <- innercv_roc(fit2)
plot(fit2$roc)
lines(inroc, col = 'blue')
lines(testroc, col = 'red')
legend('bottomright', legend = c("Nested CV", "Left-out inner CV folds",
                                 "Test partition, non-nested filtering"),
       col = c("black", "blue", "red"), lty = 1, lwd = 2, bty = "n")
```

---

nestcv.SuperLearner *Outer cross-validation of SuperLearner model*

Description

Provides a single loop of outer cross-validation to evaluate performance of ensemble models from `SuperLearner` package.

Usage

```
nestcv.SuperLearner(
  y,
  x,
  filterFUN = NULL,
  filter_options = NULL,
  weights = NULL,
  balance = NULL,
  balance_options = NULL,
  modifyX = NULL,
  modifyX_useY = FALSE,
  modifyX_options = NULL,
  outer_method = c("cv", "LOOCV"),
  n_outer_folds = 10,
  outer_folds = NULL,
  cv.cores = 1,
  final = TRUE,
  na.option = "pass",
  verbose = TRUE,
  ...
)
```

Arguments

`y`	Response vector
`x`	Dataframe or matrix of predictors. Matrix will be coerced to dataframe as this is the default for SuperLearner.
`filterFUN`	Filter function, e.g. ttest_filter or relieff_filter. Any function can be provided and is passed `y` and `x`. Ideally returns a numeric vector with indices of filtered predictors. The custom function can return a character vector of names of the filtered predictors, but this will not work with the `penalty.factor` argument in `nestcv.glmnet()`.
`filter_options`	List of additional arguments passed to the filter function specified by `filterFUN`.
`weights`	Weights applied to each sample for models which can use weights. Note `weights` and `balance` cannot be used at the same time. Weights are not applied in filters.
`balance`	Specifies method for dealing with imbalanced class data. Current options are `"randomsample"` or `"smote"`. Not available if `outercv` is called with a formula. See `randomsample()` and `smote()`
`balance_options`	List of additional arguments passed to the balancing function
`modifyX`	Character string specifying the name of a function to modify `x`. This can be an imputation function for replacing missing values, or a more complex function which alters or even adds columns to `x`. The required return value of this function depends on the `modifyX_useY` setting.
`modifyX_useY`	Logical value whether the `x` modifying function makes use of response training data from `y`. If `FALSE` then the `modifyX` function simply needs to return a modified `x` object, which will be coerced to a dataframe as required by `SuperLearner`. If `TRUE` then the `modifyX` function must return a model type object on which `predict()` can be called, so that train and test partitions of `x` can be modified independently.
`modifyX_options`	List of additional arguments passed to the `x` modifying function
`outer_method`	String of either `"cv"` or `"LOOCV"` specifying whether to do k-fold CV or leave one out CV (LOOCV) for the outer folds
`n_outer_folds`	Number of outer CV folds
`outer_folds`	Optional list containing indices of test folds for outer CV. If supplied, `n_outer_folds` is ignored.
`cv.cores`	Number of cores for parallel processing of the outer loops. NOTE: this uses `parallel::mclapply` on unix/mac and `parallel::parLapply` on windows.
`final`	Logical whether to fit final model.
`na.option`	Character value specifying how `NA`s are dealt with. `"omit"` is equivalent to `na.action = na.omit`. `"omitcol"` removes cases if there are `NA` in 'y', but columns (predictors) containing `NA` are removed from 'x' to preserve cases. Any other value means that `NA` are ignored (a message is given).
`verbose`	Logical whether to print messages and show progress
`...`	Additional arguments passed to `SuperLearner::SuperLearner()`

Details

This performs an outer CV on SuperLearner package ensemble models to measure performance, allowing balancing of imbalanced datasets as well as filtering of predictors. SuperLearner prefers dataframes as inputs for the predictors. If `x` is a matrix it will be coerced to a dataframe and variable names adjusted by `make.names()`.

Parallelisation of the outer CV folds is available on linux/mac, but not available on windows. On windows, `snowSuperLearner()` is called instead, so that parallelisation is performed across each call to SuperLearner.

Value

An object with S3 class "nestcv.SuperLearner"

`call`	the matched call
`output`	Predictions on the left-out outer folds
`outer_result`	List object of results from each outer fold containing predictions on left-out outer folds, model result and number of filtered predictors at each fold.
`dimx`	vector of number of observations and number of predictors
`y`	original response vector
`yfinal`	final response vector (post-balancing)
`outer_folds`	List of indices of outer test folds
`final_fit`	Final fitted model on whole data
`final_vars`	Column names of filtered predictors entering final model
`summary_vars`	Summary statistics of filtered predictors
`roc`	ROC AUC for binary classification where available.
`summary`	Overall performance summary. Accuracy and balanced accuracy for classification. ROC AUC for binary classification. RMSE for regression.

Note

Care should be taken with some `SuperLearner` models e.g. `SL.gbm` as some models have multicore enabled by default, which can lead to huge numbers of processes being spawned.

See Also

`SuperLearner::SuperLearner()`

`nestcv.train` *Nested cross-validation for caret*

Description

This function applies nested cross-validation (CV) to training of models using the `caret` package. The function also allows the option of embedded filtering of predictors for feature selection nested within the outer loop of CV. Predictions on the outer test folds are brought back together and error estimation/ accuracy determined. The default is 10x10 nested CV.

Usage

```
nestcv.train(
  y,
  x,
  method = "rf",
  filterFUN = NULL,
  filter_options = NULL,
  weights = NULL,
  balance = NULL,
  balance_options = NULL,
  modifyX = NULL,
  modifyX_useY = FALSE,
  modifyX_options = NULL,
  outer_method = c("cv", "LOOCV"),
  n_outer_folds = 10,
  n_inner_folds = 10,
  outer_folds = NULL,
  inner_folds = NULL,
  pass_outer_folds = FALSE,
  cv.cores = 1,
  multicore_fork = (Sys.info()["sysname"] != "Windows"),
  metric = ifelse(is.factor(y), "logLoss", "RMSE"),
  trControl = NULL,
  tuneGrid = NULL,
  savePredictions = "final",
  outer_train_predict = FALSE,
  finalCV = TRUE,
  na.option = "pass",
  verbose = TRUE,
  ...
)
```

Arguments

`y`	Response vector. For classification this should be a factor.
`x`	Matrix or dataframe of predictors

`method`	String specifying which model to use. See `caret::train()` for details.
`filterFUN`	Filter function, e.g. `ttest_filter()` or `relieff_filter()`. Any function can be provided and is passed `y` and `x`. Ideally returns a numeric vector with indices of filtered predictors. The custom function can return a character vector of names of the filtered predictors, but this will not work with the `penalty.factor` argument in `nestcv.glmnet()`.
`filter_options`	List of additional arguments passed to the filter function specified by `filterFUN`.
`weights`	Weights applied to each sample for models which can use weights. Note `weights` and `balance` cannot be used at the same time. Weights are not applied in filters.
`balance`	Specifies method for dealing with imbalanced class data. Current options are `"randomsample"` or `"smote"`. See `randomsample()` and `smote()`
`balance_options`	List of additional arguments passed to the balancing function
`modifyX`	Character string specifying the name of a function to modify `x`. This can be an imputation function for replacing missing values, or a more complex function which alters or even adds columns to `x`. The required return value of this function depends on the `modifyX_useY` setting.
`modifyX_useY`	Logical value whether the `x` modifying function makes use of response training data from `y`. If `FALSE` then the `modifyX` function simply needs to return a modified `x` object. If `TRUE` then the `modifyX` function must return a model type object on which `predict()` can be called, so that train and test partitions of `x` can be modified independently.
`modifyX_options`	List of additional arguments passed to the `x` modifying function
`outer_method`	String of either `"cv"` or `"LOOCV"` specifying whether to do k-fold CV or leave one out CV (LOOCV) for the outer folds
`n_outer_folds`	Number of outer CV folds
`n_inner_folds`	Sets number of inner CV folds. Note if `trControl` or `inner_folds` is specified then these supersede `n_inner_folds`.
`outer_folds`	Optional list containing indices of test folds for outer CV. If supplied, `n_outer_folds` is ignored.
`inner_folds`	Optional list of test fold indices for inner CV. This must be structured as a list of the outer folds each containing a list of inner folds. Can only be supplied if balancing is not applied. If supplied, `n_inner_folds` is ignored.
`pass_outer_folds`	Logical indicating whether the same outer folds are used for fitting of the final model when final CV is applied. Note this can only be applied when `n_outer_folds` and the number of inner CV folds specified in `n_inner_folds` or `trControl` are the same and that no balancing is applied.
`cv.cores`	Number of cores for parallel processing of the outer loops.
`multicore_fork`	Logical whether to use forked multicore parallel processing. Forked multicore processing uses `parallel::mclapply`. It is only available on unix/mac as windows does not allow forking. It is set to `FALSE` by default in windows and `TRUE` in unix/mac. Non-forked parallel processing is executed using `parallel::parLapply` or `pbapply::pblapply` if `verbose` is `TRUE`.

`metric`	A string that specifies what summary metric will be used to select the optimal model. By default, "logLoss" is used for classification and "RMSE" is used for regression. Note this differs from the default setting in caret which uses "Accuracy" for classification. See details.
`trControl`	A list of values generated by the `caret` function `caret::trainControl()`. This defines how inner CV training through `caret` is performed. Default for the inner loop is 10-fold CV. Setting this argument overrules `n_inner_folds`. See http://topepo.github.io/caret/using-your-own-model-in-train.html.
`tuneGrid`	Data frame of tuning values, see `caret::train()`.
`savePredictions`	Indicates whether hold-out predictions for each inner CV fold should be saved for ROC curves, accuracy etc see caret::trainControl. Default is `"final"` to capture predictions for inner CV ROC.
`outer_train_predict`	Logical whether to save predictions on outer training folds to calculate performance on outer training folds.
`finalCV`	Logical whether to perform one last round of CV on the whole dataset to determine the final model parameters. If set to `FALSE`, the median of the best hyperparameters from outer CV folds for continuous/ ordinal hyperparameters, or highest voted for categorical hyperparameters, are used to fit the final model. Performance metrics are independent of this last step. If set to `NA`, final model fitting is skipped altogether, which gives a useful speed boost if performance metrics are all that is needed.
`na.option`	Character value specifying how `NA`s are dealt with. `"omit"` is equivalent to `na.action = na.omit`. `"omitcol"` removes cases if there are `NA` in 'y', but columns (predictors) containing `NA` are removed from 'x' to preserve cases. Any other value means that `NA` are ignored (a message is given).
`verbose`	Logical whether to print messages and show progress
`...`	Arguments passed to `caret::train()`

Details

When `finalCV = TRUE`, the final fit on the whole data using is performed first. This helps flag errors generated by `caret` such as missing packages. Parallelisation of the final fit when `finalCV = TRUE` is performed in `caret` using `registerDoParallel`. `caret` itself uses `foreach`.

Parallelisation is performed on the outer CV folds using `parallel::mclapply` by default on unix/mac and `parallel::parLapply` on windows. `mclapply` uses forking which is faster. But some models use multi-threading which may cause issues in some circumstances with forked multicore processing. Setting `multicore_fork` to `FALSE` is slower but can alleviate some caret errors.

If the outer folds are run using parallelisation, then parallelisation in caret must be off, otherwise an error will be generated. Alternatively if you wish to use parallelisation in caret, then parallelisation in `nestcv.train` can be fully disabled by leaving `cv.cores = 1`.

xgboost models fitted via caret using `method = "xgbTree"` or `"xgbLinear"` invoke openMP multithreading on linux/windows by default which causes `nestcv.train` to fail when `cv.cores` >1 (nested parallelisation). Mac OS is unaffected. In order to prevent this, `nestcv.train()` sets openMP threads to 1 if `cv.cores` >1.

For classification, `metric` defaults to using 'logLoss' with the `trControl` arguments `classProbs = TRUE, summaryFunctio` rather than 'Accuracy' which is the default classification metric in `caret`. See `caret::trainControl()`. LogLoss is arguably more consistent than Accuracy for tuning parameters in datasets with small sample size.

Models can be fitted with a single set of fixed parameters, in which case `trControl` defaults to `trainControl(method = "none")` which disables inner CV as it is unnecessary. See https://topepo.github.io/caret/model-training-and-tuning.html#fitting-models-without-parameter-tuning

Value

An object with S3 class "nestcv.train"

`call`	the matched call
`output`	Predictions on the left-out outer folds
`outer_result`	List object of results from each outer fold containing predictions on left-out outer folds, caret result and number of filtered predictors at each fold.
`outer_folds`	List of indices of outer test folds
`dimx`	dimensions of `x`
`xsub`	subset of `x` containing all predictors used in both outer CV folds and the final model
`y`	original response vector
`yfinal`	final response vector (post-balancing)
`final_fit`	Final fitted caret model using best tune parameters
`final_vars`	Column names of filtered predictors entering final model
`summary_vars`	Summary statistics of filtered predictors
`roc`	ROC AUC for binary classification where available.
`trControl`	`caret::trainControl` object used for inner CV
`bestTunes`	best tuned parameters from each outer fold
`finalTune`	final parameters used for final model
`summary`	Overall performance summary. Accuracy and balanced accuracy for classification. ROC AUC for binary classification. RMSE for regression.

Author(s)

Myles Lewis

Examples

```
## sigmoid function
sigmoid <- function(x) {1 / (1 + exp(-x))}

## load iris dataset and simulate a binary outcome
data(iris)
x <- iris[, 1:4]
colnames(x) <- c("marker1", "marker2", "marker3", "marker4")
```

```
x <- as.data.frame(apply(x, 2, scale))
y2 <- sigmoid(0.5 * x$marker1 + 2 * x$marker2) > runif(nrow(x))
y2 <- factor(y2, labels = c("class1", "class2"))

## Example using random forest with caret
cvrf <- nestcv.train(y2, x, method = "rf",
                     n_outer_folds = 3,
                     cv.cores = 2)
summary(cvrf)

## Example of glmnet tuned using caret
## set up small tuning grid for quick execution
## length.out of 20-100 is usually recommended for lambda
## and more alpha values ranging from 0-1
tg <- expand.grid(lambda = exp(seq(log(2e-3), log(1e0), length.out = 5)),
                  alpha = 1)

ncv <- nestcv.train(y = y2, x = x,
                    method = "glmnet",
                    n_outer_folds = 3,
                    tuneGrid = tg, cv.cores = 2)
summary(ncv)

## plot tuning for outer fold #1
plot(ncv$outer_result[[1]]$fit, xTrans = log)

## plot final ROC curve
plot(ncv$roc)

## plot ROC for left-out inner folds
inroc <- innercv_roc(ncv)
plot(inroc)

## example to show use of custom fold indices for 5 x 5-fold nested CV
library(caret)
y <- iris$Species
out_folds <- createFolds(y, k = 5)
in_folds <- lapply(out_folds, function(i) {
  ytrain <- y[-i]
  createFolds(ytrain, k = 5)
})

res <- nestcv.train(y, x, method="rf", cv.cores = 2,
                    pass_outer_folds = TRUE,
                    inner_folds = in_folds,
                    outer_folds = out_folds)
summary(res)
res$outer_folds
res$final_fit$control$indexOut  # same as outer_folds
```

`one_hot` *One-hot encode*

Description

Fast one-hot encoding of all factor and character columns in a dataframe to convert it into a numeric matrix by creating dummy (binary) columns.

Usage

```
one_hot(x, all_levels = FALSE, rename_binary = TRUE, sep = ".")
```

Arguments

`x`	A dataframe, matrix or tibble. Matrices are returned untouched.
`all_levels`	Logical, whether to create dummy variables for all levels of each factor. Default is `FALSE` to avoid issues with regression models.
`rename_binary`	Logical, whether to rename binary factors by appending the 2nd level of the factor to aid interpretation of encoded factor levels and to allow consistency with naming.
`sep`	Character for separating factor variable names and levels for encoded columns.

Details

Binary factor columns and logical columns are converted to integers (0 or 1). Multi-level unordered factors are converted to multiple columns of 0/1 (dummy variables): if `all_levels` is set to `FALSE` (the default), then the first level is assumed to be a reference level and additional columns are created for each additional level; if `all_levels` is set to `TRUE` one column is used for each level. Unused levels are dropped. Character columns are first converted to factors and then encoded. Ordered factors are replaced by their internal codes. Numeric or integer columns are left untouched.

Having dummy variables for all levels of a factor can cause problems with multicollinearity in regression (the dummy variable trap), so `all_levels` is set to `FALSE` by default which is necessary for regression models such as `glmnet` (equivalent to full rank parameterisation). However, setting `all_levels` to `TRUE` can aid with interpretability (e.g. with SHAP values), and in some cases filtering might result in some dummy variables being excluded. Note this function is designed to quickly generate dummy variables for more general machine learning purposes. To create a proper design matrix object for regression models, use `model.matrix()`.

Value

A numeric matrix with the same number of rows as the input data. Dummy variable columns replace the input factor or character columns. Numeric columns are left intact.

See Also

`caret::dummyVars()`, `model.matrix()`

Examples

```
data(iris)
x <- iris
x2 <- one_hot(x)
head(x2)  # 3 columns for Species

x2 <- one_hot(x, all_levels = FALSE)
head(x2)  # 2 columns for Species
```

---

outercv *Outer cross-validation of selected models*

Description

This is a convenience function designed to use a single loop of cross-validation to quickly evaluate performance of specific models (random forest, naive Bayes, lm, glm) with fixed hyperparameters and no tuning. If tuning of parameters on data is required, full nested CV with inner CV is needed to tune model hyperparameters (see nestcv.train).

Usage

```
outercv(y, ...)

## Default S3 method:
outercv(
  y,
  x,
  model,
  filterFUN = NULL,
  filter_options = NULL,
  weights = NULL,
  balance = NULL,
  balance_options = NULL,
  modifyX = NULL,
  modifyX_useY = FALSE,
  modifyX_options = NULL,
  outer_method = c("cv", "LOOCV"),
  n_outer_folds = 10,
  outer_folds = NULL,
  cv.cores = 1,
  multicore_fork = (Sys.info()["sysname"] != "Windows"),
  predict_type = "prob",
  outer_train_predict = FALSE,
  returnList = FALSE,
  final = TRUE,
  na.option = "pass",
  verbose = FALSE,
  suppressMsg = verbose,
  ...
)

## S3 method for class 'formula'
outercv(
  formula,
  data,
  model,
  outer_method = c("cv", "LOOCV"),
  n_outer_folds = 10,
  outer_folds = NULL,
  cv.cores = 1,
  multicore_fork = (Sys.info()["sysname"] != "Windows"),
  predict_type = "prob",
  outer_train_predict = FALSE,
  verbose = FALSE,
  suppressMsg = verbose,
  ...,
  na.action = na.fail
)
```

Arguments

`y`	Response vector
`...`	Optional arguments passed to the function specified by `model`.
`x`	Matrix or dataframe of predictors
`model`	Character value or function of the model to be fitted.
`filterFUN`	Filter function, e.g. ttest_filter or relieff_filter. Any function can be provided and is passed `y` and `x`. Ideally returns a numeric vector with indices of filtered predictors. The custom function can return a character vector of names of the filtered predictors, but this will not work with the `penalty.factor` argument in `nestcv.glmnet()`. Not available if `outercv` is called with a formula.
`filter_options`	List of additional arguments passed to the filter function specified by `filterFUN`.
`weights`	Weights applied to each sample for models which can use weights. Note `weights` and `balance` cannot be used at the same time. Weights are not applied in filters.
`balance`	Specifies method for dealing with imbalanced class data. Current options are `"randomsample"` or `"smote"`. Not available if `outercv` is called with a formula. See `randomsample()` and `smote()`
`balance_options`	List of additional arguments passed to the balancing function
`modifyX`	Character string specifying the name of a function to modify `x`. This can be an imputation function for replacing missing values, or a more complex function which alters or even adds columns to `x`. The required return value of this function depends on the `modifyX_useY` setting.

`modifyX_useY`	Logical value whether the `x` modifying function makes use of response training data from `y`. If `FALSE` then the `modifyX` function simply needs to return a modified `x` object. If `TRUE` then the `modifyX` function must return a model type object on which `predict()` can be called, so that train and test partitions of `x` can be modified independently.
`modifyX_options`	List of additional arguments passed to the `x` modifying function
`outer_method`	String of either `"cv"` or `"LOOCV"` specifying whether to do k-fold CV or leave one out CV (LOOCV) for the outer folds
`n_outer_folds`	Number of outer CV folds
`outer_folds`	Optional list containing indices of test folds for outer CV. If supplied, `n_outer_folds` is ignored.
`cv.cores`	Number of cores for parallel processing of the outer loops.
`multicore_fork`	Logical whether to use forked multicore parallel processing. Forked multicore processing uses `parallel::mclapply`. It is only available on unix/mac as windows does not allow forking. It is set to `FALSE` by default in windows and `TRUE` in unix/mac. Non-forked parallel processing is executed using `parallel::parLapply` or `pbapply::pblapply` if `verbose` is `TRUE`.
`predict_type`	Only used with binary classification. Calculation of ROC AUC requires predicted class probabilities from fitted models. Most model functions use syntax of the form `predict(..., type = "prob")`. However, some models require a different `type` to be specified, which can be passed to `predict()` via `predict_type`.
`outer_train_predict`	Logical whether to save predictions on outer training folds to calculate performance on outer training folds.
`returnList`	Logical whether to return list of results after main outer CV loop without concatenating results. Useful for debugging.
`final`	Logical whether to fit final model.
`na.option`	Character value specifying how `NA`s are dealt with. `"omit"` is equivalent to `na.action = na.omit`. `"omitcol"` removes cases if there are `NA` in 'y', but columns (predictors) containing `NA` are removed from 'x' to preserve cases. Any other value means that `NA` are ignored (a message is given).
`verbose`	Logical whether to print messages and show progress
`suppressMsg`	Logical whether to suppress messages and printed output from model functions. This is necessary when using forked multicore parallelisation.
`formula`	A formula describing the model to be fitted
`data`	A matrix or data frame containing variables in the model.
`na.action`	Formula S3 method only: a function to specify the action to be taken if NAs are found. The default action is for the procedure to fail. An alternative is `na.omit`, which leads to rejection of cases with missing values on any required variable. (NOTE: If given, this argument must be named.)

Details

Some predictive model functions do not have an x & y interface. If the function specified by `model` requires a formula, `x` & `y` will be merged into a dataframe with `model()` called with a formula equivalent to `y ~ .`.

The S3 formula method for `outercv` is not really recommended with large data sets - it is envisaged to be primarily used to compare performance of more basic models e.g. `lm()` specified by formulae for example incorporating interactions. NOTE: filtering is not available if `outercv` is called with a formula - use the `x-y` interface instead.

An alternative method of tuning a single model with fixed parameters is to use nestcv.train with `tuneGrid` set as a single row of a data.frame. The parameters which are needed for a specific model can be identified using `caret::modelLookup()`.

Case weights can be passed to model function which accept these, however `outercv` assumes that these are passed to the model via an argument named `weights`.

Note that in the case of `model = "lm"`, although additional arguments e.g. `subset`, `weights`, `offset` are passed into the model function via `"..."` the scoping is known to go awry. Avoid using these arguments with `model = "lm"`.

`NA` handling differs between the default S3 method and the formula S3 method. The `na.option` argument takes a character string, while the more typical `na.action` argument takes a function.

Value

An object with S3 class "outercv"

`call`	the matched call
`output`	Predictions on the left-out outer folds
`outer_result`	List object of results from each outer fold containing predictions on left-out outer folds, model result and number of filtered predictors at each fold.
`dimx`	vector of number of observations and number of predictors
`outer_folds`	List of indices of outer test folds
`final_fit`	Final fitted model on whole data
`final_vars`	Column names of filtered predictors entering final model
`roc`	ROC AUC for binary classification where available.
`summary`	Overall performance summary. Accuracy and balanced accuracy for classification. ROC AUC for binary classification. RMSE for regression.

Examples

```
## Classification example

## sigmoid function
sigmoid <- function(x) {1 / (1 + exp(-x))}

# load iris dataset and simulate a binary outcome
data(iris)
dt <- iris[, 1:4]
```

```
colnames(dt) <- c("marker1", "marker2", "marker3", "marker4")
dt <- as.data.frame(apply(dt, 2, scale))
x <- dt
y2 <- sigmoid(0.5 * dt$marker1 + 2 * dt$marker2) > runif(nrow(dt))
y2 <- factor(y2)

## Random forest
library(randomForest)
cvfit <- outercv(y2, x, "randomForest")
summary(cvfit)
plot(cvfit$roc)

## Mixture discriminant analysis (MDA)
if (requireNamespace("mda", quietly = TRUE)) {
  library(mda)
  cvfit <- outercv(y2, x, "mda", predict_type = "posterior")
  summary(cvfit)
}


## Example with continuous outcome
y <- -3 + 0.5 * dt$marker1 + 2 * dt$marker2 + rnorm(nrow(dt), 0, 2)
dt$outcome <- y

## simple linear model - formula interface
cvfit <- outercv(outcome ~ ., data = dt, model = "lm")
summary(cvfit)

## random forest for regression
cvfit <- outercv(y, x, "randomForest")
summary(cvfit)

## example with lm_filter() to reduce input predictors
cvfit <- outercv(y, x, "randomForest", filterFUN = lm_filter,
                 filter_options = list(nfilter = 2, p_cutoff = NULL))
summary(cvfit)
```

`plot.cva.glmnet` *Plot lambda across range of alphas*

Description

Different types of plot showing cross-validated tuning of alpha and lambda from elastic net regression via glmnet::glmnet. If `xaxis` is set to `"lambda"`, log lambda is on the x axis while the tuning metric (log loss, deviance, accuracy, AUC etc) is on the y axis. Multiple alpha values are shown by different colours. If `xaxis` is set to `"alpha"`, alpha is on the x axis with the tuning metric on y, with error bars showing metric SD. if `xaxis` is set to `"nvar"` the number of non-zero coefficients is shown on x and how this relates to model deviance/ accuracy on y.

Usage

```
## S3 method for class 'cva.glmnet'
plot(
  x,
  xaxis = c("lambda", "alpha", "nvar"),
  errorBar = (xaxis == "alpha"),
  errorWidth = 0.015,
  min.pch = NULL,
  scheme = NULL,
  palette = "zissou",
  showLegend = "bottomright",
  ...
)
```

Arguments

`x`	Object of class 'cva.glmnet'.
`xaxis`	String specifying what is plotted on the x axis, either log lambda, alpha or the number of non-zero coefficients.
`errorBar`	Logical whether to control error bars for the standard deviation of model deviance when `xaxis = 'lambda'`. Because of overlapping lines, only the deviance of the top and bottom points at a given lambda are shown.
`errorWidth`	Width of error bars.
`min.pch`	Plotting 'character' for the minimum point of each curve. Not shown if set to `NULL`. See points.
`scheme`	Colour scheme. Overrides the `palette` argument.
`palette`	Palette name (one of `hcl.pals()`) which is passed to hcl.colors.
`showLegend`	Either a keyword to position the legend or `NULL` to hide the legend.
`...`	Other arguments passed to plot. Use `type = 'p'` to plot a scatter plot instead of a line plot.

Value

No return value

Author(s)

Myles Lewis

See Also

nestcv.glmnet

`plot.prc` *Plot precision-recall curve*

Description

Plots a precision-recall curve using base graphics. It accepts an S3 object of class 'prc', see `prc()`.

Usage

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

Arguments

`x`	An object of class 'prc'
`...`	Optional graphical arguments passed to `plot()`

Value

No return value

See Also

`prc()`

Examples

```
library(mlbench)
data(Sonar)
y <- Sonar$Class
x <- Sonar[, -61]

fit1 <- nestcv.glmnet(y, x, family = "binomial", alphaSet = 1, cv.cores = 2)
fit1$prc <- prc(fit1)  # calculate precision-recall curve

fit2 <- nestcv.train(y, x, method = "gbm", cv.cores = 2)
fit2$prc <- prc(fit2)

plot(fit1$prc)
lines(fit2$prc, col = "red")
```

plot_alphas *Plot cross-validated glmnet alpha*

Description

Plot of cross-validated glmnet alpha parameter against deviance for each outer CV fold.

Usage

```
plot_alphas(x, col = NULL, ...)
```

Arguments

`x`	Fitted "nestcv.glmnet" object
`col`	Optional vector of line colours for each fold
`...`	other arguments passed to plot

Value

No return value

Author(s)

Myles Lewis

See Also

nestcv.glmnet

plot_caret *Plot caret tuning*

Description

Plots the main tuning parameter in models built using caret::train

Usage

```
plot_caret(x, error.col = "darkgrey", ...)
```

Arguments

`x`	Object of class 'train' generated by `caret` function caret::train
`error.col`	Colour of error bars
`...`	Other arguments passed to `plot()`

Value

No return value

`plot_lambdas` *Plot cross-validated glmnet lambdas across outer folds*

Description

Plot of cross-validated glmnet lambda parameter against deviance for each outer CV fold.

Usage

```
plot_lambdas(
  x,
  scheme = NULL,
  palette = "Dark 3",
  showLegend = if (x$outer_method == "cv") "topright" else NULL,
  ...
)
```

Arguments

`x`	Fitted "nestcv.glmnet" object
`scheme`	colour scheme
`palette`	palette name (one of `hcl.pals()`) which is passed to hcl.colors
`showLegend`	Either a keyword to position the legend or `NULL` to hide the legend.
`...`	other arguments passed to plot. Use `type = 'p'` to plot a scatter plot instead of a line plot.

Value

No return value

Author(s)

Myles Lewis

See Also

nestcv.glmnet

`plot_shap_bar` *SHAP importance bar plot*

Description

SHAP importance bar plot

Usage

```
plot_shap_bar(
  shap,
  x,
  sort = TRUE,
  labels = c("Negative", "Positive"),
  top = NULL
)
```

Arguments

`shap`	a matrix of SHAP values
`x`	a matrix or dataframe of feature values containing only features values from the training data. The rows must match rows in `shap`. If a dataframe is supplied it is converted to a numeric matrix using `data.matrix()`.
`sort`	Logical whether to sort predictors by mean absolute SHAP value
`labels`	Character vector of labels for directionality
`top`	Sets a limit on the number of variables plotted or `NULL` to plot all variables. If `top` is set then variables are sorted and `sort` is overrode.

Value

A ggplot2 plot

`plot_shap_beeswarm` *SHAP importance beeswarm plot*

Description

SHAP importance beeswarm plot

Usage

```
plot_shap_beeswarm(
  shap,
  x,
  cex = 0.25,
  corral = "random",
  corral.width = 0.7,
  scheme = c("deepskyblue2", "purple3", "red"),
  sort = TRUE,
  top = NULL,
  ...
)
```

Arguments

`shap`	a matrix of SHAP values
`x`	a matrix or dataframe of feature values containing only features values from the training data. The rows must match rows in `shap`. If a dataframe is supplied it is converted to a numeric matrix using `data.matrix()`.
`cex`	Scaling for adjusting point spacing. See `ggbeeswarm::geom_beeswarm()`.
`corral`	String specifying method used to corral points. See `ggbeeswarm::geom_beeswarm()`.
`corral.width`	Numeric specifying width of corral, passed to `geom_beeswarm`
`scheme`	Colour scheme as a vector of 3 colours
`sort`	Logical whether to sort predictors by mean absolute SHAP value.
`top`	Sets a limit on the number of variables plotted or `NULL` to plot all variables. If `top` is set then variables are sorted and `sort` is overrode.
`...`	Other arguments passed to `ggbeeswarm::geom_beeswarm()` e.g. `size`.

Value

A ggplot2 plot

`plot_varImp` *Variable importance plot*

Description

Plot of variable importance of coefficients of a final fitted 'nestedcv.glmnet' model using ggplot2. Mean expression can be overlaid as the size of points as this can be informative in models of biological attributes.

Usage

```
plot_varImp(x, abs = TRUE, size = TRUE)
```

Arguments

`x`	a 'nestcv.glmnet' class object
`abs`	Logical whether to show absolute value of glmnet coefficients
`size`	Logical whether to show mean expression by size of points

Value

Returns a ggplot2 plot

`plot_var_ranks` *Plot variable importance rankings*

Description

Plots variables selected in models ranked by variable importance across the outer folds as well as the final model.

Usage

```
plot_var_ranks(x, sort = TRUE, cex = 1, corral.width = 0.75, ...)

hist_var_ranks(x, sort = TRUE)
```

Arguments

`x`	A `nestcv.glmnet` or `nestcv.train` fitted object or a list of these, or a `repeatcv` object.
`sort`	Logical whether to sort variable by mean rank.
`cex`	Scaling for adjusting point spacing. See `ggbeeswarm::geom_beeswarm()`.
`corral.width`	Numeric specifying width of corral, passed to `geom_beeswarm`
`...`	Optional arguments passed to `ggbeeswarm::geom_beeswarm()` e.g. `size`.

Value

A ggplot2 plot.

plot_var_stability *Plot variable stability*

Description

Produces a ggplot2 plot of stability (as SEM) of variable importance across models trained and tested across outer CV folds. Overlays frequency with which variables are selected across the outer folds and optionally overlays directionality for binary response outcome.

Usage

```
plot_var_stability(
  x,
  final = TRUE,
  top = NULL,
  direction = 0,
  dir_labels = NULL,
  scheme = c("royalblue", "red"),
  breaks = NULL,
  percent = TRUE,
  level = 1,
  sort = TRUE
)
```

Arguments

`x`	a `nestcv.glmnet` or `nestcv.train` fitted object or a list of these, or a `repeatcv` object.
`final`	Logical whether to restrict variables to only those which ended up in the final fitted model or to include all variables selected across all outer folds.
`top`	Limits number of variables plotted. Set to `NULL` to plot all variables.
`direction`	Integer controlling plotting of directionality for binary or regression models. `0` means no directionality is shown, `1` means directionality is overlaid as a colour, `2` means directionality is reflected in the sign of variable importance. Not available for multiclass caret models.
`dir_labels`	Character vector for controlling the legend when `direction = 1`
`scheme`	Vector of 2 colours for directionality when `direction = 1`
`breaks`	Vector of continuous breaks for legend colour/size
`percent`	Logical for `nestcv.glmnet` objects only, whether to scale coefficients to percentage of the largest coefficient in each model. If set to `FALSE`, model coefficients are shown and `direction` is ignored.
`level`	For multinomial `nestcv.glmnet` models only, either an integer specifying which level of outcome is being examined, or the level can be specified as a character value.
`sort`	Logical whether to sort by mean variable importance. Passed to `var_stability()`.

Value

A ggplot2 plot

See Also

`var_stability()`

`pls_filter` *Partial Least Squares filter*

Description

Filter using coefficients from partial least squares (PLS) regression to select optimal predictors.

Usage

```
pls_filter(
  y,
  x,
  force_vars = NULL,
  nfilter,
  ncomp = 5,
  scale_x = TRUE,
  type = c("index", "names", "full"),
  ...
)
```

Arguments

`y`	Response vector
`x`	Matrix of predictors
`force_vars`	Vector of column names within `x` which are always retained in the model (i.e. not filtered). Default `NULL` means all predictors will be filtered.
`nfilter`	Either a single value for the total number of predictors to return. Or a vector of length `ncomp` to manually return predictors from each PLS component.
`ncomp`	the number of components to include in the PLS model.
`scale_x`	Logical whether to scale predictors before fitting the PLS model. This is recommended.
`type`	Type of vector returned. Default "index" returns indices, "names" returns predictor names, "full" returns a named vector of variable importance.
`...`	Other arguments passed to `pls::plsr()`

Details

The best predictors may overlap between components, so if `nfilter` is specified as a vector, the total number of unique predictors returned may be variable.

Value

Integer vector of indices of filtered parameters (type = "index") or character vector of names (type = "names") of filtered parameters. If `type` is `"full"` full output of coefficients from `plsr` is returned as a list for each model component ordered by highest absolute coefficient.

`prc` *Build precision-recall curve*

Description

Builds a precision-recall curve for a 'nestedcv' model using `prediction()` and `performance()` functions from the ROCR package and returns an object of class 'prc' for plotting.

Usage

```
prc(...)

## Default S3 method:
prc(response, predictor, positive = 2, ...)

## S3 method for class 'data.frame'
prc(output, ...)

## S3 method for class 'nestcv.glmnet'
prc(object, ...)

## S3 method for class 'nestcv.train'
prc(object, ...)

## S3 method for class 'nestcv.SuperLearner'
prc(object, ...)

## S3 method for class 'outercv'
prc(object, ...)

## S3 method for class 'repeatcv'
prc(object, ...)
```

Arguments

`...`	other arguments
`response`	binary factor vector of response of default order controls, cases.
`predictor`	numeric vector of probabilities
`positive`	Either an integer 1 or 2 for the level of response factor considered to be 'positive' or 'relevant', or a character value for that factor.

`output`	data.frame with columns `testy` containing observed response from test folds, and `predyp` predicted probabilities for classification
`object`	a 'nestcv.glmnet', 'nestcv.train', 'nestcv.SuperLearn', 'outercv' or 'repeatcv' S3 class results object.

Value

An object of S3 class 'prc' containing the following fields:

`recall`	vector of recall values
`precision`	vector of precision values
`auc`	area under precision-recall curve value using trapezoid method
`baseline`	baseline precision value

Examples

```
library(mlbench)
data(Sonar)
y <- Sonar$Class
x <- Sonar[, -61]

fit1 <- nestcv.glmnet(y, x, family = "binomial", alphaSet = 1, cv.cores = 2)

fit1$prc <- prc(fit1)  # calculate precision-recall curve
fit1$prc$auc  # precision-recall AUC value

fit2 <- nestcv.train(y, x, method = "gbm", cv.cores = 2)
fit2$prc <- prc(fit2)
fit2$prc$auc

plot(fit1$prc, ylim = c(0, 1))
lines(fit2$prc, col = "red")

res <- nestcv.glmnet(y, x, family = "binomial", alphaSet = 1) |>
  repeatcv(n = 4, rep.cores = 2)

res$prc <- prc(res)  # precision-recall curve on repeated predictions
plot(res$prc)
```

`predict.cva.glmnet` *Predict method for cva.glmnet models*

Description

Makes predictions from a cross-validated glmnet model with optimal value of lambda and alpha.

Usage

```
## S3 method for class 'cva.glmnet'
predict(object, newx, s = "lambda.1se", ...)
```

Arguments

`object`	Fitted `cva.glmnet` object.
`newx`	Matrix of new values for `x` at which predictions are to be made.
`s`	Value of penalty parameter lambda. Default value is `s="lambda.1se"` for consistency with glmnet. Alternatively `s="lambda.min"` can be used.
`...`	Other arguments passed to `predict.cv.glmnet()`.

Value

Object returned depends on arguments in `...` such as `type`.

`predict.hsstan` *Predict from hsstan model fitted within cross-validation*

Description

Draws from the posterior predictive distribution of the outcome.

Usage

```
## S3 method for class 'hsstan'
predict(object, newdata = NULL, type = NULL, ...)
```

Arguments

`object`	An object of class `hsstan`.
`newdata`	Optional data frame containing the variables to use to predict. If `NULL` (default), the model matrix is used. If specified, its continuous variables should be standardized, since the model coefficients are learnt on standardized data.
`type`	Option for binary outcomes only. Default `NULL` will return a class with the highest probability for each sample. If set to `probs`, it will return the probabilities for outcome = 0 and for outcome = 1 for each sample.
`...`	Optional arguments passed to `hsstan::posterior_predict`

Value

For a binary outcome and type = `NULL`, a character vector with the name of the class that has the highest probability for each sample. For a binary outcome and type = `prob`, a 2-dimensional matrix with the probability of class 0 and of class 1 for each sample. For a continuous outcome a numeric vector with the predicted value for each sample.

Author(s)

Athina Spiliopoulou

---

predict.nestcv.glmnet *Predict method for nestcv.glmnet fits*

Description

Obtains predictions from the final fitted model from a nestcv.glmnet object.

Usage

```
## S3 method for class 'nestcv.glmnet'
predict(object, newdata, s = object$final_param["lambda"], modify = FALSE, ...)
```

Arguments

`object`	Fitted `nestcv.glmnet` object
`newdata`	New data to predict outcome on
`s`	Value of lambda for glmnet prediction
`modify`	Logical whether to modify `newdata` based on `modifyX` function. See `modifyX` and `modifyX_useY` arguments in `nestcv.glmnet()`.
`...`	Other arguments passed to `predict.glmnet`.

Details

Checks for missing predictors and if these are sparse (i.e. have zero coefficients) columns of 0 are automatically added to enable prediction to proceed.

Value

Object returned depends on the `...` argument passed to predict method for `glmnet` objects.

See Also

glmnet::glmnet

`predSummary` *Summarise prediction performance metrics*

Description

Quick function to calculate performance metrics: confusion matrix, accuracy and balanced accuracy for classification; ROC AUC for binary classification; RMSE, R^2 and MAE for regression. Multi-class AUC is returned for multinomial classification.

Usage

```
predSummary(output, family = "")
```

Arguments

`output`	data.frame with columns `testy` containing observed response from test folds; `predy` predicted response; `predyp` (optional) predicted probabilities for classification to calculate ROC AUC. For multiclass `output`, columns 3 onwards contain probabilities for each class in columns.
`family`	Optional character value to support specific glmnet models e.g. 'mgaussian', 'cox'.

Details

For multinomial classification, multi-class AUC as defined by Hand and Till is calculated using `pROC::multiclass.roc()`.

Multi-class balanced accuracy is calculated as the mean of the Recall for each class.

R^2 (coefficient of determination) is calculated as 1 - rss / tss, where rss = residual sum of squares, tss = total sum of squares. Pearson r^2 is also provided. Pearson r^2 can only range from 0 to 1, whereas R^2 can range from 1 to -Inf.

Value

An object of class 'predSummary'. For classification a list is returned containing the confusion matrix table and a vector containing accuracy and balanced accuracy for classification, ROC AUC for classification. For regression a vector containing RMSE, R^2 and MAE is returned. For glmnet 'cox' models, Harrell's C-index is returned.

For glmnet 'mgaussian' models, an object of class 'predSummaryMulti' is returned which is a list of vectors with regression metrics (RMSE, R^2, MAE) for each response variable (i.e. each `y` column).

See Also

`metrics()`

pred_nestcv_glmnet *Prediction wrappers to use fastshap with nestedcv*

Description

Prediction wrapper functions to enable the use of the `fastshap` package for generating SHAP values from `nestedcv` trained models.

Usage

```
pred_nestcv_glmnet(x, newdata)

pred_nestcv_glmnet_class1(x, newdata)

pred_nestcv_glmnet_class2(x, newdata)

pred_nestcv_glmnet_class3(x, newdata)

pred_train(x, newdata)

pred_train_class1(x, newdata)

pred_train_class2(x, newdata)

pred_train_class3(x, newdata)

pred_SuperLearner(x, newdata)
```

Arguments

`x`	a `nestcv.glmnet` or `nestcv.train` object
`newdata`	a matrix of new data

Details

These prediction wrapper functions are designed to be used with the `fastshap` package. The functions `pred_nestcv_glmnet` and `pred_train` work for `nestcv.glmnet` and `nestcv.train` models respectively for either binary classification or regression.

For multiclass classification use `pred_nestcv_glmnet_class1`, 2 and 3 for the first 3 classes. Similarly `pred_train_class1` etc for nestcv.train objects. These functions can be inspected and easily modified to analyse further classes.

Value

prediction wrapper function designed for use with `fastshap::explain()`

Examples

```
library(fastshap)

# Boston housing dataset
library(mlbench)
data(BostonHousing2)
dat <- BostonHousing2
y <- dat$cmedv
x <- subset(dat, select = -c(cmedv, medv, town, chas))

# Fit a glmnet model using nested CV
# Only 3 outer CV folds and 1 alpha value for speed
fit <- nestcv.glmnet(y, x, family = "gaussian", n_outer_folds = 3, alphaSet = 1)

# Generate SHAP values using fastshap::explain
# Only using 5 repeats here for speed, but recommend higher values of nsim
sh <- explain(fit, X=x, pred_wrapper = pred_nestcv_glmnet, nsim = 1)

# Plot overall variable importance
plot_shap_bar(sh, x)

# Plot beeswarm plot
plot_shap_beeswarm(sh, x, size = 1)
```

`randomsample` *Oversampling and undersampling*

Description

Random oversampling of the minority group(s) or undersampling of the majority group to compensate for class imbalance in datasets.

Usage

```
randomsample(y, x, minor = NULL, major = 1, yminor = NULL)
```

Arguments

`y`	Vector of response outcome as a factor
`x`	Matrix of predictors
`minor`	Amount of oversampling of the minority class. If set to `NULL` then all classes will be oversampled up to the number of samples in the majority class. To turn off oversampling set `minor = 1`.
`major`	Amount of undersampling of the majority class
`yminor`	Optional character value specifying the level in `y` which is to be oversampled. If `NULL`, this is set automatically to the class with the smallest sample size.

Details

`minor` < 1 and `major` > 1 are ignored.

Value

List containing extended matrix `x` of synthesised data and extended response vector `y`

Examples

```
## Imbalanced dataset
set.seed(1, "L'Ecuyer-CMRG")
x <- matrix(rnorm(150 * 2e+04), 150, 2e+04)  #' predictors
y <- factor(rbinom(150, 1, 0.2))  #' imbalanced binary response
table(y)

## first 30 parameters are weak predictors
x[, 1:30] <- rnorm(150 * 30, 0, 1) + as.numeric(y)*0.5

## Balance x & y outside of CV loop by random oversampling minority group
out <- randomsample(y, x)
y2 <- out$y
x2 <- out$x
table(y2)

## Nested CV glmnet with unnested balancing by random oversampling on
## whole dataset
fit1 <- nestcv.glmnet(y2, x2, family = "binomial", alphaSet = 1,
                      cv.cores=2,
                      filterFUN = ttest_filter)
fit1$summary

## Balance x & y outside of CV loop by random oversampling minority group
out <- randomsample(y, x, minor=1, major=0.4)
y2 <- out$y
x2 <- out$x
table(y2)

## Nested CV glmnet with unnested balancing by random undersampling on
## whole dataset
fit1b <- nestcv.glmnet(y2, x2, family = "binomial", alphaSet = 1,
                       cv.cores=2,
                       filterFUN = ttest_filter)
fit1b$summary

## Balance x & y outside of CV loop by SMOTE
out <- smote(y, x)
y2 <- out$y
x2 <- out$x
table(y2)

## Nested CV glmnet with unnested balancing by SMOTE on whole dataset
fit2 <- nestcv.glmnet(y2, x2, family = "binomial", alphaSet = 1,
                      cv.cores=2,
                      filterFUN = ttest_filter)
fit2$summary

## Nested CV glmnet with nested balancing by random oversampling
fit3 <- nestcv.glmnet(y, x, family = "binomial", alphaSet = 1,
                      cv.cores=2,
                      balance = "randomsample",
                      filterFUN = ttest_filter)
fit3$summary
class_balance(fit3)

## Plot ROC curves
plot(fit1$roc, col='green')
lines(fit1b$roc, col='red')
lines(fit2$roc, col='blue')
lines(fit3$roc)
legend('bottomright', legend = c("Unnested random oversampling",
                                 "Unnested SMOTE",
                                 "Unnested random undersampling",
                                 "Nested balancing"),
       col = c("green", "blue", "red", "black"), lty=1, lwd=2)
```

`ranger_filter` *Random forest ranger filter*

Description

Fits a random forest model via the `ranger` package and ranks variables by variable importance.

Usage

```
ranger_filter(
  y,
  x,
  nfilter = NULL,
  type = c("index", "names", "full"),
  num.trees = 1000,
  mtry = ncol(x) * 0.2,
  ...
)
```

Arguments

`y`	Response vector
`x`	Matrix or dataframe of predictors
`nfilter`	Number of predictors to return. If `NULL` all predictors are returned.

`type`	Type of vector returned. Default "index" returns indices, "names" returns predictor names, "full" returns a named vector of variable importance.
`num.trees`	Number of trees to grow. See ranger::ranger.
`mtry`	Number of predictors randomly sampled as candidates at each split. See ranger::ranger.
`...`	Optional arguments passed to ranger::ranger.

Details

This filter uses the `ranger()` function from the `ranger` package. Variable importance is calculated using mean decrease in gini impurity.

Value

Integer vector of indices of filtered parameters (type = "index") or character vector of names (type = "names") of filtered parameters. If `type` is `"full"` a named vector of variable importance is returned.

`relieff_filter` *ReliefF filter*

Description

Uses ReliefF algorithm from the CORElearn package to rank predictors in order of importance.

Usage

```
relieff_filter(
  y,
  x,
  nfilter = NULL,
  estimator = "ReliefFequalK",
  type = c("index", "names", "full"),
  ...
)
```

Arguments

`y`	Response vector
`x`	Matrix or dataframe of predictors
`nfilter`	Number of predictors to return. If `NULL` all predictors are returned.
`estimator`	Type of algorithm used, see CORElearn::attrEval
`type`	Type of vector returned. Default "index" returns indices, "names" returns predictor names, "full" returns a named vector of variable importance.
`...`	Other arguments passed to CORElearn::attrEval

Value

Integer vector of indices of filtered parameters (type = "index") or character vector of names (type = "names") of filtered parameters. If `type` is `"full"` a named vector of variable importance is returned.

See Also

`CORElearn::attrEval()`

`repeatcv` *Repeated nested CV*

Description

Performs repeated calls to a `nestedcv` model to determine performance across repeated runs of nested CV.

Usage

```
repeatcv(
  expr,
  n = 5,
  repeat_folds = NULL,
  keep = FALSE,
  extra = FALSE,
  progress = TRUE,
  rep.cores = 1L
)
```

Arguments

`expr`	An expression containing a call to `nestcv.glmnet()`, `nestcv.train()`, `nestcv.SuperLearner()` or `outercv()`.
`n`	Number of repeats
`repeat_folds`	Optional list containing fold indices to be applied to the outer CV folds.
`keep`	Logical whether to save repeated outer CV fitted models for variable importance, SHAP etc. Note this can make the resulting object very large.
`extra`	Logical whether additional performance metrics are gathered for binary classification models. See `metrics()`.
`progress`	Logical whether to show progress.
`rep.cores`	Integer specifying number of cores/threads to invoke.

Details

We recommend using this with the R pipe |> (see examples).

When comparing models, it is recommended to fix the sets of outer CV folds used across each repeat for comparing performance between models. The function `repeatfolds()` can be used to create a fixed set of outer CV folds for each repeat.

Parallelisation over repeats is performed using `parallel::mclapply` (not available on windows). Beware that `cv.cores` can still be set within calls to `nestedcv` models (= nested parallelisation). This means that `rep.cores` x `cv.cores` number of processes/forks will be spawned, so be careful not to overload your CPU. In general parallelisation of repeats using `rep.cores` is faster than parallelisation using `cv.cores`.

Value

List of S3 class 'repeatcv' containing:

`call`	the model call
`result`	matrix of performance metrics
`output`	a matrix or dataframe containing the outer CV predictions from each repeat
`roc`	(binary classification models only) a ROC curve object based on predictions across all repeats as returned in `output`, generated by `pROC::roc()`
`fits`	(if `keep = TRUE`) list of length `n` containing slimmed 'nestedcv' model objects for calculating variable importance or SHAP values

Examples

```
data("iris")
dat <- iris
y <- dat$Species
x <- dat[, 1:4]

res <- nestcv.glmnet(y, x, family = "multinomial", alphaSet = 1,
                     n_outer_folds = 4) |>
       repeatcv(3, rep.cores = 2)
res
summary(res)

## set up fixed fold indices
set.seed(123, "L'Ecuyer-CMRG")
folds <- repeatfolds(y, repeats = 3, n_outer_folds = 4)
res <- nestcv.glmnet(y, x, family = "multinomial", alphaSet = 1,
                     n_outer_folds = 4) |>
       repeatcv(3, repeat_folds = folds, rep.cores = 2)
res
```

`repeatfolds` *Create folds for repeated nested CV*

Description

Create folds for repeated nested CV

Usage

```
repeatfolds(y, repeats = 5, n_outer_folds = 10)
```

Arguments

`y`	Outcome vector
`repeats`	Number of repeats
`n_outer_folds`	Number of outer CV folds

Value

List containing indices of outer CV folds

Examples

```
data("iris")
dat <- iris
y <- dat$Species
x <- dat[, 1:4]

## set up fixed fold indices
set.seed(123, "L'Ecuyer-CMRG")
folds <- repeatfolds(y, repeats = 3, n_outer_folds = 4)

res <- nestcv.glmnet(y, x, family = "multinomial", alphaSet = 1,
                     n_outer_folds = 4, cv.cores = 2) |>
       repeatcv(3, repeat_folds = folds)
res
```

`rf_filter` *Random forest filter*

Description

Fits a random forest model and ranks variables by variable importance.

Usage

```
rf_filter(
  y,
  x,
  nfilter = NULL,
  type = c("index", "names", "full"),
  ntree = 1000,
  mtry = ncol(x) * 0.2,
  ...
)
```

Arguments

`y`	Response vector
`x`	Matrix or dataframe of predictors
`nfilter`	Number of predictors to return. If `NULL` all predictors are returned.
`type`	Type of vector returned. Default "index" returns indices, "names" returns predictor names, "full" returns a named vector of variable importance.
`ntree`	Number of trees to grow. See randomForest::randomForest.
`mtry`	Number of predictors randomly sampled as candidates at each split. See randomForest::randomForest.
`...`	Optional arguments passed to randomForest::randomForest.

Details

This filter uses the `randomForest()` function from the `randomForest` package. Variable importance is calculated using the randomForest::importance function, specifying type 1 = mean decrease in accuracy. See randomForest::importance.

Value

Integer vector of indices of filtered parameters (type = "index") or character vector of names (type = "names") of filtered parameters. If `type` is `"full"` a named vector of variable importance is returned.

`slim` *Slim nestedcv models*

Description

Slims nestedcv objects to only the models in the outer CV folds.

Usage

```
slim(x)
```

Arguments

`x`	A 'nestedcv' or 'cva.glmnet' fitted model object.

Value

For 'nestedcv' objects, a list object of the same class but only containing `outer_result`. For 'cva.glmnet' models, only the cv.glmnet model with the best alpha value is kept. Models for all other values of alpha are discarded.

See Also

`nestcv.glmnet()` `nestcv.train()` `outercv()` `cva.glmnet()`

`smote` *SMOTE*

Description

Synthetic Minority Oversampling Technique (SMOTE) algorithm for imbalanced classification data.

Usage

```
smote(y, x, k = 5, over = NULL, yminor = NULL)
```

Arguments

`y`	Vector of response outcome as a factor
`x`	Matrix of predictors
`k`	Range of KNN to consider for generation of new data
`over`	Amount of oversampling of the minority class. If set to `NULL` then all classes will be oversampled up to the number of samples in the majority class.
`yminor`	Optional character value specifying the level in `y` which is to be oversampled. If `NULL`, this is set automatically to the class with the smallest sample size.

Value

List containing extended matrix `x` of synthesised data and extended response vector `y`

References

Chawla, N. V., Bowyer, K. W., Hall, L. O., and Kegelmeyer, W. P. (2002). *Smote: Synthetic minority over-sampling technique*. Journal of Artificial Intelligence Research, 16:321-357.

`stat_filter` *Univariate filter for binary classification with mixed predictor datatypes*

Description

Univariate statistic filter for dataframes of predictors with mixed numeric and categorical datatypes. Different statistical tests are used depending on the data type of response vector and predictors:

Binary class response: `bin_stat_filter()` t-test for continuous data, chi-squared test for categorical data

Multiclass response: `class_stat_filter()` one-way ANOVA for continuous data, chi-squared test for categorical data

Continuous response: `cor_stat_filter()` correlation (or linear regression) for continuous data and binary data, one-way ANOVA for categorical data

Usage

```
stat_filter(y, x, ...)

bin_stat_filter(
  y,
  x,
  force_vars = NULL,
  nfilter = NULL,
  p_cutoff = 0.05,
  rsq_cutoff = NULL,
  type = c("index", "names", "full", "list"),
  ...
)

class_stat_filter(
  y,
  x,
  force_vars = NULL,
  nfilter = NULL,
  p_cutoff = 0.05,
  rsq_cutoff = NULL,
  type = c("index", "names", "full", "list"),
  ...
)

cor_stat_filter(
  y,
  x,
  cor_method = c("pearson", "spearman", "lm"),
  force_vars = NULL,
  nfilter = NULL,
  p_cutoff = 0.05,
  rsq_cutoff = NULL,
  rsq_method = "pearson",
  type = c("index", "names", "full", "list"),
  ...
)
```

Arguments

`y`	Response vector
`x`	Matrix or dataframe of predictors
`...`	optional arguments, e.g. `rsq_method`: see `collinear()`.
`force_vars`	Vector of column names within `x` which are always retained in the model (i.e. not filtered). Default `NULL` means all predictors will be passed to `filterFUN`.
`nfilter`	Number of predictors to return. If `NULL` all predictors with p-values < `p_cutoff` are returned.
`p_cutoff`	p value cut-off
`rsq_cutoff`	r^2 cutoff for removing predictors due to collinearity. Default `NULL` means no collinearity filtering. Predictors are ranked based on t-test. If 2 or more predictors are collinear, the first ranked predictor by t-test is retained, while the other collinear predictors are removed. See `collinear()`.
`type`	Type of vector returned. Default "index" returns indices, "names" returns predictor names, "full" returns a dataframe of statistics, "list" returns a list of 2 matrices of statistics, one for continuous predictors, one for categorical predictors.
`cor_method`	For `cor_stat_filter()` only, either `"pearson"`, `"spearman"` or `"lm"` controlling whether continuous predictors are filtered by correlation (faster) or regression (slower but allows inclusion of covariates via `force_vars`).
`rsq_method`	character string indicating which correlation coefficient is to be computed. One of "pearson" (default), "kendall", or "spearman". See `collinear()`.

Details

`stat_filter()` is a wrapper which calls `bin_stat_filter()`, `class_stat_filter()` or `cor_stat_filter()` depending on whether `y` is binary, multiclass or continuous respectively. Ordered factors are converted to numeric (integer) levels and analysed as if continuous.

Value

Integer vector of indices of filtered parameters (type = "index") or character vector of names (type = "names") of filtered parameters in order of test p-value. If `type` is `"full"` full output is returned containing a dataframe of statistical results. If `type` is `"list"` the output is returned as a list of 2 matrices containing statistical results separated by continuous and categorical predictors.

Examples

```
library(mlbench)
data(BostonHousing2)
dat <- BostonHousing2
y <- dat$cmedv  ## continuous outcome
x <- subset(dat, select = -c(cmedv, medv, town))

stat_filter(y, x, type = "full")
stat_filter(y, x, nfilter = 5, type = "names")
stat_filter(y, x)

data(iris)
y <- iris$Species  ## 3 class outcome
x <- subset(iris, select = -Species)
stat_filter(y, x, type = "full")
```

`summary_vars` *Summarise variables*

Description

Summarise variables

Usage

```
summary_vars(x)
```

Arguments

x	Matrix or dataframe with variables in columns

Value

A matrix with variables in rows and mean, median and SD for each variable or number of levels if the variable is a factor. If `NA` are detected, an extra column `n.NA` is added with the numbers of `NA` for each variable.

`supervisedPCA` *Supervised PCA plot*

Description

Performs supervised principle component analysis (PCA) after filtering dataset to help determine whether filtering has been useful for separating samples according to the outcome variable.

Usage

```
supervisedPCA(y, x, filterFUN = NULL, filter_options = NULL, plot = TRUE, ...)
```

Arguments

`y`	Response vector
`x`	Matrix of predictors
`filterFUN`	Filter function, e.g. ttest_filter or relieff_filter. Any function can be provided and is passed `y` and `x`. Must return a character vector with names of filtered predictors.
`filter_options`	List of additional arguments passed to the filter function specified by `filterFUN`.
`plot`	Logical whether to plot a ggplot2 object or return the PC scores
`...`	Optional arguments passed to `princomp()`

Value

If `plot=TRUE` returns a ggplot2 plot, otherwise returns the principle component scores.

`train_preds` *Outer training fold predictions*

Description

Obtain predictions on outer training folds which can be used for performance metrics and ROC curves.

Usage

```
train_preds(x)
```

Arguments

`x`	a `nestcv.glmnet`, `nestcv.train` or `outercv` fitted object

Details

Note: the argument `outer_train_predict` must be set to `TRUE` in the original call to either `nestcv.glmnet`, `nestcv.train` or `outercv`.

Value

Dataframe with columns `ytrain` and `predy` containing observed and predicted values from training folds. For binomial and multinomial models additional columns are added with class probabilities or log likelihood values.

`train_roc` *Build ROC curve from outer CV training folds*

Description

Build ROC (receiver operating characteristic) curve from outer training folds. Object can be plotted using `plot()` or passed to functions `pROC::auc()` etc.

Usage

```
train_roc(x, direction = "<", ...)
```

Arguments

`x`	a `nestcv.glmnet`, `nestcv.train` or `outercv` object
`direction`	Set ROC directionality pROC::roc
`...`	Other arguments passed to pROC::roc

Details

Note: the argument `outer_train_predict` must be set to `TRUE` in the original call to either `nestcv.glmnet`, `nestcv.train` or `outercv`.

Value

`"roc"` object, see pROC::roc

train_summary *Summarise performance on outer training folds*

Description

Calculates performance metrics on outer training folds: confusion matrix, accuracy and balanced accuracy for classification; ROC AUC for binary classification; RMSE, R^2 and mean absolute error (MAE) for regression.

Usage

```
train_summary(x)
```

Arguments

x	a `nestcv.glmnet`, `nestcv.train` or `outercv` object

Details

Note: the argument `outer_train_predict` must be set to `TRUE` in the original call to either `nestcv.glmnet`, `nestcv.train` or `outercv`.

Value

Returns performance metrics from outer training folds, see predSummary

See Also

predSummary

Examples

```
data(iris)
x <- iris[, 1:4]
y <- iris[, 5]

fit <- nestcv.glmnet(y, x,
                     family = "multinomial",
                     alpha = 1,
                     outer_train_predict = TRUE,
                     n_outer_folds = 3)
summary(fit)
innercv_summary(fit)
train_summary(fit)

fit2 <- nestcv.train(y, x,
                    model="svm",
                    outer_train_predict = TRUE,
                    n_outer_folds = 3,
                   cv.cores = 2)
summary(fit2)
innercv_summary(fit2)
train_summary(fit2)
```

---

ttest_filter *Univariate filters*

Description

A selection of simple univariate filters using t-test, Wilcoxon test, one-way ANOVA or correlation (Pearson or Spearman) for ranking variables. These filters are designed for speed. `ttest_filter` uses the `Rfast` package, `wilcoxon_filter` (Mann-Whitney) test uses matrixTests::row_wilcoxon_twosample, `anova_filter` uses matrixTests::col_oneway_welch (Welch's F-test) from the `matrixTests` package. Can be applied to all or a subset of predictors. For mixed datasets (combined continuous & categorical) see `stat_filter()`

Usage

```
ttest_filter(
  y,
  x,
  force_vars = NULL,
  nfilter = NULL,
  p_cutoff = 0.05,
  rsq_cutoff = NULL,
  type = c("index", "names", "full"),
  keep_factors = TRUE,
  ...
)

anova_filter(
  y,
  x,
  force_vars = NULL,
  nfilter = NULL,
  p_cutoff = 0.05,
  rsq_cutoff = NULL,
  type = c("index", "names", "full"),
  keep_factors = TRUE,
  ...
)

wilcoxon_filter(
  y,
  x,
  force_vars = NULL,
  nfilter = NULL,
  p_cutoff = 0.05,
  rsq_cutoff = NULL,
  type = c("index", "names", "full"),
  exact = FALSE,
  keep_factors = TRUE,
  ...
)

correl_filter(
  y,
  x,
  method = "pearson",
  force_vars = NULL,
  nfilter = NULL,
  p_cutoff = 0.05,
  rsq_cutoff = NULL,
  type = c("index", "names", "full"),
  keep_factors = TRUE,
  ...
)
```

Arguments

`y`	Response vector
`x`	Matrix or dataframe of predictors
`force_vars`	Vector of column names within `x` which are always retained in the model (i.e. not filtered). Default `NULL` means all predictors will be passed to `filterFUN`.
`nfilter`	Number of predictors to return. If `NULL` all predictors with p-values < `p_cutoff` are returned.
`p_cutoff`	p value cut-off
`rsq_cutoff`	r^2 cutoff for removing predictors due to collinearity. Default `NULL` means no collinearity filtering. Predictors are ranked based on t-test. If 2 or more predictors are collinear, the first ranked predictor by t-test is retained, while the other collinear predictors are removed. See `collinear()`.
`type`	Type of vector returned. Default "index" returns indices, "names" returns predictor names, "full" returns a matrix of p values.
`keep_factors`	Logical affecting factors with 3 or more levels. Dataframes are coerced to a matrix using data.matrix. Binary factors are converted to numeric values 0/1 and analysed as such. If `keep_factors` is `TRUE` (the default), factors with 3 or more levels are not filtered and are retained. If `keep_factors` is `FALSE`, they are removed.
`...`	optional arguments, including `rsq_method` passed to `collinear()` or arguments passed to matrixTests::row_wilcoxon_twosample in `wilcoxon_filter()`.
`exact`	Logical whether exact or approximate p-value is calculated. Default is `FALSE` for speed.

`method`	Type of correlation, either "pearson" or "spearman".

Value

Integer vector of indices of filtered parameters (type = "index") or character vector of names (type = "names") of filtered parameters in order of t-test p-value. If `type` is `"full"` full output from Rfast::ttests is returned.

See Also

`lm_filter()` `stat_filter()`

Examples

```
## sigmoid function
sigmoid <- function(x) {1 / (1 + exp(-x))}

## load iris dataset and simulate a binary outcome
data(iris)
dt <- iris[, 1:4]
colnames(dt) <- c("marker1", "marker2", "marker3", "marker4")
dt <- as.data.frame(apply(dt, 2, scale))
y2 <- sigmoid(0.5 * dt$marker1 + 2 * dt$marker2) > runif(nrow(dt))
y2 <- factor(y2, labels = c("C1", "C2"))

ttest_filter(y2, dt)  # returns index of filtered predictors
ttest_filter(y2, dt, type = "name")  # shows names of predictors
ttest_filter(y2, dt, type = "full")  # full results table

data(iris)
dt <- iris[, 1:4]
y3 <- iris[, 5]
anova_filter(y3, dt)  # returns index of filtered predictors
anova_filter(y3, dt, type = "full")  # shows names of predictors
anova_filter(y3, dt, type = "name")  # full results table
```

`txtProgressBar2` *Text Progress Bar 2*

Description

Text progress bar in the R console. Modified from `utils::txtProgressBar()` to include title and timing.

Usage

```
txtProgressBar2(
  min = 0,
  max = 1,
  initial = 0,
  char = "=",
  width = NA,
  title = ""
)
```

Arguments

`min`	Numeric value for minimum of the progress bar.
`max`	Numeric value for maximum of the progress bar.
`initial`	Initial value for the progress bar.
`char`	The character (or character string) to form the progress bar.
`width`	The width of the progress bar, as a multiple of the width of `char`. If `NA`, the default, the number of characters is that which fits into `getOption("width")`.
`title`	Title for the progress bar.

Details

Use `utils::setTxtProgressBar()` to set the progress bar and `close()` to close it.

Value

An object of class "`txtProgressBar`".

`var_direction` *Variable directionality*

Description

Determines directionality of final predictors for binary or regression models, using the sign of the t-statistic or correlation coefficient respectively for each variable compared to the outcomes.

Usage

```
var_direction(object)
```

Arguments

`object`	a `nestcv.glmnet` or `nestcv.train` fitted model

Details

Categorical features with >2 levels are assumed to have a meaningful order for the purposes of directionality. Factors are coerced to ordinal using `data.matrix()`. If factors are multiclass then directionality results should be ignored.

Value

named vector showing the directionality of final predictors. If the response vector is multinomial `NULL` is returned.

`var_stability` *Variable stability*

Description

Uses variable importance across models trained and tested across outer CV folds to assess stability of variable importance. For glmnet, variable importance is measured as the absolute model coefficients, optionally scaled as a percentage. The frequency with which each variable is selected in outer folds as well as the final model is also returned which is helpful for sparse models or with filters to determine how often variables end up in the model in each fold. For glmnet, the direction of effect is taken directly from the sign of model coefficients. For `caret` models, direction of effect is not readily available, so as a substitute, the directionality of each predictor is determined by the function `var_direction()` using the sign of a t-test for binary classification or the sign of regression coefficient for continuous outcomes (not available for multiclass caret models). To better understand direction of effect of each predictor within the final model, we recommend using SHAP values - see the vignette "Explaining nestedcv models with Shapley values". See `pred_train()` for an example.

Usage

```
var_stability(x, ...)

## S3 method for class 'nestcv.glmnet'
var_stability(
  x,
  ranks = FALSE,
  summary = TRUE,
  percent = TRUE,
  level = 1,
  sort = TRUE,
  ...
)

## S3 method for class 'nestcv.train'
var_stability(x, ranks = FALSE, summary = TRUE, sort = TRUE, ...)

## S3 method for class 'repeatcv'
var_stability(x, ...)
```

Arguments

`x`	a `nestcv.glmnet` or `nestcv.train` fitted object or a list of these, or a `repeatcv` object.
`...`	Optional arguments for compatibility
`ranks`	Logical whether to rank variables by importance
`summary`	Logical whether to return summary statistics on variable importance. Ignored if `ranks` is `TRUE`.
`percent`	Logical for `nestcv.glmnet` objects only, whether to scale coefficients to percentage of the largest coefficient in each model
`level`	For multinomial `nestcv.glmnet` models only, either an integer specifying which level of outcome is being examined, or the level can be specified as a character value
`sort`	Logical whether to sort variables by mean importance

Details

Note that for caret models `caret::varImp()` may require the model package to be fully loaded in order to function. During the fitting process `caret` often only loads the package by namespace.

Value

If `ranks` is `FALSE` and `summary` is `TRUE`, returns a dataframe containing mean, sd, sem of variable importance and frequency by which each variable is selected in outer folds. If `summary` is `FALSE`, a matrix of either variable importance or, if `ranks = TRUE`, rankings across the outer folds and the final model is returned, with variables in rows and folds in columns.

See Also

`cv_coef()` `cv_varImp()` `pred_train()`

`weight` *Calculate weights for class imbalance*

Description

Calculate weights for class imbalance

Usage

```
weight(y)
```

Arguments

`y`	Factor or character response vector. If a character vector is supplied it is coerced into a factor. Unused levels are dropped.

Value

Vector of weights

Index