

# Package ‘nestedcv’

September 10, 2022

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

**Version** 0.3.0

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

## 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>. 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** Boruta, caret, CORElearn, data.table, ggplot2, glmnet, hsstan, matrixTests, methods, parallel, pROC, randomForest, RcppEigen, Rfast, rlang

**RoxygenNote** 7.2.1

**Suggests** mda, rmarkdown, knitr

**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** 2022-09-10 21:50:02 UTC

## R topics documented:

anova\_filter . . . . . 2

boot_filter . . . . .	4
boot_ttest . . . . .	4
boruta_filter . . . . .	5
boxplot_model . . . . .	6
class_balance . . . . .	7
coef.nestcv.glmnet . . . . .	7
collinear . . . . .	8
combo_filter . . . . .	9
correls2 . . . . .	9
correl_filter . . . . .	10
cva.glmnet . . . . .	11
glmnet_coefs . . . . .	12
glmnet_filter . . . . .	12
innercv_roc . . . . .	13
lm_filter . . . . .	15
model.hsstan . . . . .	16
nestcv.glmnet . . . . .	18
nestcv.train . . . . .	21
outercv . . . . .	24
plot.cva.glmnet . . . . .	27
plot_alphas . . . . .	29
plot_caret . . . . .	29
plot_lambdas . . . . .	30
plot_varImp . . . . .	31
predict.hsstan . . . . .	31
predict.nestcv.glmnet . . . . .	32
predSummary . . . . .	33
randomsample . . . . .	33
relieff_filter . . . . .	34
rf_filter . . . . .	35
smote . . . . .	36
summary_vars . . . . .	36
ttest_filter . . . . .	37
wilcoxon_filter . . . . .	38
<b>Index</b>	<b>40</b>

---

 anova\_filter

*ANOVA filter*


---

### 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 <sup>2</sup> 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 <a href="#">collinear()</a> .
type	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. 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
```

---

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. <a href="#">ttest_filter()</a> .
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 <a href="#">Boruta::Boruta</a>

**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, data, scheme = NULL, palette = "Dark 3", ...)
```

**Arguments**

x	Either a "nestedcv" object or a character vector of predictors to be plotted
data	matrix of predictors
scheme	colour scheme
palette	palette name (one of <code>hcl.pals()</code> ) which is passed to <a href="#">hcl.colors</a>
...	other arguments passed to <a href="#">boxplot</a> .

**Value**

No return value

**Author(s)**

Myles Lewis

**See Also**[nестcv.glmnet](#)


---

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

---

**Description**

Check class balance in training folds

**Usage**

```
class_balance(object)

## S3 method for class 'nестcv.glmnet'
class_balance(object)

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

## S3 method for class 'outercv'
class_balance(object)
```

**Arguments**

object            Object of class `nестdсv.glmnet`, `nестcv.train` or `outercv`

**Value**

Invisibly a table of the response classes in the training folds

---

coef.nестcv.glmnet	<i>Extract coefficients from nестcv.glmnet object</i>
--------------------	---

---

**Description**

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

**Usage**

```
## S3 method for class 'nестcv.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 <a href="#">coef.glmnet</a>

**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, 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
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 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

---

correl_filter	<i>Correlation filter</i>
---------------	---------------------------

---

**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**

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
method	Type of correlation, either "pearson" or "spearman".
type	Type of vector returned. Default "index" returns indices, "names" returns predictor names, "full" returns a matrix of p-values.
...	Further arguments passed to <code>correls</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" full output from `correls` is returned.

---

`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), ...)
```

**Arguments**

<code>x</code>	Matrix of predictors
<code>y</code>	Response vector
<code>nfolds</code>	Number of folds (default 10)
<code>alphaSet</code>	Sequence of alpha values to cross-validate
<code>...</code>	Other arguments passed to <a href="#">cv.glmnet</a>

**Value**

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

<code>fits</code>	List of fitted <a href="#">cv.glmnet</a> objects
<code>alphaSet</code>	Sequence of alpha values used
<code>alpha_cvm</code>	The mean cross-validated error - a vector of length <code>length(alphaSet)</code> .
<code>best_alpha</code>	Value of alpha giving lowest <code>alpha_cvm</code> .
<code>which_alpha</code>	Index of <code>alphaSet</code> with lowest <code>alpha_cvm</code>

**Author(s)**

Myles Lewis

**See Also**

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

---

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

fit	A <a href="#">cv.glmnet</a> fitted model object.
s	Value of lambda. See <a href="#">coef.glmnet</a> and <a href="#">predict.cv.glmnet</a>
...	Other arguments passed to <a href="#">coef.glmnet</a>

### 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 properties of elastic net regression using glmnet to calculate variable importance.

### Usage

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

**Arguments**

y	Response vector
x	Matrix of predictors
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 <a href="#">glmnet</a>

**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](#).

**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\_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 `auc()` etc.

**Usage**

```
innercv_roc(x, ...)

## S3 method for class 'nestcv.glmnet'
innercv_roc(x, direction = "<", ...)

## S3 method for class 'nestcv.train'
innercv_roc(x, direction = "<", ...)
```

**Arguments**

x                    Fitted nestedcv object  
 ...                  Other arguments passed to `pROC::roc`  
 direction           Set ROC directionality `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))
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")
```

---

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 = NULL,
  rsq_cutoff = NULL,
  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 < <code>p_cutoff</code> 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 <a href="#">collinear()</a> .
<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

---

*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 `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.

**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)

# 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

# load hsstan package to examine the Bayesian model
library(hsstan)
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,
  alphaSet = seq(0, 1, 0.1),
  min_lse = 0,
  keep = TRUE,
  weights = NULL,
  penalty.factor = rep(1, ncol(x)),
  cv.cores = 1,
  na.option = "omit",
  ...
)
```

**Arguments**

y	Response vector
x	Matrix of predictors. Dataframes will be coerced to a matrix.
family	Either a character string representing one of the built-in families, or else a glm() family object. Passed to <code>cv.glmnet</code> and <code>glmnet</code>
filterFUN	Filter function, e.g. <code>ttest_filter</code> or <code>relieff_filter</code> . 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.
balance	Specifies method for dealing with imbalanced class data. Current options are "randomsample" or "smote". See <code>randomsample()</code> and <code>smote()</code>

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
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.
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 <a href="#">cv.glmnet</a> .
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 <a href="#">glmnet</a>
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" (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).
...	Optional arguments passed to <a href="#">cv.glmnet</a>

### 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
final_param	Final mean best lambda and alpha from each fold

final_fit	Final fitted glmnet 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.

**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
fit2 <- nestcv.glmnet(y, x, family = "binomial", alphaSet = 1,
                    filterFUN = ttest_filter,
                    filter_options = list(nfilter = 100))
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.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,
  filterFUN = NULL,
  filter_options = NULL,
  balance = NULL,
  balance_options = NULL,
  outer_method = c("cv", "LOOCV"),
  n_outer_folds = 10,
  outer_folds = NULL,
  cv.cores = 1,
  metric = ifelse(is.factor(y), "logLoss", "RMSE"),
  trControl = NULL,
  tuneGrid = NULL,
  savePredictions = "final",
  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>filterFUN</code>	Filter function, e.g. <a href="#">ttest_filter</a> or <a href="#">relieff_filter</a> . 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.
<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 <a href="#">randomsample()</a> and <a href="#">smote()</a>

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.
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 <code>trainControl</code> . This defines how inner CV training through caret is performed. Default for the inner loop is 10-fold CV. See <a href="http://topepo.github.io/caret/using-your-own-model-in-train.html">http://topepo.github.io/caret/using-your-own-model-in-train.html</a> .
tuneGrid	Data frame of tuning values, see <code>caret::train</code> .
savePredictions	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.
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).
...	Arguments passed to <code>caret::train</code> including method

## Details

Parallelisation is performed on the outer folds using `mclapply`. For classification `metric` defaults to using 'logLoss' with the `trControl` arguments `classProbs = TRUE`, `summaryFunction = mnLogLoss`, rather than 'Accuracy' which is the default classification metric in caret. See `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.

dimx	dimensions of x
outer_folds	List of indices of outer test folds
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)

```

---

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

## 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",
  ...,
  na.action = na.fail
)

```

## Arguments

y	Response vector
...	Optional arguments passed to the function specified by model.
x	Matrix or dataframe of predictors
model	Model function to be fitted.
filterFUN	Filter function, e.g. <a href="#">ttest_filter</a> or <a href="#">relieff_filter</a> . Any function can be provided and is passed y and x. Must return a character vector with names of filtered predictors. Not available if outercv is called with a formula.
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". Not available if outercv is called with a formula. See <a href="#">randomsample()</a> and <a href="#">smote()</a>
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.
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 <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> .
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).
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 <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  $y \sim ..$

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()`.

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 <a href="#">points</a>
scheme	Colour scheme. Overrides the palette argument.
palette	Palette name (one of <code>hcl.pals()</code> ) which is passed to <a href="#">hcl.colors</a>
showLegend	Either a keyword to position the legend or NULL to hide the legend.
...	Other arguments passed to <a href="#">plot</a> . 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_alphas	<i>Plot cross-validated glmnet alpha</i>
-------------	--

---

**Description**

Plot of cross-validated glmnet alpha parameter against deviance.

**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	<i>Plot caret tuning</i>
------------	--------------------------

---

**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 <a href="#">train</a>
error.col	Colour of error bars
...	Other arguments passed to <a href="#">plot()</a>

**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 <code>hcl.pals()</code> ) which is passed to <a href="#">hcl.colors</a>
showLegend	Either a keyword to position the legend or NULL to hide the legend.
...	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_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 'nestedcv.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

---

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 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"], ...)
```

**Arguments**

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

**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: accuracy and balanced accuracy for classification; ROC AUC for binary classification; RMSE for regression.

**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

**Value**

Vector containing accuracy and balanced accuracy for classification, ROC AUC for binary classification, RMSE for regression.

---

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  $y$

---

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**

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

**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**

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

**Details**

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

---

ttest_filter	<i>t-test filter</i>
--------------	----------------------

---

**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 <sup>2</sup> 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 <a href="#">collinear()</a> .
type	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 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
```

---

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,
  type = c("index", "names", "full"),
  exact = FALSE,
  ...
)
```

**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

rsq_cutoff	r <sup>2</sup> 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 <a href="#">collinear()</a> .
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 <a href="#">matrixTests::row_wilcoxon_twosample</a>

**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, [2](#), [9](#)  
anova\_filter(), [4](#), [5](#)  
auc(), [13](#)

boot\_anova (boot\_ttest), [4](#)  
boot\_correl (boot\_ttest), [4](#)  
boot\_filter, [4](#)  
boot\_filter(), [5](#)  
boot\_lm (boot\_ttest), [4](#)  
boot\_ttest, [4](#)  
boot\_ttest(), [4](#)  
boot\_wilcoxon (boot\_ttest), [4](#)  
Boruta: :Boruta, [6](#)  
boruta\_filter, [5](#)  
boxplot, [6](#)  
boxplot\_model, [6](#)

caret::modelLookup(), [26](#)  
caret::train, [22](#), [29](#)  
caret::trainControl, [22](#)  
class\_balance, [7](#)  
coef.glmnet, [8](#), [12](#)  
coef.nestcv.glmnet, [7](#)  
collinear, [8](#)  
collinear(), [3](#), [15](#), [37](#), [39](#)  
combo\_filter, [9](#)  
cor, [9](#)  
cor.test, [10](#)  
CORElearn::attrEval, [9](#), [34](#)  
correl\_filter, [10](#)  
correl\_filter(), [4](#), [5](#)  
correls, [10](#)  
correls2, [9](#)  
cv.glmnet, [11](#), [12](#), [18](#), [19](#)  
cva.glmnet, [11](#)

glmnet, [11](#), [13](#), [18](#), [19](#), [27](#)  
glmnet::glmnet, [32](#)  
glmnet\_coefs, [12](#)  
glmnet\_filter, [12](#)

hcl.colors, [6](#), [28](#), [30](#)

importance, [35](#)  
innercv\_roc, [13](#)

lm\_filter, [15](#)  
lm\_filter(), [4](#), [5](#)

matrixTests::row\_wilcoxon\_twosample,  
[39](#)

model.hsstan, [16](#)

nestcv.glmnet, [7](#), [18](#), [28–30](#), [32](#)  
nestcv.train, [21](#), [24](#), [26](#)

outercv, [24](#)

plot, [28](#)  
plot(), [29](#)  
plot.cva.glmnet, [27](#)  
plot\_alphas, [29](#)  
plot\_caret, [29](#)  
plot\_lambdas, [30](#)  
plot\_varImp, [31](#)  
points, [28](#)  
predict.cv.glmnet, [12](#)  
predict.hsstan, [31](#)  
predict.nestcv.glmnet, [32](#)  
predSummary, [33](#)  
pROC::roc, [14](#)

randomForest, [35](#)  
randomsample, [33](#)  
randomsample(), [18](#), [21](#), [25](#)  
relieff\_filter, [9](#), [18](#), [21](#), [25](#), [34](#)  
rf\_filter, [35](#)  
Rfast::ftests, [3](#)  
Rfast::ttests, [37](#)

smote, [36](#)  
smote(), [18](#), [21](#), [25](#)



stats::cor.test, [9](#)  
summary.lm, [16](#)  
summary\_vars, [36](#)

train, [29](#)  
trainControl, [22](#)  
ttest\_filter, [9](#), [18](#), [21](#), [25](#), [37](#)  
ttest\_filter(), [4](#), [5](#)

wilcoxon\_filter, [38](#)  
wilcoxon\_filter(), [4](#), [5](#)