# Package 'nlpred'

July 22, 2025

**Title** Estimators of Non-Linear Cross-Validated Risks Optimized for Small Samples

**Version** 1.0.1

**Description** Methods for obtaining improved estimates of non-linear cross-validated risks are obtained using targeted minimum loss-based estimation, estimating equations, and one-step estimation (Benkeser, Petersen, van der Laan (2019), <doi:10.1080/01621459.2019.1668794>). Cross-validated area under the receiver operating characteristics curve (LeDell, Petersen, van der Laan (2015), <doi:10.1214/15-EJS1035>) and other metrics are included.

**Depends** R (>= 3.2.0), data.table

**Imports** stats, utils, SuperLearner, cvAUC, ROCR, Rdpack, bde, np, assertthat

**Suggests** knitr, rmarkdown, testthat, prettydoc, randomForest, ranger, xgboost, glmnet,

**License** MIT + file LICENSE

**Encoding** UTF-8

**VignetteBuilder** knitr, rmarkdown

**LazyData** true

**RoxygenNote** 7.0.2

**NeedsCompilation** no

**Author** David Benkeser [aut, cre]

**Maintainer** David Benkeser <benkeser@emory.edu>

**Repository** CRAN

**Date/Publication** 2020-02-23 17:30:05 UTC

## Contents

---

`.Dy` *Compute one of the terms of the efficient influence function*

---

## Description

Compute one of the terms of the efficient influence function

## Usage

```
.Dy(full_long_data, y)
```

## Arguments

| | |
|---|---|
| full_long_data | A long form data set |
| y | Which portion of the EIF to compute |

## Value

Vector of one piece of EIF evaluated at estimates in full_long_data

---

# .estim_fn *An estimating function for cvAUC*

---

## Description

An estimating function for cvAUC

## Usage

```
.estim_fn(auc = 0.5, prediction_list, gn)
```

## Arguments

| | |
|---|---|
| auc | The value of auc to find root for |
| prediction_list | Entry in prediction_list |
| gn | Marginal probability of outcome |

## Value

A numeric value of the estimating function evaluated at current auc estimate.

## .estim_fn_nested_cv *An estimating function for cvAUC with initial estimates generated via nested cross-validation*

### Description

An estimating function for cvAUC with initial estimates generated via nested cross-validation

### Usage

```
.estim_fn_nested_cv(auc = 0.5, prediction_list, folds, gn, K)
```

### Arguments

| | |
|---|---|
| auc | The value of auc to find root for |
| prediction_list | Entry in prediction_list |
| folds | Cross-validation folds |
| gn | Marginal probability of outcome |
| K | Number of CV folds |

### Value

A numeric value of the estimating function evaluated at current auc estimate.

## .get_auc *Compute the AUC given the cdf and pdf of psi*

### Description

See ?.get_psi_distribution to understand expected input format

### Usage

```
.get_auc(dist_y0, dist_y1)
```

### Arguments

| | |
|---|---|
| dist_y0 | Distribution of psi given Y = 0 |
| dist_y1 | Distribution of psi given Y = 1 |

### Value

Numeric value of AUC

## .get_cv_estim *Helper function to turn prediction_list into CV estimate of SCRNP*

### Description

Helper function to turn prediction_list into CV estimate of SCRNP

### Usage

```
.get_cv_estim(prediction_list, sens, gn, quantile_type = 8, ...)
```

### Arguments

| | |
|---|---|
| prediction_list | Properly formatted list of predictions. |
| sens | The sensitivity constraint. |
| gn | The marginal probability that Y = 1. |
| quantile_type | The type of quantile estimate to use. |
| ... | Other options (not currently used) |

## .get_density *Function to estimate density needed to evaluate standard errors.*

### Description

Function to estimate density needed to evaluate standard errors.

### Usage

```
.get_density(
  x,
  c0,
  bounded_kernel = FALSE,
  x_name = "train_pred",
  y_name = "train_y",
  nested_cv = FALSE,
  prediction_list = NULL,
  folds = NULL,
  maxDens = 1000,
  ...
)
```

### Arguments

| | |
|---|---|
| x | An entry in prediction_list. |
| c0 | The point at which the density estimate is evaluated. |
| bounded_kernel | Should a bounded kernel be used? Default is FALSE. |
| x_name | Name of variable to compute density of. |
| y_name | Name of variable to stratify density computation on. |
| nested_cv | Use nested CV to estimate density? |
| prediction_list | Properly formatted list of predictions. |
| folds | Cross-validation fold assignments. |
| maxDens | The maximum allowed value for the density. |
| ... | Other options (not currently used) |

---

## .get_nested_cv_quantile

*Helper function to get quantile for a single training fold data when nested CV is used.*

---

### Description

Helper function to get quantile for a single training fold data when nested CV is used.

### Usage

```
.get_nested_cv_quantile(x, p, prediction_list, folds, quantile_type = 8)
```

### Arguments

| | |
|---|---|
| x | An entry in prediction_list. |
| p | The quantile to get. |
| prediction_list | Properly formatted list of predictions. |
| folds | Cross-validation fold assignments. |
| quantile_type | The type of quantile estimate to use. |

## .get_one_fold *Helper function to get results for a single cross-validation fold*

### Description

Helper function to get results for a single cross-validation fold

### Usage

```
.get_one_fold(x, sens, gn, quantile_type = 8, ...)
```

### Arguments

| | |
|---|---|
| x | An entry in prediction_list. |
| sens | The sensitivity constraint. |
| gn | An estimate of the marginal probability that Y = 1. |
| quantile_type | The type of quantile estimate to use. |
| ... | Other options (not currently used) |

## .get_predictions *Worker function for fitting prediction functions (possibly in parallel)*

### Description

Worker function for fitting prediction functions (possibly in parallel)

### Usage

```
.get_predictions(
  learner,
  Y,
  X,
  K = 10,
  folds,
  parallel,
  nested_cv = FALSE,
  nested_K = K - 1
)
```

### Arguments

| | |
|---|---|
| learner | The wrapper to use |
| Y | The outcome |
| X | The predictors |
| K | The number of folds |
| folds | Vector of CV fold assignments |
| parallel | Whether to compute things in parallel using future |
| nested_cv | Is nested CV being used? |
| nested_K | How many folds of nested CV? |

### Value

A list of the result of the wrapper executed in each fold

---

## .get_psi_distribution *Compute the conditional (given Y = y) estimated distribution of psi*

---

### Description

Compute the conditional (given Y = y) estimated distribution of psi

### Usage

```
.get_psi_distribution(x, y, epsilon = 0)
```

### Arguments

| | |
|---|---|
| x | An entry in the output from .get_predictions |
| y | What value of Y to compute dist. est. |
| epsilon | A vector of estimated coefficients form tmle fluctuation submodels. |

### Value

A data.frame with the distribution of psi given Y = y with names psix (what value estimates are evaluated at), dFn (density estimates), Fn (cdf estimates)

## .get_psi_distribution_nested_cv

*Compute the conditional (given Y = y) CV-estimated distribution of psi*

### Description

Compute the conditional (given Y = y) CV-estimated distribution of psi

### Usage

```
.get_psi_distribution_nested_cv(x, y, prediction_list, folds, epsilon = 0)
```

### Arguments

| | |
|---|---|
| x | The outer validation fold withheld |
| y | What value of Y to compute dist. est. |
| prediction_list | List output from .get_predictions. |
| folds | Cross validation fold indicator. |
| epsilon | A vector of estimated coefficients form tmle fluctuation submodels. |

### Value

A data.frame with the distribution of psi given Y = y with names psix (what value estimates are evaluated at), dFn (density estimates), Fn (cdf estimates)

## .get_quantile

*Helper function to get quantile for a single training fold data when nested CV is NOT used.*

### Description

Helper function to get quantile for a single training fold data when nested CV is NOT used.

### Usage

```
.get_quantile(x, p, quantile_type = 8)
```

### Arguments

| | |
|---|---|
| x | An entry in prediction_list. |
| p | The quantile to get. |
| quantile_type | The type of quantile estimate to use. |

`.make_long_data` *Worker function to make long form data set needed for CVTMLE targeting step*

## Description

Worker function to make long form data set needed for CVTMLE targeting step

## Usage

```
.make_long_data(
  x,
  gn,
  update = FALSE,
  epsilon_0 = 0,
  epsilon_1 = 0,
  tol = 0.001
)
```

## Arguments

| | |
|---|---|
| `x` | An entry in the "predictions list" that has certain named values (see `?.get_predictions`) |
| `gn` | An estimate of the probability that `Y = 1`. |
| `update` | A boolean of whether this is called for initial construction of the long data set or as part of the targeting loop. If the former, empirical "density" estimates are used. If the latter these are derived from the targeted cdf. |
| `epsilon_0` | If `update = TRUE`, a vector of TMLE fluctuation parameter estimates used to add the CDF and PDF of Psi(X) to the data set. |
| `epsilon_1` | Same as for `epsilon_0`. |
| `tol` | A truncation level when taking logit transformations. |

## Value

A long form data list of a particular set up. Columns are named id (multiple rows per observation in validation sample), u (if Yi = 0, these are the values of psi(x) in the training sample for obs with Y = 1, if Yi = 1, these are values of psi(x) in the training sample for obs. with Y = 0), Yi (this observation's value of Y), Fn (estimated value of the cdf of psi(X) given Y = Yi in the training sample), dFn (estimated value of the density of psi(X) given Y = (1-Yi) in the training sample), psi (the value of this observations Psihat(P_n,B_n^0)), gn (estimate of marginal of Y e.g., computed in whole sample), outcome (indicator that psix <= u), logit_Fn (the cdf estimate on the logit scale, needed for offset in targeting model).

`.make_long_data_nested_cv` *Worker function to make long form data set needed for CVTMLE targeting step when nested cv is used*

## Description

Worker function to make long form data set needed for CVTMLE targeting step when nested cv is used

## Usage

```
.make_long_data_nested_cv(
  x,
  prediction_list,
  folds,
  gn,
  update = FALSE,
  epsilon_0 = 0,
  epsilon_1 = 0,
  tol = 0.001
)
```

## Arguments

| | |
|---|---|
| `x` | The outer validation fold |
| `prediction_list` | The full prediction list |
| `folds` | Vector of CV folds |
| `gn` | An estimate of the marginal dist. of Y |
| `update` | Boolean of whether this is called for initial construction of the long data set or as part of the targeting loop. If the former, cross-validated empirical "density" estimates are used. If the latter these are derived from the targeted cdf. |
| `epsilon_0` | If `update = TRUE`, a vector of TMLE fluctuation parameter estimates used to add the CDF and PDF of Psi(X) to the data set |
| `epsilon_1` | Ditto above |
| `tol` | A truncation level when taking logit transformations. |

## Value

A long form data list of a particular set up. Columns are named id (multiple per obs. in validation sample), u (if Yi = 0, these are the unique values of psi(x) in the inner validation samples for psi fit on inner training samples for obs with Y = 1, if Yi = 1, these are values of psi(x) in the inner validation samples for psi fit on inner training samples for obs. with Y = 0), Yi (this id's value of Y), Fn (cross-validation estimated value of the cdf of psi(X) given Y = Yi in the training sample),

dFn (cross-validated estimate of the density of psi(X) given Y = (1-Yi) in the training sample), psi (the value of this observations Psihat(P_n,B_n^0)), gn (estimate of marginal of Y e.g., computed in whole sample), outcome (indicator that psix <= u), logit_Fn (the cdf estimate on the logit scale, needed for offset in targeting model).

---

`.make_targeting_data` *Helper function for making data set in proper format for CVTMLE*

---

## Description

Helper function for making data set in proper format for CVTMLE

## Usage

```
.make_targeting_data(
  x,
  prediction_list,
  quantile_list,
  density_list,
  folds,
  nested_cv = FALSE,
  gn
)
```

## Arguments

| | |
|---|---|
| `x` | A numeric identifier of which entry in `prediction_list` to operate on. |
| `prediction_list` | Properly formatted list of predictions. |
| `quantile_list` | List of estimated quantile for each fold. |
| `density_list` | List of density estimates for each fold. |
| `folds` | Cross-validation fold assignments. |
| `nested_cv` | A boolean indicating whether nested CV was used in estimation. |
| `gn` | An estimate of the marginal probability that `Y = 1`. |

## .process_input *Unexported function from cvAUC package*

### Description

Unexported function from cvAUC package

### Usage

```
.process_input(
  predictions,
  labels,
  label.ordering = NULL,
  folds = NULL,
  ids = NULL,
  confidence = NULL
)
```

### Arguments

| | |
|---|---|
| predictions | A vector, matrix, list, or data frame containing the predictions. |
| labels | A vector, matrix, list, or data frame containing the true class labels. Must have the same dimensions as predictions. |
| label.ordering | The default ordering of the classes can be changed by supplying a vector containing the negative and the positive class label (negative label first, positive label second). |
| folds | If specified, this must be a vector of fold ids equal in length to predictions and labels, or a list of length V (for V-fold cross-validation) of vectors of indexes for the observations contained in each fold. The folds argument must only be specified if the predictions and labels arguments are vectors. |
| ids | Vector of ids |
| confidence | confidence interval level |

## adult *adult*

### Description

The "Adult" data set from UCI machine learning repository. Raw data have been processed and an outcome column added.

## Details

Description (copied from UCI):

Extraction was done by Barry Becker from the 1994 Census database. A set of reasonably clean records was extracted using the following conditions: ((AAGE>16) && (AGI>100) && (AFNLWGT>1)&& (HRSWK>0))

Prediction task is to determine whether a person makes over 50K a year (column `outcome`).

Listing of attributes:

>50K, <=50K

age: continuous.

workclass: Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, State-gov, Without-pay, Never-worked.

fnlwgt: continuous.

education: Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc-acdm, Assoc-voc, 9th, 7th-8th, 12th, Masters, 1st-4th, 10th, Doctorate, 5th-6th, Preschool.

education-num: continuous.

marital-status: Married-civ-spouse, Divorced, Never-married, Separated, Widowed, Married-spouse-absent, Married-AF-spouse.

occupation: Tech-support, Craft-repair, Other-service, Sales, Exec-managerial, Prof-specialty, Handlers-cleaners, Machine-op-inspct, Adm-clerical, Farming-fishing, Transport-moving, Priv-house-serv, Protective-serv, Armed-Forces.

relationship: Wife, Own-child, Husband, Not-in-family, Other-relative, Unmarried.

race: White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black.

sex: Female, Male.

capital-gain: continuous.

capital-loss: continuous.

hours-per-week: continuous.

native-country: United-States, Cambodia, England, Puerto-Rico, Canada, Germany, Outlying-US(Guam-USVI-etc), India, Japan, Greece, South, China, Cuba, Iran, Honduras, Philippines, Italy, Poland, Jamaica, Vietnam, Mexico, Portugal, Ireland, France, Dominican-Republic, Laos, Ecuador, Taiwan, Haiti, Columbia, Hungary, Guatemala, Nicaragua, Scotland, Thailand, Yugoslavia, El-Salvador, Trinadad&Tobago, Peru, Hong, Holand-Netherlands.

## Source

https://archive.ics.uci.edu/ml/datasets/Adult

## References

http://robotics.stanford.edu/~ronnyk/nbtree.pdf

bank *bank*

## Description

Bank data from UCI Machine Learning Repository. The raw bank data have been processed and an `outcome` column added.

## Details

Description (copied from UCI):

The data is related with direct marketing campaigns of a Portuguese banking institution. The marketing campaigns were based on phone calls. Often, more than one contact to the same client was required, in order to access if the product (bank term deposit) would be ('yes') or not ('no') subscribed. There are four datasets:

1) (included in `predtmle`) bank-additional-full.csv with all examples (41188) and 20 inputs, ordered by date (from May 2008 to November 2010), very close to the data analyzed in [Moro et al., 2014]

2) bank-additional.csv with 10% of the examples (4119), randomly selected from 1), and 20 inputs.

3) bank-full.csv with all examples and 17 inputs, ordered by date (older version of this dataset with less inputs).

4) bank.csv with 10% of the examples and 17 inputs, randomly selected from 3 (older version of this dataset with less inputs).

The smallest datasets are provided to test more computationally demanding machine learning algorithms (e.g., SVM). The classification goal is to predict if the client will subscribe (yes/no) a term deposit (variable y).

Attribute Information:

Input variables:

# bank client data:

1 - age (numeric)

2 - job : type of job (categorical: 'admin.','blue-collar','entrepreneur','housemaid','management','retired','self-employed','services','student','technician','unemployed','unknown')

3 - marital : marital status (categorical: 'divorced','married','single','unknown'; note: 'divorced' means divorced or widowed)

4 - education (categorical: 'basic.4y','basic.6y','basic.9y','high.school','illiterate','professional.course','university.degree','u

5 - default: has credit in default? (categorical: 'no','yes','unknown') 6 - housing: has housing loan? (categorical: 'no','yes','unknown')

7 - loan: has personal loan? (categorical: 'no','yes','unknown')

# related with the last contact of the current campaign:

8 - contact: contact communication type (categorical: 'cellular','telephone')

9 - month: last contact month of year (categorical: 'jan', 'feb', 'mar', ..., 'nov', 'dec')

10 - day_of_week: last contact day of the week (categorical: 'mon','tue','wed','thu','fri')

11 - duration: last contact duration, in seconds (numeric). Important note: this attribute highly affects the output target (e.g., if duration=0 then y='no'). Yet, the duration is not known before a call is performed. Also, after the end of the call y is obviously known. Thus, this input should only be included for benchmark purposes and should be discarded if the intention is to have a realistic predictive model.

# other attributes:

12 - campaign: number of contacts performed during this campaign and for this client (numeric, includes last contact)

13 - pdays: number of days that passed by after the client was last contacted from a previous campaign (numeric; 999 means client was not previously contacted)

14 - previous: number of contacts performed before this campaign and for this client (numeric)

15 - poutcome: outcome of the previous marketing campaign (categorical: 'failure','nonexistent','success')

# social and economic context attributes

16 - emp.var.rate: employment variation rate - quarterly indicator (numeric)

17 - cons.price.idx: consumer price index - monthly indicator (numeric)

18 - cons.conf.idx: consumer confidence index - monthly indicator (numeric)

19 - euribor3m: euribor 3 month rate - daily indicator (numeric)

20 - nr.employed: number of employees - quarterly indicator (numeric)

Output variable (desired target):

21 - y - has the client subscribed a term deposit? (binary: 'yes','no')

## Source

https://archive.ics.uci.edu/ml/datasets/Bank+Marketing

## References

S. Moro, P. Cortez and P. Rita. A Data-Driven Approach to Predict the Success of Bank Telemarketing. Decision Support Systems, Elsevier, 62:22-31, June 2014

---

`boot_auc` *Compute the bootstrap-corrected estimator of AUC.*

---

## Description

This estimator is computed by re-sampling with replacement (i.e., bootstrap sampling) from the data. The AUC is computed for the learner trained on the full data. The AUC is then computed for the learner trained on each bootstrap sample. The average difference between the full data-trained learner and the bootstrap-trained learner is computed to estimate the bias in the full-data-estimated AUC. The final estimate of AUC is given by the difference in the full-data AUC and the estimated bias.

### Usage

```
boot_auc(Y, X, B = 500, learner = "glm_wrapper", correct632 = FALSE, ...)
```

### Arguments

| | |
|---|---|
| Y | A numeric vector of outcomes, assume to equal 0 or 1. |
| X | A data.frame of variables for prediction. |
| B | The number of bootstrap samples. |
| learner | A wrapper that implements the desired method for building a prediction algorithm. See ?glm_wrapper or read the package vignette for more information on formatting learners. |
| correct632 | A boolean indicating whether to use the .632 correction. |
| ... | Other options, not currently used. |

### Value

A list with $auc as the bootstrap-corrected AUC estimate and $n_valid_boot as the number of bootstrap of bootstrap samples where learner successfully executed.

### Examples

```
# simulate data
X <- data.frame(x1 = rnorm(50))
Y <- rbinom(50, 1, plogis(X$x1))
# compute lpo_auc for logistic regression
# use small B for fast run
boot <- boot_auc(Y = Y, X = X, B = 25, learner = "glm_wrapper")
```

---

## boot_scrnp *Compute the bootstrap-corrected estimator of SCRNP.*

---

### Description

This estimator is computed by re-sampling with replacement (i.e., bootstrap sampling) from the data. The SCRNP is computed for the learner trained on the full data. The SCRNP is then computed for the learner trained on each bootstrap sample. The average difference between the full data-trained learner and the bootstrap-trained learner is computed to estimate the bias in the full-data-estimated SCRNP. The final estimate of SCRNP is given by the difference in the full-data SCRNP and the estimated bias.

## Usage

```
boot_scrnp(
  Y,
  X,
  B = 200,
  learner = "glm_wrapper",
  sens = 0.95,
  correct632 = FALSE,
  ...
)
```

## Arguments

| | |
|---|---|
| Y | A numeric vector of outcomes, assume to equal 0 or 1. |
| X | A data.frame of variables for prediction. |
| B | The number of bootstrap samples. |
| learner | A wrapper that implements the desired method for building a prediction algorithm. See ?glm_wrapper or read the package vignette for more information on formatting learners. |
| sens | The sensitivity constraint to use. |
| correct632 | A boolean indicating whether to use the .632 correction. |
| ... | Other options, not currently used. |

## Value

A list with $scrnp the bootstrap-corrected estimate of SCRNP and $n_valid_boot as the number of bootstrap of bootstrap samples where learner successfully executed.

## Examples

```
# simulate data
X <- data.frame(x1 = rnorm(50))
Y <- rbinom(50, 1, plogis(X$x1))
# compute bootstrap estimate of scrnp for logistic regression
# use small B for fast run
boot <- boot_scrnp(Y = Y, X = X, B = 25, learner = "glm_wrapper")
```

---

# cardio *Cardiotocography*

---

## Description

Cardiotocography data from UCI machine learning repository. Raw data have been cleaned and an outcome column added that is a binary variable of predicting NSP (described below) = 2.

## Details

Data Set Information: 2126 fetal cardiotocograms (CTGs) were automatically processed and the respective diagnostic features measured. The CTGs were also classified by three expert obstetricians and a consensus classification label assigned to each of them. Classification was both with respect to a morphologic pattern (A, B, C. ...) and to a fetal state (N, S, P). Therefore the dataset can be used either for 10-class or 3-class experiments.

Attribute Information:

LB - FHR baseline (beats per minute)

AC - # of accelerations per second

FM - # of fetal movements per second

UC - # of uterine contractions per second

DL - # of light decelerations per second

DS - # of severe decelerations per second

DP - # of prolongued decelerations per second

ASTV - percentage of time with abnormal short term variability

MSTV - mean value of short term variability

ALTV - percentage of time with abnormal long term variability

MLTV - mean value of long term variability

Width - width of FHR histogram

Min - minimum of FHR histogram

Max - Maximum of FHR histogram

Nmax - # of histogram peaks

Nzeros - # of histogram zeros

Mode - histogram mode

Mean - histogram mean

Median - histogram median

Variance - histogram variance

Tendency - histogram tendency

CLASS - FHR pattern class code (1 to 10)

NSP - fetal state class code (N=normal; S=suspect; P=pathologic)

## Source

https://archive.ics.uci.edu/ml/datasets/Cardiotocography

## References

Ayres de Campos et al. (2000) SisPorto 2.0 A Program for Automated Analysis of Cardiotocograms. J Matern Fetal Med 5:311-318

`ci.cvAUC_withIC` *ci.cvAUC_withIC*

## Description

This function is nearly verbatim ci.cvAUC from the cvAUC package. The only difference is that it additionally returns estimated influence functions.

## Usage

```
ci.cvAUC_withIC(
  predictions,
  labels,
  label.ordering = NULL,
  folds = NULL,
  confidence = 0.95
)
```

## Arguments

| | |
|---|---|
| `predictions` | A vector, matrix, list, or data frame containing the predictions. |
| `labels` | A vector, matrix, list, or data frame containing the true class labels. Must have the same dimensions as `predictions`. |
| `label.ordering` | The default ordering of the classes can be changed by supplying a vector containing the negative and the positive class label (negative label first, positive label second). |
| `folds` | If specified, this must be a vector of fold ids equal in length to `predictions` and `labels`, or a list of length V (for V-fold cross-validation) of vectors of indexes for the observations contained in each fold. The `folds` argument must only be specified if the `predictions` and `labels` arguments are vectors. |
| `confidence` | number between 0 and 1 that represents confidence level. |

## Value

A list containing the following named elements:

| | |
|---|---|
| `cvAUC` | Cross-validated area under the curve estimate. |
| `se` | Standard error. |
| `ci` | A vector of length two containing the upper and lower bounds for the confidence interval. |
| `confidence` | A number between 0 and 1 representing the confidence. |
| `ic` | A vector of the influence function evaluated at observations. |

cv_auc *Estimates of CVAUC*

## Description

This function computes K-fold cross-validated estimates of the area under the receiver operating characteristics (ROC) curve (hereafter, AUC). This quantity can be interpreted as the probability that a randomly selected case will have higher predicted risk than a randomly selected control.

## Usage

```
cv_auc(
  Y,
  X,
  K = 10,
  learner = "glm_wrapper",
  nested_cv = TRUE,
  nested_K = K - 1,
  parallel = FALSE,
  max_cvtmle_iter = 10,
  cvtmle_ictol = 1/length(Y),
  prediction_list = NULL,
  ...
)
```

## Arguments

| | |
|---|---|
| Y | A numeric vector of outcomes, assume to equal 0 or 1. |
| X | A data.frame or matrix of variables for prediction. |
| K | The number of cross-validation folds (default is 10). |
| learner | A wrapper that implements the desired method for building a prediction algorithm. See See ?glm_wrapper or read the package vignette for more information on formatting learners. |
| nested_cv | A boolean indicating whether nested cross validation should be used to estimate the distribution of the prediction function. Default (TRUE) is best choice for aggressive learner's, while FALSE is reasonable for smooth learner's (e.g., logistic regression). |
| nested_K | If nested cross validation is used, how many inner folds should there be? Default (K-1) affords quicker computation by reusing training fold learner fits. |
| parallel | A boolean indicating whether prediction algorithms should be trained in parallel. Default to FALSE. |
| max_cvtmle_iter | Maximum number of iterations for the bias correction step of the CV-TMLE estimator (default 10). |

`cvtmle_ictol` The CV-TMLE will iterate `max_cvtmle_iter` is reached or mean of cross-validated efficient influence function is less than `cvtmle_ictol`.

`prediction_list`
For power users: a list of predictions made by `learner` that has a format compatible with `cvauc`.

`...` Other arguments, not currently used

## Details

To estimate the AUC of a particular prediction algorithm, K-fold cross-validation is commonly used: data are partitioned into K distinct groups and the prediction algorithm is developed using K-1 of these groups. In standard K-fold cross-validation, the AUC of this prediction algorithm is estimated using the remaining fold. This can be problematic when the number of observations is small or the number of cross-validation folds is large.

Here, we estimate relevant nuisance parameters in the training sample and use the validation sample to perform some form of bias correction – either through cross-validated targeted minimum loss-based estimation, estimating equations, or one-step estimation. When aggressive learning algorithms are applied, it is necessary to use an additional layer of cross-validation in the training sample to estimate the nuisance parameters. This is controlled via the `nested_cv` option below.

## Value

An object of class `"cvauc"`.

`est_cvtmle` cross-validated targeted minimum loss-based estimator of K-fold CV AUC

`iter_cvtmle` iterations needed to achieve convergence of CVTMLE algorithm

`cvtmle_trace` the value of the CVTMLE at each iteration of the targeting algorithm

`se_cvtmle` estimated standard error based on targeted nuisance parameters

`est_init` plug-in estimate of CV AUC where nuisance parameters are estimated in the training sample

`est_empirical` the standard K-fold CV AUC estimator

`se_empirical` estimated standard error for the standard estimator

`est_onestep` cross-validated one-step estimate of K-fold CV AUC

`se_onestep` estimated standard error for the one-step estimator

`est_esteq` cross-validated estimating equations estimate of K-fold CV AUC

`se_esteq` estimated standard error for the estimating equations estimator (same as for one-step)

`folds` list of observation indexes in each validation fold

`ic_cvtmle` influence function evaluated at the targeted nuisance parameter estimates

`ic_onestep` influence function evaluated at the training-fold-estimated nuisance parameters

`ic_esteq` influence function evaluated at the training-fold-estimated nuisance parameters

`ic_empirical` influence function evaluated at the validation-fold estimated nuisance parameters

`prediction_list` a list of output from the cross-validated model training; see the individual wrapper function documentation for further details

## Examples

```
# simulate data
n <- 200
p <- 10
X <- data.frame(matrix(rnorm(n*p), nrow = n, ncol = p))
Y <- rbinom(n, 1, plogis(X[,1] + X[,10]))

# get cv auc estimates for logistic regression
cv_auc_ests <- cv_auc(Y = Y, X = X, K = 5, learner = "glm_wrapper")

# get cv auc estimates for random forest
# using nested cross-validation for nuisance parameter estimation

fit <- cv_auc(Y = Y, X = X, K = 5,
              learner = "randomforest_wrapper",
              nested_cv = TRUE)
```

---

# cv_scrnp *Estimates of CV SCNP*

---

## Description

This function computes K-fold cross-validated estimates of estimates of cross-validated sensitivity-constrained rate of negative prediction (SCRNP). This quantity can be interpreted as the rate of negative classification for a fixed constraint on the sensitivity of a prediction algorithm. Thus, if an algorithm has a high SCRNP, it will also have a high positive predictive value.

## Usage

```
cv_scrnp(
  Y,
  X,
  K = 10,
  sens = 0.95,
  learner = "glm_wrapper",
  nested_cv = TRUE,
  nested_K = K - 1,
  parallel = FALSE,
  max_cvtmle_iter = 10,
  cvtmle_ictol = 1/length(Y),
  quantile_type = 8,
  prediction_list = NULL,
  ...
)
```

## Arguments

| | |
|---|---|
| Y | A numeric vector of outcomes, assume to equal 0 or 1. |
| X | A data.frame or matrix of variables for prediction. |
| K | The number of cross-validation folds (default is 10). |
| sens | The sensitivity constraint imposed on the rate of negative prediction (see description). |
| learner | A wrapper that implements the desired method for building a prediction algorithm. |
| nested_cv | A boolean indicating whether nested cross validation should be used to estimate the distribution of the prediction function. Default (TRUE) is best choice for aggressive learner's, while FALSE is reasonable for smooth learner's (e.g., logistic regression). |
| nested_K | If nested cross validation is used, how many inner folds should there be? Default (K-1) affords quicker computation by reusing training fold learner fits. |
| parallel | A boolean indicating whether prediction algorithms should be trained in parallel. Default to FALSE. |
| max_cvtmle_iter | Maximum number of iterations for the bias correction step of the CV-TMLE estimator (default 10). |
| cvtmle_ictol | The CV-TMLE will iterate max_cvtmle_iter is reached or mean of cross-validated efficient influence function is less than cvtmle_cvtmle_ictol. |
| quantile_type | Type of quantile estimator to be used. See quantile for description. |
| prediction_list | For power users: a list of predictions made by learner that has a format compatible with cvauc. |
| ... | Other arguments, not currently used |

## Details

To estimate the SCRNP using K-fold cross-validation is problematic. If data are partitioned into K distinct groups, depending on the sample size and choice of K, the validation sample may be quite small. In order to estimate SCRNP, we require estimation of a quantile of the predictor's distribution. More extreme quantiles (which correspond to high sensitivity constraints) are difficult to estimate using few observations. Here, we estimate relevant nuisance parameters in the training sample and use the validation sample to perform some form of bias correction – either through cross-validated targeted minimum loss-based estimation, estimating equations, or one-step estimation. When aggressive learning algorithms are applied, it is necessary to use an additional layer of cross-validation in the training sample to estimate the nuisance parameters. This is controlled via the nested_cv option below.

## Value

An object of class "scrnp".

est_cvtmle cross-validated targeted minimum loss-based estimator of K-fold CV AUC

`iter_cvtmle` iterations needed to achieve convergence of CVTMLE algorithm

`cvtmle_trace` the value of the CVTMLE at each iteration of the targeting algorithm

`se_cvtmle` estimated standard error based on targeted nuisance parameters

`est_init` plug-in estimate of CV AUC where nuisance parameters are estimated in the training sample

`est_empirical` the standard K-fold CV AUC estimator

`se_empirical` estimated standard error for the standard estimator

`est_onestep` cross-validated one-step estimate of K-fold CV AUC

`se_onestep` estimated standard error for the one-step estimator

`est_esteq` cross-validated estimating equations estimate of K-fold CV AUC (here, equivalent to one-step, since the estimating equation is linear in SCRNP)

`se_esteq` estimated standard error for the estimating equations estimator (same as one-step)

`folds` list of observation indexes in each validation fold

`ic_cvtmle` influence function evaluated at the targeted nuisance parameter estimates

`ic_onestep` influence function evaluated at the training-fold-estimated nuisance parameters

`ic_esteq` influence function evaluated at the training-fold-estimated nuisance parameters

`ic_empirical` influence function evaluated at the validation-fold estimated nuisance parameters

`prediction_list` a list of output from the cross-validated model training; see the individual wrapper function documentation for further details

## Examples

```
# simulate data
n <- 200
p <- 10
X <- data.frame(matrix(rnorm(n*p), nrow = n, ncol = p))
Y <- rbinom(n, 1, plogis(X[,1] + X[,10]))

# estimate cv scrnp of logistic regression
scrnp_ests <- cv_scrnp(Y = Y, X = X, K = 5,
                       nested_cv = FALSE,
                       learner = "glm_wrapper")

# estimate cv scrnp of random forest with nested
# cross-validation for nuisance parameter estimation

scrnp_ests <- cv_scrnp(Y = Y, X = X, K = 5,
                       nested_cv = TRUE,
                       learner = "randomforest_wrapper")
```

drugs *drugs*

## Description

"Drug consumption (quantified) Data Set" from UCI Machine Learning Repository. Raw data have been processed and an `outcome` (heroin use) column added.

## Details

Data Set Information (copied from UCI library):

Database contains records for 1885 respondents. For each respondent 12 attributes are known: Personality measurements which include NEO-FFI-R (neuroticism, extraversion, openness to experience, agreeableness, and conscientiousness), BIS-11 (impulsivity), and ImpSS (sensation seeking), level of education, age, gender, country of residence and ethnicity. All input attributes are originally categorical and are quantified. After quantification values of all input features can be considered as real-valued. In addition, participants were questioned concerning their use of 18 legal and illegal drugs (alcohol, amphetamines, amyl nitrite, benzodiazepine, cannabis, chocolate, cocaine, caffeine, crack, ecstasy, heroin, ketamine, legal highs, LSD, methadone, mushrooms, nicotine and volatile substance abuse and one fictitious drug (Semeron) which was introduced to identify over-claimers. For each drug they have to select one of the answers: never used the drug, used it over a decade ago, or in the last decade, year, month, week, or day.

Database contains 18 classification problems. Each of independent label variables contains seven classes: "Never Used", "Used over a Decade Ago", "Used in Last Decade", "Used in Last Year", "Used in Last Month", "Used in Last Week", and "Used in Last Day".

Problem which can be solved:

* Seven class classifications for each drug separately.

* Problem can be transformed to binary classification by union of part of classes into one new class. For example, "Never Used", "Used over a Decade Ago" form class "Non-user" and all other classes form class "User".

* The best binarization of classes for each attribute.

* Evaluation of risk to be drug consumer for each drug.

Detailed description of database and process of data quantification are presented in E. Fehrman, A. K. Muhammad, E. M. Mirkes, V. Egan and A. N. Gorban, "The Five Factor Model of personality and evaluation of drug consumption risk.," arXiv [Web Link], 2015

Paper above solve binary classification problem for all drugs. For most of drugs sensitivity and specificity are greater than 75%.

## Source

https://archive.ics.uci.edu/ml/datasets/Drug+consumption+%28quantified%29

## References

https://arxiv.org/abs/1506.06297

## fluc_mod_optim_0 *Helper function for CVTMLE grid search*

### Description

Helper function for CVTMLE grid search

### Usage

```
fluc_mod_optim_0(epsilon, fld, tol = 0.001)
```

### Arguments

| | |
|---|---|
| epsilon | Fluctuation parameter |
| fld | The full_long_data_list object created |
| tol | Tolerance on predictions close to 0 or 1 |

### Value

A numeric value of negative log-likelihood

## fluc_mod_optim_1 *Helper function for CVTMLE grid search*

### Description

Helper function for CVTMLE grid search

### Usage

```
fluc_mod_optim_1(epsilon, fld, tol = 0.001)
```

### Arguments

| | |
|---|---|
| epsilon | Fluctuation parameter |
| fld | full_long_data_list |
| tol | Tolerance on predictions close to 0 or 1 |

### Value

A numeric value of negative log-likelihood

## F_nBn_star — *Compute the targeted conditional cumulative distribution of the learner at a point*

### Description

Compute the targeted conditional cumulative distribution of the learner at a point

### Usage

```
F_nBn_star(psi_x, y, train_pred, train_y, epsilon = 0, tol = 0.001)
```

### Arguments

| | |
|---|---|
| psi_x | Value to compute conditional (on Y=y) cdf of learner |
| y | Value of Y to condition on |
| train_pred | Values of Psi_nBn(X) from training sample |
| train_y | Values of Y from training sample |
| epsilon | Vector of fluctuation parameter estimates |
| tol | Truncation level for logistic transformation |

### Value

Numeric value of CDF at psi_x

## F_nBn_star_nested_cv — *Compute the targeted conditional cumulative distribution of the learner at a point where the initial distribution is based on cross validation*

### Description

Compute the targeted conditional cumulative distribution of the learner at a point where the initial distribution is based on cross validation

### Usage

```
F_nBn_star_nested_cv(
  psi_x,
  y,
  inner_valid_prediction_and_y_list,
  epsilon = 0,
  tol = 0.001
)
```

### Arguments

| | |
|---|---|
| psi_x | Value to compute conditional (on Y=y) cdf of learner |
| y | Value of Y to condition on |
| inner_valid_prediction_and_y_list | A list of predictions and y's from .get_predictions. |
| epsilon | Vector of fluctuation parameter estimates |
| tol | A truncation level when taking logit transformations. |

### Value

Numeric value of CDF at psi_x

---

## glmnet_wrapper *Wrapper for fitting a lasso using package* glmnet.

---

### Description

Compatible learner wrappers for this package should have a specific format. Namely they should take as input a list called train that contains named objects $Y and $X, that contain, respectively, the outcomes and predictors in a particular training fold. Other options may be passed in to the function as well. The function must output a list with the following named objects: test_pred = predictions of test$Y based on the learner fit using train$X; train_pred = prediction of train$Y based on the learner fit using train$X; model = the fitted model (only necessary if you desire to look at this model later, not used for internal computations); train_y = a copy of train$Y; test_y = a copy of test$Y.

### Usage

```
glmnet_wrapper(
  train,
  test,
  alpha = 1,
  nfolds = 5,
  nlambda = 100,
  use_min = TRUE,
  loss = "deviance",
  ...
)
```

### Arguments

| | |
|---|---|
| train | A list with named objects Y and X (see description). |
| test | A list with named objects Y and X (see description). |
| alpha | See glmnet for further description. |
| nfolds | See glmnet for further description. |

| | |
|---|---|
| nlambda | See glmnet for further description. |
| use_min | See glmnet for further description. |
| loss | See glmnet for further description. |
| ... | Other options (passed to cv.glmnet) |

## Details

This particular wrapper implements glmnet. We refer readers to the original package's documentation for more details.

## Value

A list with named objects (see description).

## Examples

```
# load super learner package
library(glmnet)
# simulate data
# make list of training data
train_X <- data.frame(x1 = runif(50), x2 = runif(50))
train_Y <- rbinom(50, 1, plogis(train_X$x1))
train <- list(Y = train_Y, X = train_X)
# make list of test data
test_X <- data.frame(x1 = runif(50), x2 = runif(50))
test_Y <- rbinom(50, 1, plogis(train_X$x1))
test <- list(Y = test_Y, X = test_X)
# fit super learner
glmnet_wrap <- glmnet_wrapper(train = train, test = test)
```

---

# glm_wrapper *Wrapper for fitting a logistic regression using* glm.

---

## Description

Compatible learner wrappers for this package should have a specific format. Namely they should take as input a list called train that contains named objects $Y and $X, that contain, respectively, the outcomes and predictors in a particular training fold. Other options may be passed in to the function as well. The function must output a list with the following named objects: test_pred = predictions of test$Y based on the learner fit using train$X; train_pred = prediction of train$Y based on the learner fit using train$X; model = the fitted model (only necessary if you desire to look at this model later, not used for internal computations); train_y = a copy of train$Y; test_y = a copy of test$Y.

## Usage

```
glm_wrapper(train, test)
```

### Arguments

| | |
|---|---|
| train | A list with named objects Y and X (see description). |
| test | A list with named objects Y and X (see description). |

### Details

This particular wrapper implements a logistic regression using glm. We refer readers to the original package's documentation for more details.

### Value

A list with named objects (see description).

### Examples

```
# simulate data
# make list of training data
train_X <- data.frame(x1 = runif(50))
train_Y <- rbinom(50, 1, plogis(train_X$x1))
train <- list(Y = train_Y, X = train_X)
# make list of test data
test_X <- data.frame(x1 = runif(50))
test_Y <- rbinom(50, 1, plogis(train_X$x1))
test <- list(Y = test_Y, X = test_X)
# fit glm
glm_wrap <- glm_wrapper(train = train, test = test)
```

---

## lpo_auc *Compute the leave-pair-out cross-validation estimator of AUC.*

---

### Description

This estimator is computed by leaving out a pair of one case (Y = 1) and one control (Y = 0). The learner is trained on the remaining observations and predicted values are obtained for the left-out pair. The estimate is given by the proportion of left-out pairs for which the case had higher predicted risk than the control.

### Usage

```
lpo_auc(Y, X, learner = "glm_wrapper", max_pairs = NULL, parallel = FALSE, ...)
```

### Arguments

| | |
|---|---|
| Y | A numeric vector of outcomes, assume to equal 0 or 1. |
| X | A data.frame of variables for prediction. |
| learner | A wrapper that implements the desired method for building a prediction algorithm. See ?glm_wrapper or read the package vignette for more information on formatting learners. |

| | |
|---|---|
| max_pairs | The maximum number of pairs to leave out. |
| parallel | A boolean indicating whether prediction algorithms should be trained in parallel. Default to FALSE. |
| ... | Other options (not currently used) |

## Examples

```
# simulate data
X <- data.frame(x1 = rnorm(50))
Y <- rbinom(50, 1, plogis(X$x1))
# compute lpo_auc for logistic regression
lpo <- lpo_auc(Y = Y, X = X, learner = "glm_wrapper")
```

---

one_boot_auc *Internal function used to perform one bootstrap sample. The function* trys *to fit* learner *on a bootstrap sample. If for some reason (e.g., the bootstrap sample contains no observations with* Y = 1*) the learner fails, then the function returns* NA*. These* NAs *are ignored later when computing the bootstrap corrected estimate.*

---

## Description

Internal function used to perform one bootstrap sample. The function trys to fit learner on a bootstrap sample. If for some reason (e.g., the bootstrap sample contains no observations with Y = 1) the learner fails, then the function returns NA. These NAs are ignored later when computing the bootstrap corrected estimate.

## Usage

```
one_boot_auc(Y, X, n, correct632, learner)
```

## Arguments

| | |
|---|---|
| Y | A numeric binary outcome |
| X | A data.frame of variables for prediction. |
| n | Number of observations |
| correct632 | A boolean indicating whether to use the .632 correction. |
| learner | A wrapper that implements the desired method for building a prediction algorithm. See ?glm_wrapper or read the package vignette for more information on formatting learners. |

## Value

If learner executes successfully, a numeric estimate of AUC on this bootstrap sample. Otherwise the function returns NA.

## one_boot_scrnp *Internal function used to perform one bootstrap sample. The function* trys *to fit* learner *on a bootstrap sample. If for some reason (e.g., the bootstrap sample contains no observations with* Y = 1*) the learner fails, then the function returns* NA*. These* NAs *are ignored later when computing the bootstrap corrected estimate.*

### Description

Internal function used to perform one bootstrap sample. The function trys to fit learner on a bootstrap sample. If for some reason (e.g., the bootstrap sample contains no observations with Y = 1) the learner fails, then the function returns NA. These NAs are ignored later when computing the bootstrap corrected estimate.

### Usage

```
one_boot_scrnp(Y, X, n, correct632, learner, sens)
```

### Arguments

| | |
|---|---|
| Y | A numeric binary outcome |
| X | A data.frame of variables for prediction. |
| n | Number of observations |
| correct632 | A boolean indicating whether to use the .632 correction. |
| learner | A wrapper that implements the desired method for building a prediction algorithm. See ?glm_wrapper or read the package vignette for more information on formatting learners. |
| sens | The sensitivity constraint to use. |

### Value

If learner executes successfully, a numeric estimate of AUC on this bootstrap sample. Otherwise the function returns NA.

## print.cvauc *Print results of cv_auc*

### Description

Print results of cv_auc

### Usage

```
## S3 method for class 'cvauc'
print(x, ci_level = 0.95, se_type = "std", ...)
```

### Arguments

| | |
|---|---|
| x | An object of class "cvauc" |
| ci_level | Level of confidence interval to print. Defaults to 0.95. |
| se_type | The type of standard error (currently only "std") |
| ... | Other options (not currently used) |

---

## print.scrnp *Print results of cv_scrnp*

---

### Description

Print results of cv_scrnp

### Usage

```
## S3 method for class 'scrnp'
print(x, se_type = "std", ci_level = 0.95, ...)
```

### Arguments

| | |
|---|---|
| x | An object of class "cvauc" |
| se_type | The type of standard error (currently only "std") |
| ci_level | Level of confidence interval to print. Defaults to 0.95. |
| ... | Other options (not currently used) |

---

## randomforest_wrapper *Wrapper for fitting a random forest using randomForest.*

---

### Description

Compatible learner wrappers for this package should have a specific format. Namely they should take as input a list called `train` that contains named objects `$Y` and `$X`, that contain, respectively, the outcomes and predictors in a particular training fold. Other options may be passed in to the function as well. The function must output a list with the following named objects: `test_pred` = predictions of `test$Y` based on the learner fit using `train$X`; `train_pred` = prediction of `train$Y` based on the learner fit using `train$X`; `model` = the fitted model (only necessary if you desire to look at this model later, not used for internal computations); `train_y` = a copy of `train$Y`; `test_y` = a copy of `test$Y`.

## Usage

```
randomforest_wrapper(
  train,
  test,
  mtry = floor(sqrt(ncol(train$X))),
  ntree = 1000,
  nodesize = 1,
  maxnodes = NULL,
  importance = FALSE,
  ...
)
```

## Arguments

| | |
|---|---|
| `train` | A list with named objects `Y` and `X` (see description). |
| `test` | A list with named objects `Y` and `X` (see description). |
| `mtry` | See randomForest. |
| `ntree` | See randomForest. |
| `nodesize` | See randomForest. |
| `maxnodes` | See randomForest. |
| `importance` | See randomForest. |
| `...` | Other options (passed to `randomForest`) |

## Details

This particular wrapper implements the randomForest ensemble methodology. We refer readers to the original package's documentation for more details.

## Value

A list with named objects (see description).

## Examples

```
# simulate data
# make list of training data
train_X <- data.frame(x1 = runif(50))
train_Y <- rbinom(50, 1, plogis(train_X$x1))
train <- list(Y = train_Y, X = train_X)
# make list of test data
test_X <- data.frame(x1 = runif(50))
test_Y <- rbinom(50, 1, plogis(train_X$x1))
test <- list(Y = test_Y, X = test_X)
# fit randomforest
rf_wrap <- randomforest_wrapper(train = train, test = test)
```

ranger_wrapper *Wrapper for fitting a random forest using ranger.*

## Description

Compatible learner wrappers for this package should have a specific format. Namely they should take as input a list called `train` that contains named objects `$Y` and `$X`, that contain, respectively, the outcomes and predictors in a particular training fold. Other options may be passed in to the function as well. The function must output a list with the following named objects: `test_pred` = predictions of `test$Y` based on the learner fit using `train$X`; `train_pred` = prediction of `train$Y` based on the learner fit using `train$X`; `model` = the fitted model (only necessary if you desire to look at this model later, not used for internal computations); `train_y` = a copy of `train$Y`; `test_y` = a copy of `test$Y`.

## Usage

```
ranger_wrapper(
  train,
  test,
  num.trees = 500,
  mtry = floor(sqrt(ncol(train$X))),
  write.forest = TRUE,
  probability = TRUE,
  min.node.size = 5,
  replace = TRUE,
  sample.fraction = ifelse(replace, 1, 0.632),
  num.threads = 1,
  verbose = TRUE,
  ...
)
```

## Arguments

| | |
|---|---|
| `train` | A list with named objects `Y` and `X` (see description). |
| `test` | A list with named objects `Y` and `X` (see description). |
| `num.trees` | See ranger. |
| `mtry` | See ranger. |
| `write.forest` | See ranger. |
| `probability` | See ranger. |
| `min.node.size` | See ranger. |
| `replace` | See ranger. |
| `sample.fraction` | See ranger. |
| `num.threads` | See ranger. |
| `verbose` | See ranger. |
| `...` | Other options (passed to `ranger`) |

## Details

This particular wrapper implements the ranger ensemble methodology. We refer readers to the original package's documentation for more details.

## Value

A list with named objects (see description).

## Examples

```
# simulate data
# make list of training data
train_X <- data.frame(x1 = runif(50))
train_Y <- rbinom(50, 1, plogis(train_X$x1))
train <- list(Y = train_Y, X = train_X)
# make list of test data
test_X <- data.frame(x1 = runif(50))
test_Y <- rbinom(50, 1, plogis(train_X$x1))
test <- list(Y = test_Y, X = test_X)
# fit ranger
rf_wrap <- ranger_wrapper(train = train, test = test)
```

---

# stepglm_wrapper *Wrapper for fitting a forward stepwise logistic regression using* glm.

---

## Description

Compatible learner wrappers for this package should have a specific format. Namely they should take as input a list called `train` that contains named objects `$Y` and `$X`, that contain, respectively, the outcomes and predictors in a particular training fold. Other options may be passed in to the function as well. The function must output a list with the following named objects: `test_pred` = predictions of `test$Y` based on the learner fit using `train$X`; `train_pred` = prediction of `train$Y` based on the learner fit using `train$X`; `model` = the fitted model (only necessary if you desire to look at this model later, not used for internal computations); `train_y` = a copy of `train$Y`; `test_y` = a copy of `test$Y`.

## Usage

```
stepglm_wrapper(train, test)
```

## Arguments

| | |
|---|---|
| `train` | A list with named objects `Y` and `X` (see description). |
| `test` | A list with named objects `Y` and `X` (see description). |

## Details

This particular wrapper implements a forward stepwise logistic regression using glm and step. We refer readers to the original package's documentation for more details.

## Value

A list with named objects (see description).

## Examples

```
# simulate data
# make list of training data
train_X <- data.frame(x1 = runif(50))
train_Y <- rbinom(50, 1, plogis(train_X$x1))
train <- list(Y = train_Y, X = train_X)
# make list of test data
test_X <- data.frame(x1 = runif(50))
test_Y <- rbinom(50, 1, plogis(train_X$x1))
test <- list(Y = test_Y, X = test_X)
# fit stepwise glm
step_wrap <- stepglm_wrapper(train = train, test = test)
```

---

# superlearner_wrapper *Wrapper for fitting a super learner based on* SuperLearner.

---

## Description

Compatible learner wrappers for this package should have a specific format. Namely they should take as input a list called `train` that contains named objects `$Y` and `$X`, that contain, respectively, the outcomes and predictors in a particular training fold. Other options may be passed in to the function as well. The function must output a list with the following named objects: `test_pred` = predictions of `test$Y` based on the learner fit using `train$X`; `train_pred` = prediction of `train$Y` based on the learner fit using `train$X`; `model` = the fitted model (only necessary if you desire to look at this model later, not used for internal computations); `train_y` = a copy of `train$Y`; `test_y` = a copy of `test$Y`.

## Usage

```
superlearner_wrapper(train, test, SL.library = c("SL.mean"), ...)
```

## Arguments

| | |
|---|---|
| `train` | A list with named objects `Y` and `X` (see description). |
| `test` | A list with named objects `Y` and `X` (see description). |
| `SL.library` | `SuperLearner` library. See SuperLearner for further description. |
| `...` | Other options (passed to `SuperLearner`) |

## Details

This particular wrapper implements the SuperLearner ensemble methodology. We refer readers to the original package's documentation for more details.

## Value

A list with named objects (see description).

## Examples

```
# load super learner package
library(SuperLearner)
# simulate data
# make list of training data
train_X <- data.frame(x1 = runif(50))
train_Y <- rbinom(50, 1, plogis(train_X$x1))
train <- list(Y = train_Y, X = train_X)
# make list of test data
test_X <- data.frame(x1 = runif(50))
test_Y <- rbinom(50, 1, plogis(train_X$x1))
test <- list(Y = test_Y, X = test_X)
# fit super learner
sl_wrap <- superlearner_wrapper(train = train,
                                test = test,
                                SL.library = c("SL.mean","SL.glm"))
```

---

`wine` *wine*

---

## Description

"Wine Quality" data set from UCI Machine Learning Repository. The red and white wine data sets have been combined with an added attribute for red vs. white.

## Details

Data Set Information (copied from UCI):

The two datasets are related to red and white variants of the Portuguese "Vinho Verde" wine. For more details, consult: [Web Link] or the reference [Cortez et al., 2009]. Due to privacy and logistic issues, only physicochemical (inputs) and sensory (the output) variables are available (e.g. there is no data about grape types, wine brand, wine selling price, etc.).

These datasets can be viewed as classification or regression tasks. The classes are ordered and not balanced (e.g. there are munch more normal wines than excellent or poor ones). Outlier detection algorithms could be used to detect the few excellent or poor wines. Also, we are not sure if all input variables are relevant. So it could be interesting to test feature selection methods.

Attribute Information:

For more information, read [Cortez et al., 2009].

Input variables (based on physicochemical tests):

1 - fixed acidity

2 - volatile acidity

3 - citric acid

4 - residual sugar

5 - chlorides

6 - free sulfur dioxide

7 - total sulfur dioxide

8 - density

9 - pH

10 - sulphates

11 - alcohol

Output variable (based on sensory data):

12 - quality (score between 0 and 10)

## Source

https://archive.ics.uci.edu/ml/datasets/Wine+Quality

## References

P. Cortez, A. Cerdeira, F. Almeida, T. Matos and J. Reis. Modeling wine preferences by data mining from physicochemical properties. In Decision Support Systems, Elsevier, 47(4):547-553, 2009.

https://doi.org/10.1016/j.dss.2009.05.016

---

`xgboost_wrapper` *Wrapper for fitting eXtreme gradient boosting via* `xgboost`

---

## Description

Compatible learner wrappers for this package should have a specific format. Namely they should take as input a list called `train` that contains named objects `$Y` and `$X`, that contain, respectively, the outcomes and predictors in a particular training fold. Other options may be passed in to the function as well. The function must output a list with the following named objects: `test_pred` = predictions of `test$Y` based on the learner fit using `train$X`; `train_pred` = prediction of `train$Y` based on the learner fit using `train$X`; `model` = the fitted model (only necessary if you desire to look at this model later, not used for internal computations); `train_y` = a copy of `train$Y`; `test_y` = a copy of `test$Y`.

## Usage

```
xgboost_wrapper(
  test,
  train,
  ntrees = 500,
  max_depth = 4,
  shrinkage = 0.1,
```

```
  minobspernode = 2,
  params = list(),
  nthread = 1,
  verbose = 0,
  save_period = NULL
)
```

## Arguments

| | |
|---|---|
| test | A list with named objects Y and X (see description). |
| train | A list with named objects Y and X (see description). |
| ntrees | See xgboost |
| max_depth | See xgboost |
| shrinkage | See xgboost |
| minobspernode | See xgboost |
| params | See xgboost |
| nthread | See xgboost |
| verbose | See xgboost |
| save_period | See xgboost |

## Details

This particular wrapper implements eXtreme gradient boosting using xgboost. We refer readers to the original package's documentation for more details.

## Value

A list with named objects (see description).

## Examples

```
# simulate data
# make list of training data
train_X <- data.frame(x1 = runif(50))
train_Y <- rbinom(50, 1, plogis(train_X$x1))
train <- list(Y = train_Y, X = train_X)
# make list of test data
test_X <- data.frame(x1 = runif(50))
test_Y <- rbinom(50, 1, plogis(train_X$x1))
test <- list(Y = test_Y, X = test_X)
# fit xgboost
xgb_wrap <- xgboost_wrapper(train = train, test = test)
```

# Index