Package 'nestedcv'

October 23, 2022

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

Version 0.4.0

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

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

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

Description
Implements nested k*l-fold cross-validation for lasso and elastic-net regularised linear models via the 'glmnet' package and other machine learning models via the 'caret' package. 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, doParallel, foreach, ggplot2, glmnet, hsstan, matrixStats, matrixTests, methods, parallel, pROC, randomForest, RcppEigen, Rfast, rlang, SuperLearner

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 2022-10-23 16:12:36 UTC

R topics documented:

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^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.

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::ftests 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. `ttest_filter()`.
`B`	Number of times to bootstrap
`nfilter`	Number of predictors to return
`type`	Type of vector returned. Default "index" returns indices, "full" returns full output.
`...`	Optional arguments passed to the function specified by `filterFUN`

Value

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

See Also

`boot_ttest()`

`boot_ttest` *Bootstrap univariate filters*

Description

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

Usage

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

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

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

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

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

Arguments

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

Value

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

See Also

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

`boruta_filter` *Boruta filter*

Description

Filter using Boruta algorithm.

Usage

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

Arguments

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

Details

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

Value

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

boxplot_model *Boxplot model predictors*

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 `hcl.pals()`) which is passed to hcl.colors
...	other arguments passed to boxplot.

Value

No return value

Author(s)

Myles Lewis

See Also

nestcv.glmnet

class_balance *Check class balance in training folds*

Description

Check class balance in training folds

Usage

```
class_balance(object)

## Default S3 method:
class_balance(object)

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

Arguments

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

Value

Invisibly a table of the response classes in the training folds

---

`coef.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 *Filter to reduce collinearity in predictors*

Description

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

Usage

```
collinear(x, rsq_cutoff = 0.9, 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` *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

`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 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` *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

`x`	Matrix of predictors
`y`	Response vector
`nfolds`	Number of folds (default 10)
`alphaSet`	Sequence of alpha values to cross-validate
`...`	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

glmnet_coefs *glmnet coefficients*

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 cv.glmnet fitted model object.
`s`	Value of lambda. See coef.glmnet and predict.cv.glmnet
`...`	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 *glmnet filter*

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 glmnet

Details

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

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")
```

`layer_filter` *Multilayer filter*

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` *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

`y`	Numeric or integer response vector
`x`	Matrix of predictors. If `x` is a data.frame it will be turned into a matrix. But note that factors will be reduced to numeric values, but a full design matrix is not generated, so if factors have 3 or more levels, it is recommended to convert `x` into a design (model) matrix first.
`force_vars`	Vector of column names `x` which are incorporated into the linear model.
`nfilter`	Number of predictors to return. If `NULL` all predictors with p-values < `p_cutoff` are returned.
`p_cutoff`	p-value cut-off. P-values are calculated by t-statistic on the estimated coefficient for the predictor being tested.
`rsq_cutoff`	r^2 cutoff for removing predictors due to collinearity. Default `NULL` means no collinearity filtering. Predictors are ranked based on AIC from a linear model. If 2 or more predictors are collinear, the first ranked predictor by AIC is retained, while the other collinear predictors are removed. See `collinear()`.
`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 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 `hsstsan` model. Predictions on the testing folds are brought back together and error estimation/ accuracy determined. The default is 10-fold CV. The function is implemented within the `nestedcv` package. The `hsstan` models do not require tuning of meta-parameters and therefore only a single CV procedure is needed to evaluate performance. This is implemented using the `outer` CV procedure in the `nestedcv` package.

Usage

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

Arguments

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

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_1se = 0,
  keep = TRUE,
  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.
`filter_options`	List of additional arguments passed to the filter function specified by `filterFUN`.

balance	Specifies method for dealing with imbalanced class data. Current options are "randomsample" or "smote". See randomsample() and smote()
balance_options	List of additional arguments passed to the balancing function
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 cv.glmnet.
weights	Weights applied to each sample. Note weights and balance cannot be used at the same time. Weights are only applied in glmnet and not in filters.
penalty.factor	Separate penalty factors can be applied to each coefficient. Can be 0 for some variables, which implies no shrinkage, and that variable is always included in the model. Default is 1 for all variables. See glmnet
cv.cores	Number of cores for parallel processing of the outer loops. NOTE: this uses parallel::mclapply on unix/mac and parallel::parLapply on windows.
finalCV	Logical whether to perform one last round of CV on the whole dataset to determine the final model parameters. If set to FALSE, the median of hyperparameters from outer CV folds are used for the final model. Performance metrics are independent of this last step.
na.option	Character value specifying how NAs are dealt with. "omit" (the default) is equivalent to na.action = na.omit. "omitcol" removes cases if there are NA in 'y', but columns (predictors) containing NA are removed from 'x' to preserve cases. Any other value means that NA are ignored (a message is given).
...	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
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
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.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 outercv is called with a formula.
filter_options	List of additional arguments passed to the filter function specified by filterFUN.
weights	Weights applied to each sample for models which can use weights. Note weights and balance cannot be used at the same time. Weights are not applied in filters.
balance	Specifies method for dealing with imbalanced class data. Current options are "randomsample" or "smote". Not available if outercv is called with a formula. See randomsample() and smote()
balance_options	List of additional arguments passed to the balancing function
outer_method	String of either "cv" or "LOOCV" specifying whether to do k-fold CV or leave one out CV (LOOCV) for the outer folds
n_outer_folds	Number of outer CV folds
outer_folds	Optional list containing indices of test folds for outer CV. If supplied, n_outer_folds is ignored.
cv.cores	Number of cores for parallel processing of the outer loops. NOTE: this uses parallel::mclapply on unix/mac and parallel::parLapply on windows.
na.option	Character value specifying how NAs are dealt with. "omit" is equivalent to na.action = na.omit. "omitcol" removes cases if there are NA in 'y', but columns (predictors) containing NA are removed from 'x' to preserve cases. Any other value means that NA are ignored (a message is given).
...	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"

call	the matched call
output	Predictions on the left-out outer folds
outer_result	List object of results from each outer fold containing predictions on left-out outer folds, model result and number of filtered predictors at each fold.
dimx	vector of number of observations and number of predictors
y	original response vector
yfinal	final response vector (post-balancing)
outer_folds	List of indices of outer test folds
final_fit	Final fitted model on whole data

final_vars	Column names of filtered predictors entering final model
summary_vars	Summary statistics of filtered predictors
roc	ROC AUC for binary classification where available.
summary	Overall performance summary. Accuracy and balanced accuracy for classification. ROC AUC for binary classification. RMSE for regression.

Note

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

See Also

SuperLearner::SuperLearner()

---

nestcv.train *Nested cross-validation for caret*

Description

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

Usage

```
nestcv.train(
  y,
  x,
  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,
  metric = ifelse(is.factor(y), "logLoss", "RMSE"),
  trControl = NULL,
  tuneGrid = NULL,
  savePredictions = "final",
  finalCV = TRUE,
  na.option = "pass",
  ...
)
```

Arguments

y	Response vector. For classification this should be a factor.
x	Matrix or dataframe 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.
weights	Weights applied to each sample for models which can use weights. Note weights and balance cannot be used at the same time. Weights are not applied in filters.
balance	Specifies method for dealing with imbalanced class data. Current options are "randomsample" or "smote". See randomsample() and smote()
balance_options	List of additional arguments passed to the balancing function
outer_method	String of either "cv" or "LOOCV" specifying whether to do k-fold CV or leave one out CV (LOOCV) for the outer folds
n_outer_folds	Number of outer CV folds
outer_folds	Optional list containing indices of test folds for outer CV. If supplied, n_outer_folds is ignored.
cv.cores	Number of cores for parallel processing of the outer loops. NOTE: this uses parallel::mclapply on unix/mac and parallel::parLapply on windows.
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 trainControl. This defines how inner CV training through caret is performed. Default for the inner loop is 10-fold CV. See http://topepo.github.io/caret/using-your-own-model-in-train.html.
tuneGrid	Data frame of tuning values, see caret::train.
savePredictions	Indicates whether hold-out predictions for each inner CV fold should be saved for ROC curves, accuracy etc see caret::trainControl. Default is "final" to capture predictions for inner CV ROC.
finalCV	Logical whether to perform one last round of CV on the whole dataset to determine the final model parameters. If set to FALSE, the median of the best hyperparameters from outer CV folds for continuous/ ordinal hyperparameters, or highest voted for categorical hyperparameters, are used to fit the final model. Performance metrics are independent of this last step.
na.option	Character value specifying how NAs are dealt with. "omit" is equivalent to na.action = na.omit. "omitcol" removes cases if there are NA in 'y', but columns (predictors) containing NA are removed from 'x' to preserve cases. Any other value means that NA are ignored (a message is given).
...	Arguments passed to caret::train including method

Details

Parallelisation is performed on the outer folds using `parallel::mclapply` on unix/mac and `parallel::parLapply` on windows.

We strongly recommend that you try calls to `nestcv.train` with `cv.cores=1` first. With `caret` this may flag up that specific packages are not installed or that there are problems with input variables `y` and `x` which may have to be corrected for the call to run in multicore mode.

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, summaryFunctio` 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.
`outer_folds`	List of indices of outer test folds
`dimx`	dimensions of `x`
`y`	original response vector
`yfinal`	final response vector (post-balancing)
`final_fit`	Final fitted caret model using best tune parameters
`final_vars`	Column names of filtered predictors entering final model
`summary_vars`	Summary statistics of filtered predictors
`roc`	ROC AUC for binary classification where available.
`trControl`	`caret::trainControl` object used for inner CV
`bestTunes`	best tuned parameters from each outer fold
`finalTune`	final parameters used for final model
`summary`	Overall performance summary. Accuracy and balanced accuracy for classification. ROC AUC for binary classification. RMSE for regression.

Author(s)

Myles Lewis

Examples

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

## load iris dataset and simulate a binary outcome
data(iris)
x <- iris[, 1:4]
colnames(x) <- c("marker1", "marker2", "marker3", "marker4")
x <- as.data.frame(apply(x, 2, scale))
y2 <- sigmoid(0.5 * x$marker1 + 2 * x$marker2) > runif(nrow(x))
y2 <- factor(y2, labels = c("class1", "class2"))

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

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

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

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

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

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

`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",
  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",
  ...,
  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. 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 outercv is called with a formula.
filter_options	List of additional arguments passed to the filter function specified by filterFUN.
weights	Weights applied to each sample for models which can use weights. Note weights and balance cannot be used at the same time. Weights are not applied in filters.
balance	Specifies method for dealing with imbalanced class data. Current options are "randomsample" or "smote". Not available if outercv is called with a formula. See randomsample() and smote()
balance_options	List of additional arguments passed to the balancing function
outer_method	String of either "cv" or "LOOCV" specifying whether to do k-fold CV or leave one out CV (LOOCV) for the outer folds
n_outer_folds	Number of outer CV folds
outer_folds	Optional list containing indices of test folds for outer CV. If supplied, n_outer_folds is ignored.
cv.cores	Number of cores for parallel processing of the outer loops. NOTE: this uses parallel::mclapply on unix/mac and parallel::parLapply on windows.
predict_type	Only used with binary classification. Calculation of ROC AUC requires predicted class probabilities from fitted models. Most model functions use syntax of the form predict(..., type = "prob"). However, some models require a different type to be specified, which can be passed to predict() via predict_type.
na.option	Character value specifying how NAs are dealt with. "omit" is equivalent to na.action = na.omit. "omitcol" removes cases if there are NA in 'y', but columns (predictors) containing NA are removed from 'x' to preserve cases. Any other value means that NA are ignored (a message is given).
returnList	Logical whether to return list of results after main outer CV loop without concatenating results. Useful for debugging.
formula	A formula describing the model to be fitted
data	A matrix or data frame containing variables in the model.
na.action	Formula S3 method only: a function to specify the action to be taken if NAs are found. The default action is for the procedure to fail. An alternative is na.omit, which leads to rejection of cases with missing values on any required variable. (NOTE: If given, this argument must be named.)

Details

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

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

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

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

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

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

Value

An object with S3 class "outercv"

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

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 `hcl.pals()`) which is passed to hcl.colors
showLegend	Either a keyword to position the legend or `NULL` to hide the legend.
...	Other arguments passed to plot. Use `type = 'p'` to plot a scatter plot instead of a line plot.

Value

No return value

Author(s)

Myles Lewis

See Also

nestcv.glmnet

`plot_alphas` *Plot cross-validated glmnet alpha*

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 *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 lambdas across outer folds*

Description

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

Usage

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

Arguments

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

Value

No return value

Author(s)

Myles Lewis

See Also

nestcv.glmnet

plot_varImp *Variable importance plot*

Description

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

Usage

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

Arguments

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

Value

Returns a ggplot2 plot

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

Description

Draws from the posterior predictive distribution of the outcome.

Usage

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

Arguments

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

Value

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

Author(s)

Athina Spiliopoulou

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

Description

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

Usage

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

Arguments

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

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` *ReliefF filter*

Description

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

Usage

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

Arguments

`y`	Response vector
`x`	Matrix 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.
`mtry`	Number of predictors randomly sampled as candidates at each split. See randomForest.
`...`	Optional arguments passed to randomForest.

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 *SMOTE*

Description

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

Usage

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

Arguments

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

Value

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

References

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

summary_vars *Summarise variables*

Description

Summarise variables

Usage

```
summary_vars(x)
```

Arguments

x	Matrix or dataframe with variables in columns

Value

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

`supervisedPCA` *Supervised PCA plot*

Description

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

Usage

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

Arguments

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

Value

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

`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.

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
```

`weight` *Calculate weights for class imbalance*

Description

Calculate weights for class imbalance

Usage

```
weight(y)
```

Arguments

`y`	Response vector

Value

Vector of weights

`wilcoxon_filter` *Wilcoxon test filter*

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

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().
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