

Package ‘nestedcv’

May 30, 2023

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

Version 0.6.4

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 -fold cross-validation for lasso and elastic-net regularised linear models via the 'glmnet' package and other machine learning models via the 'caret' package. 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](https://doi.org/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](https://doi.org/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, reshape2, Rfast, rlang

RoxygenNote 7.2.1

Suggests Boruta, CORElearn, fastshap, gbm, ggbeeswarm, ggpubr, hsstan, mda, mlbench, randomForest, ranger, RcppEigen, rmarkdown, knitr, SuperLearner

VignetteBuilder knitr

NeedsCompilation no

Author Myles Lewis [aut, cre] (<<https://orcid.org/0000-0001-9365-5345>>),
Athina Spiliopoulou [aut] (<<https://orcid.org/0000-0002-5929-6585>>),
Katriona Goldmann [aut] (<<https://orcid.org/0000-0002-9073-6323>>)

Repository CRAN

Date/Publication 2023-05-30 10:50:02 UTC

R topics documented:

anova_filter	3
barplot_var_stability	4
boot_filter	5
boot_ttest	6
boruta_filter	7
boxplot_model	8
class_balance	8
coef.cva.glmnet	9
coef.nestcv.glmnet	9
collinear	10
combo_filter	10
correls2	11
correl_filter	12
cva.glmnet	13
cv_coef	14
cv_varImp	14
glmnet_coefs	15
glmnet_filter	15
innercv_preds	17
innercv_roc	17
innercv_summary	19
layer_filter	20
lm_filter	21
model.hsstan	22
nestcv.glmnet	24
nestcv.SuperLearner	27
nestcv.train	29
outercv	33
plot.cva.glmnet	37
plot_alphas	38
plot_caret	39
plot_lambdas	40
plot_shap_bar	41
plot_shap_beeswarm	41
plot_varImp	42
plot_var_stability	43
predict.hsstan	44
predict.nestcv.glmnet	45
predSummary	46
pred_nestcv_glmnet	46
randomsample	48
ranger_filter	50
relieff_filter	51
rf_filter	52
smote	53
summary_vars	53

supervisedPCA	54
train_preds	54
train_roc	55
train_summary	56
ttest_filter	57
var_direction	58
var_stability	59
weight	60
wilcoxon_filter	60

Index 62

anova_filter	<i>ANOVA filter</i>
--------------	---------------------

Description

Simple univariate filter using anova (Welch's F-test) using the Rfast package for speed.

Usage

```
anova_filter(
  y,
  x,
  force_vars = NULL,
  nfilter = NULL,
  p_cutoff = 0.05,
  rsq_cutoff = NULL,
  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 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 anova test. If 2 or more predictors are collinear, the first ranked predictor by anova 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.
...	optional arguments, e.g. rsq_method: see collinear() .

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 [Rfast::fests](#) is returned.

Examples

```
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
```

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,
  breaks = NULL,
  percent = TRUE,
  level = 1,
  sort = TRUE
)
```

Arguments

x	a <code>nestcv.glmnet</code> or <code>nestcv.train</code> 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
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	<i>Bootstrap for filter functions</i>
-------------	---------------------------------------

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	<i>Boruta filter</i>
---------------	----------------------

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_model	<i>Boxplot model predictors</i>
---------------	---------------------------------

Description

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

Usage

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

Arguments

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

Value

No return value

Author(s)

Myles Lewis

See Also

[nestedcv.glmnet](#)

class_balance	<i>Check class balance in training folds</i>
---------------	--

Description

Check class balance in training folds

Usage

```
class_balance(object)

## Default S3 method:
class_balance(object)

## S3 method for class 'nestedcv.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.

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 `coef.glmnet`

Value

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

collinear	<i>Filter to reduce collinearity in predictors</i>
-----------	--

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	<i>Combo filter</i>
--------------	---------------------

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

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 <code>cor</code>

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

`correl_filter`*Correlation filter*

Description

Filter using correlation (Pearson or Spearman) for ranking variables.

Usage

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

Arguments

<code>y</code>	Response vector
<code>x</code>	Matrix of predictors
<code>force_vars</code>	Vector of column names within <code>x</code> which are always retained in the model (i.e. not filtered). Default <code>NULL</code> means all predictors will be passed to <code>filterFUN</code> .
<code>nfilter</code>	Number of predictors to return. If <code>NULL</code> all predictors with <code>p</code> values $< p_cutoff$ are returned.
<code>p_cutoff</code>	<code>p</code> value cut-off
<code>method</code>	Type of correlation, either "pearson" or "spearman".
<code>type</code>	Type of vector returned. Default "index" returns indices, "names" returns predictor names, "full" returns a matrix of <code>p</code> -values.
<code>...</code>	Further arguments passed to correls

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 [correls](#) is returned.

cva.glmnet	<i>Cross-validation of alpha for glmnet</i>
------------	---

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 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 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

[cv.glmnet](#), [glmnet](#)

cv_coef	<i>Coefficients from outer CV glmnet models</i>
---------	---

Description

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

Usage

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

Arguments

x	a <code>nestcv.glmnet</code> 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	<i>Extract variable importance from outer CV caret models</i>
-----------	---

Description

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

Usage

```
cv_varImp(x)
```

Arguments

x	a <code>nestcv.train</code> 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	<i>glmnet coefficients</i>
--------------	----------------------------

Description

Convenience function for retrieving coefficients from a [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

<code>fit</code>	A cv.glmnet fitted model object.
<code>s</code>	Value of lambda. See coef.glmnet and predict.cv.glmnet
<code>...</code>	Other arguments passed to coef.glmnet

Value

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

glmnet_filter	<i>glmnet filter</i>
---------------	----------------------

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 <code>glm()</code> family object. See <code>glmnet()</code> . 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 <code>glmnet</code>

Details

The `glmnet` elastic net mixing parameter α 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`.

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](#)

innercv_preds	<i>Inner CV predictions</i>
---------------	-----------------------------

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	<i>Build ROC curve from left-out folds from inner CV</i>
-------------	--

Description

Build ROC (receiver operating characteristic) curve from left-out folds from inner CV. Object can be plotted using `plot()` or passed to functions `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	<i>Summarise performance on inner CV test folds</i>
-----------------	---

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)
```

layer_filter	<i>Multilayer filter</i>
--------------	--------------------------

Description

Experimental filter designed for use with imbalanced datasets. Each round a simple t-test is used to rank predictors and keep a certain number. After each round a set number of cases are culled determined as the most outlying cases - those which if used as a cutoff for classification have the smallest number of misclassified cases. The t-test is repeated on the culled dataset so that after successive rounds the most influential outlying samples have been removed and different samples drive the t-test filter.

Usage

```
layer_filter(
  y,
  x,
  nfilter = NULL,
  imbalance = TRUE,
  cull = 5,
  force_vars = NULL,
  verbose = FALSE,
  type = c("index", "names", "full")
)
```

Arguments

y	Response vector
x	Matrix of predictors
nfilter	Vector of number of target predictors to keep at each round. The length of this vector determines the number of rounds of culling.
imbalance	Logical whether to assume the dataset is imbalanced, in which case samples are only culled from the majority class.
cull	number of samples to cull at each round
force_vars	not implemented yet
verbose	whether to show sample IDs of culled individuals at each round
type	Type of vector returned. Default "index" returns indices, "names" returns predictor names.

Value

Integer vector of indices of filtered parameters (type = "index") or character vector of names (type = "names") of filtered parameters.

lm_filter	<i>Linear model filter</i>
-----------	----------------------------

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 = NULL,
  rsq_cutoff = NULL,
  rsq_method = "pearson",
  type = c("index", "names", "full")
)
```

Arguments

<code>y</code>	Numeric or integer response vector
<code>x</code>	Matrix of predictors. If <code>x</code> is a <code>data.frame</code> 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 <code>x</code> into a design (model) matrix first.
<code>force_vars</code>	Vector of column names <code>x</code> which are incorporated into the linear model.
<code>nfilter</code>	Number of predictors to return. If <code>NULL</code> all predictors with p-values $< p_cutoff$ are returned.
<code>p_cutoff</code>	p-value cut-off. P-values are calculated by t-statistic on the estimated coefficient for the predictor being tested.
<code>rsq_cutoff</code>	r^2 cutoff for removing predictors due to collinearity. Default <code>NULL</code> 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() .
<code>rsq_method</code>	character string indicating which correlation coefficient is to be computed. One of "pearson" (default), "kendall", or "spearman". See collinear() .
<code>type</code>	Type of vector returned. Default "index" returns indices, "names" returns predictor names, "full" returns a matrix of p values.

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 values, sigma, the residual standard error (see [summary.lm](#)), t-statistic and p-values for the tested predictor is returned.

model.hsstan	<i>hsstan model for cross-validation</i>
--------------	--

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 `hsstan` 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

<code>y</code>	Response vector. For classification this should be a factor.
<code>x</code>	Matrix of predictors
<code>unpenalized</code>	Vector of column names <code>x</code> which are always retained into the model (i.e. not penalized). Default <code>NULL</code> 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 <code>filter_options\$force_vars</code> argument. Filters currently implementing this option are the <code>partial_ttest_filter</code> for binary outcomes and the <code>lm_filter</code> for continuous outcomes.
<code>...</code>	Optional arguments passed to <code>hsstan</code>

Value

An object of class `hsstan`

Author(s)

Athina Spiliopoulou

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.

# 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
oldopt <- options(mc.cores = 2)
res.cv.hsstan <- outercv(y = dt$outcome.cont, x = dt[, c(uvars, pvars)],
                        model = "model.hsstan",
                        filterFUN = lm_filter,
                        filter_options = list(force_vars = uvars,
                                             nfilter = 2,
                                             p_cutoff = NULL,
                                             rsq_cutoff = 0.9),
                        n_outer_folds = 3, chains = 2,
                        unpenalized = uvars, warmup = 100, iter = 200)
# view prediction performance based on testing folds
res.cv.hsstan$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)

# 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,
  outer_method = c("cv", "LOOCV"),
  n_outer_folds = 10,
  n_inner_folds = 10,
  outer_folds = NULL,
  pass_outer_folds = FALSE,
  alphaSet = seq(0, 1, 0.1),
  min_lse = 0,
  keep = TRUE,
  outer_train_predict = FALSE,
  weights = NULL,
  penalty.factor = rep(1, ncol(x)),
  cv.cores = 1,
  finalCV = TRUE,
  na.option = "omit",
  ...
)
```

Arguments

y	Response vector
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 cv.glmnet and 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 character vector with names of filtered predictors.

<code>filter_options</code>	List of additional arguments passed to the filter function specified by <code>filterFUN</code> .
<code>balance</code>	Specifies method for dealing with imbalanced class data. Current options are "randomsample" or "smote". See randomsample() and smote()
<code>balance_options</code>	List of additional arguments passed to the balancing function
<code>outer_method</code>	String of either "cv" or "LOOCV" specifying whether to do k-fold CV or leave one out CV (LOOCV) for the outer folds
<code>n_outer_folds</code>	Number of outer CV folds
<code>n_inner_folds</code>	Number of inner CV folds
<code>outer_folds</code>	Optional list containing indices of test folds for outer CV. If supplied, <code>n_outer_folds</code> is ignored.
<code>pass_outer_folds</code>	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 <code>n_outer_folds</code> and <code>n_inner_folds</code> are the same and no balancing is applied.
<code>alphaSet</code>	Vector of alphas to be tuned
<code>min_1se</code>	Value from 0 to 1 specifying choice of optimal lambda from 0= λ_{\min} to 1= λ_{1se}
<code>keep</code>	Logical indicating whether inner CV predictions are retained for calculating left-out inner CV fold accuracy etc. See argument <code>keep</code> in cv.glmnet .
<code>outer_train_predict</code>	Logical whether to save predictions on outer training folds to calculate performance on outer training folds.
<code>weights</code>	Weights applied to each sample. Note <code>weights</code> and <code>balance</code> cannot be used at the same time. Weights are only applied in <code>glmnet</code> and not in filters.
<code>penalty.factor</code>	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
<code>cv.cores</code>	Number of cores for parallel processing of the outer loops. NOTE: this uses <code>parallel::mclapply</code> on unix/mac and <code>parallel::parLapply</code> on windows.
<code>finalCV</code>	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.
<code>na.option</code>	Character value specifying how NAs are dealt with. "omit" (the default) is equivalent to <code>na.action = na.omit</code> . "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).
<code>...</code>	Optional arguments passed to 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,
outer_method = c("cv", "LOOCV"),
n_outer_folds = 10,
outer_folds = NULL,
cv.cores = 1,
na.option = "pass",
...
)

```

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. Must return a character vector with names of filtered predictors. Not available if outer_cv 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 outer_cv is called with a formula. See randomsample() and smote()
balance_options	List of additional arguments passed to the balancing 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 <code>parallel::mclapply</code> on unix/mac and <code>parallel::parLapply</code> on windows.
na.option	Character value specifying how NAs are dealt with. "omit" is equivalent to <code>na.action = na.omit</code> . "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).
...	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()`.

Value

An object with S3 class "nestcv.SuperLearner"

<code>call</code>	the matched call
<code>output</code>	Predictions on the left-out outer folds
<code>outer_result</code>	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.
<code>dimx</code>	vector of number of observations and number of predictors
<code>y</code>	original response vector
<code>yfinal</code>	final response vector (post-balancing)
<code>outer_folds</code>	List of indices of outer test folds
<code>final_fit</code>	Final fitted model on whole data
<code>final_vars</code>	Column names of filtered predictors entering final model
<code>summary_vars</code>	Summary statistics of filtered predictors
<code>roc</code>	ROC AUC for binary classification where available.
<code>summary</code>	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,
  outer_method = c("cv", "LOOCV"),
  n_outer_folds = 10,
  outer_folds = NULL,
  pass_outer_folds = FALSE,
  cv.cores = 1,
  metric = ifelse(is.factor(y), "logLoss", "RMSE"),
  trControl = NULL,
  tuneGrid = NULL,
  savePredictions = "final",
  outer_train_predict = FALSE,
  finalCV = TRUE,
  na.option = "pass",
  ...
)

```

Arguments

<code>y</code>	Response vector. For classification this should be a factor.
<code>x</code>	Matrix or dataframe of predictors
<code>method</code>	String specifying which model to use. See caret::train() for details.
<code>filterFUN</code>	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.
<code>filter_options</code>	List of additional arguments passed to the filter function specified by <code>filterFUN</code> .
<code>weights</code>	Weights applied to each sample for models which can use weights. Note <code>weights</code> and <code>balance</code> cannot be used at the same time. Weights are not applied in filters.
<code>balance</code>	Specifies method for dealing with imbalanced class data. Current options are "randomsample" or "smote". See randomsample() and smote()
<code>balance_options</code>	List of additional arguments passed to the balancing function
<code>outer_method</code>	String of either "cv" or "LOOCV" specifying whether to do k-fold CV or leave one out CV (LOOCV) for the outer folds
<code>n_outer_folds</code>	Number of outer CV folds
<code>outer_folds</code>	Optional list containing indices of test folds for outer CV. If supplied, <code>n_outer_folds</code> is ignored.

<code>pass_outer_folds</code>	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 <code>n_outer_folds</code> and the number of inner CV folds specified in <code>trControl</code> are the same and that no balancing is applied.
<code>cv.cores</code>	Number of cores for parallel processing of the outer loops. NOTE: this uses <code>parallel::mclapply</code> on unix/mac and <code>parallel::parLapply</code> on windows.
<code>metric</code>	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 <code>caret</code> which uses "Accuracy" for classification. See details.
<code>trControl</code>	A list of values generated by the <code>caret</code> function <code>caret::trainControl()</code> . This defines how inner CV training through <code>caret</code> is performed. Default for the inner loop is 10-fold CV. See http://topepo.github.io/caret/using-your-own-model-in-train.html .
<code>tuneGrid</code>	Data frame of tuning values, see <code>caret::train()</code> .
<code>savePredictions</code>	Indicates whether hold-out predictions for each inner CV fold should be saved for ROC curves, accuracy etc see <code>caret::trainControl</code> . Default is "final" to capture predictions for inner CV ROC.
<code>outer_train_predict</code>	Logical whether to save predictions on outer training folds to calculate performance on outer training folds.
<code>finalCV</code>	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.
<code>na.option</code>	Character value specifying how NAs are dealt with. "omit" is equivalent to <code>na.action = na.omit</code> . "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).
<code>...</code>	Arguments passed to <code>caret::train()</code>

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` on unix/mac and `parallel::parLapply` on windows.

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`.

For classification, `metric` defaults to using 'logLoss' with the `trControl` arguments `classProbs = TRUE`, `summaryFunction` 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"

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

```

 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,
  outer_method = c("cv", "LOOCV"),
  n_outer_folds = 10,
  outer_folds = NULL,
  cv.cores = 1,
  predict_type = "prob",
  outer_train_predict = FALSE,
  na.option = "pass",
  returnList = FALSE,
  ...
)

## S3 method for class 'formula'
outercv(
  formula,
  data,
  model,
  outer_method = c("cv", "LOOCV"),
  n_outer_folds = 10,
  outer_folds = NULL,
  cv.cores = 1,
  predict_type = "prob",
  outer_train_predict = FALSE,
  ...,
  na.action = na.fail
)

```

Arguments

<code>y</code>	Response vector
<code>...</code>	Optional arguments passed to the function specified by <code>model</code> .
<code>x</code>	Matrix or dataframe of predictors
<code>model</code>	Character value or function of the model to be fitted.
<code>filterFUN</code>	Filter function, e.g. ttest_filter or relieff_filter . Any function can be provided and is passed <code>y</code> and <code>x</code> . Must return a character vector with names of filtered predictors. Not available if <code>outercv</code> is called with a formula.
<code>filter_options</code>	List of additional arguments passed to the filter function specified by <code>filterFUN</code> .
<code>weights</code>	Weights applied to each sample for models which can use weights. Note <code>weights</code> and <code>balance</code> cannot be used at the same time. Weights are not applied in filters.

<code>balance</code>	Specifies method for dealing with imbalanced class data. Current options are "randomsample" or "smote". Not available if <code>outercv</code> is called with a formula. See randomsample() and smote()
<code>balance_options</code>	List of additional arguments passed to the balancing function
<code>outer_method</code>	String of either "cv" or "LOOCV" specifying whether to do k-fold CV or leave one out CV (LOOCV) for the outer folds
<code>n_outer_folds</code>	Number of outer CV folds
<code>outer_folds</code>	Optional list containing indices of test folds for outer CV. If supplied, <code>n_outer_folds</code> is ignored.
<code>cv.cores</code>	Number of cores for parallel processing of the outer loops. NOTE: this uses <code>parallel::mclapply</code> on unix/mac and <code>parallel::parLapply</code> on windows.
<code>predict_type</code>	Only used with binary classification. Calculation of ROC AUC requires predicted class probabilities from fitted models. Most model functions use syntax of the form <code>predict(..., type = "prob")</code> . However, some models require a different type to be specified, which can be passed to <code>predict()</code> via <code>predict_type</code> .
<code>outer_train_predict</code>	Logical whether to save predictions on outer training folds to calculate performance on outer training folds.
<code>na.option</code>	Character value specifying how NAs are dealt with. "omit" is equivalent to <code>na.action = na.omit</code> . "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).
<code>returnList</code>	Logical whether to return list of results after main outer CV loop without concatenating results. Useful for debugging.
<code>formula</code>	A formula describing the model to be fitted
<code>data</code>	A matrix or data frame containing variables in the model.
<code>na.action</code>	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 <code>na.omit</code> , 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"

<code>call</code>	the matched call
<code>output</code>	Predictions on the left-out outer folds
<code>outer_result</code>	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.
<code>dimx</code>	vector of number of observations and number of predictors
<code>outer_folds</code>	List of indices of outer test folds
<code>final_fit</code>	Final fitted model on whole data
<code>final_vars</code>	Column names of filtered predictors entering final model
<code>summary_vars</code>	Summary statistics of filtered predictors
<code>roc</code>	ROC AUC for binary classification where available.
<code>summary</code>	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))
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](#). 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 <code>hcl.pals()</code>) 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_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 train
error.col	Colour of error bars
...	Other arguments passed to plot()

Value

No return value

`plot_lambdas`*Plot cross-validated glmnet lambda parameter 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

<code>x</code>	Fitted "nestcv.glmnet" object
<code>scheme</code>	colour scheme
<code>palette</code>	palette name (one of <code>hcl.pals()</code>) which is passed to hcl.colors
<code>showLegend</code>	Either a keyword to position the legend or NULL to hide the legend.
<code>...</code>	other arguments passed to plot. Use <code>type = 'p'</code> 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 of feature values containing only features values from the training data. The rows must match rows in shap.
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 of feature values containing only features values from the training data. The rows must match rows in shap.
cex	Scaling for adjusting point spacing. See <code>ggbeeswarm::geom_beeswarm()</code> .
corral	String specifying method used to corral points. See <code>ggbeeswarm::geom_beeswarm()</code> .
corral.width	Numeric specifying width of corral, passed to <code>geom_beeswarm</code>
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 <code>ggbeeswarm::geom_beeswarm()</code>

Value

A `ggplot2` plot

plot_varImp	<i>Variable importance plot</i>
-------------	---------------------------------

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_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,
  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
breaks	Vector of continuous breaks for legend colour/size

percent	Logical for <code>nestcv.glmnet</code> objects only, whether to scale coefficients to percentage of the largest coefficient in each model. If set to <code>FALSE</code> , model coefficients are shown and direction is ignored.
level	For multinomial <code>nestcv.glmnet</code> 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 <code>var_stability()</code> .

Value

A `ggplot2` plot

See Also

[var_stability\(\)](#)

predict.hsstan	<i>Predict from hsstan model fitted within cross-validation</i>
----------------	---

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 <code>hsstan</code> .
newdata	Optional data frame containing the variables to use to predict. If <code>NULL</code> (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 <code>NULL</code> will return a class with the highest probability for each sample. If set to <code>probs</code> , it will return the probabilities for outcome = 0 and for outcome = 1 for each sample.
...	Optional arguments passed to <code>hsstan::posterior_predict</code>

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"], ...)
```

Arguments

<code>object</code>	Fitted <code>nestcv.glmnet</code> object
<code>newdata</code>	New data to predict outcome on
<code>s</code>	Value of lambda for glmnet prediction
<code>...</code>	Other arguments passed to <code>predict.glmnet</code> .

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 and R² for regression. Multi-class AUC is returned for multinomial classification.

Usage

```
predSummary(output)
```

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

Details

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

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 and R² is returned.

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)
```

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	<i>Oversampling and undersampling</i>
--------------	---------------------------------------

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
```



```

                                "Nested balancing"),
col = c("green", "blue", "red", "black"), lty=1, lwd=2)

```

ranger_filter	<i>Random forest ranger filter</i>
---------------	------------------------------------

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 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	<i>ReliefF filter</i>
----------------	-----------------------

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 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\(\)](#)

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 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.

smote	<i>SMOTE</i>
-------	--------------

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.

summary_vars	<i>Summarise variables</i>
--------------	----------------------------

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	<i>Supervised PCA plot</i>
---------------	----------------------------

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	<i>Outer training fold predictions</i>
-------------	--

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	<i>Build ROC curve from outer CV training folds</i>
-----------	---

Description

Build ROC (receiver operating characteristic) curve from outer training folds. Object can be plotted using `plot()` or passed to functions `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	<i>Summarise performance on outer training folds</i>
---------------	--

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

t-test filter

Description

Simple univariate filter using t-test using the Rfast package for speed. Can be applied to all or a subset of predictors.

Usage

```

ttest_filter(
  y,
  x,
  force_vars = NULL,
  nfilter = NULL,
  p_cutoff = 0.05,
  rsq_cutoff = NULL,
  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 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.
...	optional arguments, e.g. rsq_method: see collinear() .

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.

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
```

var_direction	<i>Variable directionality</i>
---------------	--------------------------------

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

Value

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

var_stability	<i>Variable stability</i>
---------------	---------------------------

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 'nestedcv.glmnet'
var_stability(x, percent = TRUE, level = 1, sort = TRUE, ...)

## S3 method for class 'nestedcv.train'
var_stability(x, sort = TRUE, ...)
```

Arguments

x	a <code>nestedcv.glmnet</code> or <code>nestedcv.train</code> fitted object
...	Optional arguments for compatibility
percent	Logical for <code>nestedcv.glmnet</code> objects only, whether to scale coefficients to percentage of the largest coefficient in each model
level	For multinomial <code>nestedcv.glmnet</code> 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

Dataframe containing mean, sd, sem of variable importance and frequency by which each variable is selected in outer folds.

See Also

[cv_coef\(\)](#) [cv_varImp\(\)](#) [pred_train\(\)](#)

weight	<i>Calculate weights for class imbalance</i>
--------	--

Description

Calculate weights for class imbalance

Usage

```
weight(y)
```

Arguments

y Response vector

Value

Vector of weights

wilcoxon_filter	<i>Wilcoxon test filter</i>
-----------------	-----------------------------

Description

Simple univariate filter using Wilcoxon (Mann-Whitney) test using the matrixTests package.

Usage

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

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 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 Wilcoxon test. If 2 or more predictors are collinear, the first ranked predictor by Wilcoxon test 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.
exact	Logical whether exact or approximate p-value is calculated. Default is FALSE for speed.
...	Further arguments passed to matrixTests::row_wilcoxon_twosample

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 [matrixTests::row_wilcoxon_twosample](#) is returned.

Index

anova_filter, [3](#), [11](#)
anova_filter(), [6](#)
auc(), [17](#), [55](#)

barplot_var_stability, [4](#)
boot_anova (boot_ttest), [6](#)
boot_correl (boot_ttest), [6](#)
boot_filter, [5](#)
boot_filter(), [6](#)
boot_lm (boot_ttest), [6](#)
boot_ttest, [6](#)
boot_ttest(), [6](#)
boot_wilcoxon (boot_ttest), [6](#)
Boruta::Boruta(), [7](#)
boruta_filter, [7](#)
boxplot, [8](#)
boxplot_model, [8](#)

caret::modelLookup(), [35](#)
caret::train, [39](#)
caret::train(), [30](#), [31](#)
caret::trainControl, [31](#)
caret::trainControl(), [31](#), [32](#)
caret::varImp(), [14](#), [59](#)
class_balance, [8](#)
coef.cva.glmnet, [9](#)
coef.glmnet, [9](#), [15](#)
coef.nestcv.glmnet, [9](#)
collinear, [10](#)
collinear(), [3](#), [21](#), [57](#), [61](#)
combo_filter, [10](#)
cor, [11](#)
cor(), [10](#)
cor.test, [11](#)
CORElearn::attrEval, [11](#), [51](#)
CORElearn::attrEval(), [51](#)
correl_filter, [12](#)
correl_filter(), [6](#)
correls, [12](#)
correls2, [11](#)

cv.glmnet, [13](#), [15](#), [24](#), [25](#)
cv_coef, [14](#)
cv_coef(), [15](#), [60](#)
cv_varImp, [14](#)
cv_varImp(), [14](#), [60](#)
cva.glmnet, [13](#)
cva.glmnet(), [9](#)

fastshap::explain(), [47](#)

glmnet, [13](#), [16](#), [24](#), [25](#), [37](#)
glmnet(), [16](#)
glmnet::glmnet, [45](#)
glmnet_coefs, [15](#)
glmnet_filter, [15](#)

hcl.colors, [8](#), [38](#), [40](#)

innervcv_preds, [17](#)
innervcv_roc, [17](#)
innervcv_summary, [19](#)

layer_filter, [20](#)
lm_filter, [21](#)
lm_filter(), [6](#)

make.names(), [29](#)
matrixTests::row_wilcoxon_twosample,
[61](#)
model.hsstan, [22](#)

nestcv.glmnet, [8](#), [24](#), [38–40](#), [45](#)
nestcv.SuperLearner, [27](#)
nestcv.train, [29](#), [33](#), [35](#), [47](#)

outervcv, [33](#)

plot, [38](#)
plot(), [39](#)
plot.cva.glmnet, [37](#)
plot_alphas, [38](#)

plot_caret, 39
plot_lambdas, 40
plot_shap_bar, 41
plot_shap_beeswarm, 41
plot_var_stability, 43
plot_varImp, 42
points, 38
pred_nestcv_glmnet, 46
pred_nestcv_glmnet_class1
(pred_nestcv_glmnet), 46
pred_nestcv_glmnet_class2
(pred_nestcv_glmnet), 46
pred_nestcv_glmnet_class3
(pred_nestcv_glmnet), 46
pred_train(pred_nestcv_glmnet), 46
pred_train(), 59, 60
pred_train_class1(pred_nestcv_glmnet),
46
pred_train_class2(pred_nestcv_glmnet),
46
pred_train_class3(pred_nestcv_glmnet),
46
predict.cv.glmnet, 15
predict.hsstan, 44
predict.nestcv.glmnet, 45
predSummary, 19, 46, 56
princomp(), 54
pROC::multiclass.roc(), 46
pROC::roc, 17, 18, 55

randomForest::importance, 52
randomForest::randomForest, 52
randsample, 48
randsample(), 25, 28, 30, 35
ranger::ranger, 50
ranger_filter, 50
relieff_filter, 11, 24, 28, 34, 51, 54
relieff_filter(), 30
rf_filter, 52
Rfast::ftests, 4
Rfast::ttests, 58

smote, 53
smote(), 25, 28, 30, 35
stats::cor.test, 11
summary.lm, 22
summary_vars, 53
SuperLearner::SuperLearner(), 28, 29
supervisedPCA, 54

train, 39
train_preds, 54
train_roc, 55
train_summary, 56
ttest_filter, 11, 24, 28, 34, 54, 57
ttest_filter(), 5, 6, 30

var_direction, 58
var_direction(), 59
var_stability, 59
var_stability(), 5, 44

weight, 60
wilcoxon_filter, 60
wilcoxon_filter(), 6