Package 'cutpointr'

April 13, 2022

Type Package

Title Determine and Evaluate Optimal Cutpoints in Binary Classification Tasks

Version 1.1.2

Date 2022-04-13

Description Estimate cutpoints that optimize a specified metric in binary classification tasks and validate performance using bootstrapping. Some methods for more robust cutpoint estimation are supported, e.g. a parametric method assuming normal distributions, bootstrapped cutpoints, and smoothing of the metric values per cutpoint using Generalized Additive Models. Various plotting functions are included. For an overview of the package see Thiele and Hirschfeld (2021) <doi:10.18637/jss.v098.i11>.

License GPL-3

URL https://github.com/thie1e/cutpointr

BugReports https://github.com/thie1e/cutpointr/issues

Encoding UTF-8

LazyData true

Depends R (>= 3.5.0)

LinkingTo Rcpp

Imports gridExtra (>= 2.2.1), foreach (>= 1.4.3), dplyr (>= 0.8.0), tidyselect (>= 1.1.0), tidyr (>= 1.0.0), purrr (>= 0.3.0), tibble (>= 3.0.0), ggplot2 (>= 3.0.0), Rcpp (>= 0.12.12), stats, utils, rlang (>= 0.4.0)

RoxygenNote 7.1.2

Suggests KernSmooth (>= 2.23-15), fANCOVA (>= 0.5-1), testthat (>= 1.0.2), doRNG (>= 1.6), doParallel (>= 1.0.11), knitr, rmarkdown, mgcv (>= 1.8), crayon (>= 1.3.4), registry (>= 0.5-1), pkgmaker(>= 0.31.1), vctrs (>= 0.2.4)

VignetteBuilder knitr

NeedsCompilation yes

Author Christian Thiele [cre, aut] (<https://orcid.org/0000-0002-1156-5117>)

Maintainer Christian Thiele <c.thiele@gmx-topmail.de>

Repository CRAN

Date/Publication 2022-04-13 18:12:29 UTC

R topics documented:

`abs_d_ppv_npv` *Calculate the absolute difference of positive and negative predictive value*

Description

Calculate the absolute difference of positive predictive value (PPV) and negative predictive value (NPV) from true positives, false positives, true negatives and false negatives. The inputs must be vectors of equal length.

ppv = tp / (tp + fp)
npv = tn / (tn + fn)
abs_d_ppv_npv = |ppv - npv|

Usage

```
abs_d_ppv_npv(tp, fp, tn, fn, ...)
```

Arguments

tp	(numeric) number of true positives.
fp	(numeric) number of false positives.
tn	(numeric) number of true negatives.
fn	(numeric) number of false negatives.
...	for capturing additional arguments passed by method.

See Also

Other metric functions: F1_score(), Jaccard(), abs_d_sens_spec(), accuracy(), cohens_kappa(), cutpoint(), false_omission_rate(), metric_constrain(), misclassification_cost(), npv(), odds_ratio(), p_chisquared(), plr(), ppv(), precision(), prod_ppv_npv(), prod_sens_spec(), recall(), risk_ratio(), roc01(), sensitivity(), specificity(), sum_ppv_npv(), sum_sens_spec(), total_utility(), tpr(), tp(), youden()

Examples

```
abs_d_ppv_npv(10, 5, 20, 10)
abs_d_ppv_npv(c(10, 8), c(5, 7), c(20, 12), c(10, 18))
```

---

abs_d_sens_spec *Calculate the absolute difference of sensitivity and specificity*

Description

Calculate the absolute difference of sensitivity and specificity from true positives, false positives, true negatives and false negatives. The inputs must be vectors of equal length.

sensitivity = tp / (tp + fn)
specificity = tn / (tn + fp)
abs_d_sens_spec = |sensitivity - specificity|

Usage

```
abs_d_sens_spec(tp, fp, tn, fn, ...)
```

Arguments

tp	(numeric) number of true positives.
fp	(numeric) number of false positives.
tn	(numeric) number of true negatives.
fn	(numeric) number of false negatives.
...	for capturing additional arguments passed by method.

See Also

Other metric functions: F1_score(), Jaccard(), abs_d_ppv_npv(), accuracy(), cohens_kappa(), cutpoint(), false_omission_rate(), metric_constrain(), misclassification_cost(), npv(), odds_ratio(), p_chisquared(), plr(), ppv(), precision(), prod_ppv_npv(), prod_sens_spec(), recall(), risk_ratio(), roc01(), sensitivity(), specificity(), sum_ppv_npv(), sum_sens_spec(), total_utility(), tpr(), tp(), youden()

Examples

```
abs_d_sens_spec(10, 5, 20, 10)
abs_d_sens_spec(c(10, 8), c(5, 7), c(20, 12), c(10, 18))
```

---

accuracy *Calculate accuracy*

Description

Calculate accuracy from true positives, false positives, true negatives and false negatives. The inputs must be vectors of equal length.

accuracy = (tp + tn) / (tp + fp + tn + fn)

Usage

```
accuracy(tp, fp, tn, fn, ...)
```

Arguments

tp	(numeric) number of true positives.
fp	(numeric) number of false positives.
tn	(numeric) number of true negatives.
fn	(numeric) number of false negatives.
...	for capturing additional arguments passed by method.

See Also

Other metric functions: F1_score(), Jaccard(), abs_d_ppv_npv(), abs_d_sens_spec(), cohens_kappa(), cutpoint(), false_omission_rate(), metric_constrain(), misclassification_cost(), npv(), odds_ratio(), p_chisquared(), plr(), ppv(), precision(), prod_ppv_npv(), prod_sens_spec(), recall(), risk_ratio(), roc01(), sensitivity(), specificity(), sum_ppv_npv(), sum_sens_spec(), total_utility(), tpr(), tp(), youden()

Examples

```
accuracy(10, 5, 20, 10)
accuracy(c(10, 8), c(5, 7), c(20, 12), c(10, 18))
```

add_metric *Add metrics to a cutpointr or roc_cutpointr object*

Description

By default, the output of cutpointr includes the optimized metric and several other metrics. This function adds further metrics. Suitable metric functions are all metric functions that are included in the package or that comply with those standards.

Usage

```
add_metric(object, metric)

## S3 method for class 'cutpointr'
add_metric(object, metric)

## S3 method for class 'multi_cutpointr'
add_metric(object, metric)

## S3 method for class 'roc_cutpointr'
add_metric(object, metric)
```

Arguments

object	A cutpointr or roc_cutpointr object.
metric	(list) A list of metric functions to be added.

Value

A cutpointr or roc_cutpointr object (a data.frame) with one or more added columns.

See Also

Other main cutpointr functions: boot_ci(), boot_test(), cutpointr(), multi_cutpointr(), predict.cutpointr(), roc()

Examples

```
library(dplyr)
library(cutpointr)
cutpointr(suicide, dsi, suicide, gender) %>%
  add_metric(list(ppv, npv)) %>%
  select(optimal_cutpoint, subgroup, AUC, sum_sens_spec, ppv, npv)
```

auc *Calculate AUC from a roc_cutpointr or cutpointr object*

Description

Calculate the area under the ROC curve using the trapezoidal rule.

Usage

```
auc(x)

## S3 method for class 'roc_cutpointr'
auc(x)

## S3 method for class 'cutpointr'
auc(x)
```

Arguments

x	Data frame resulting from the roc() or cutpointr() function.

Value

Numeric vector of AUC values

Source

Forked from the AUC package

boot_ci *Calculate bootstrap confidence intervals from a cutpointr object*

Description

Given a `cutpointr` object that includes bootstrap results this function calculates a bootstrap confidence interval for a selected variable. Missing values are removed before calculating the quantiles. In the case of multiple optimal cutpoints all cutpoints / metric values are included in the calculation. Values of the selected variable are returned for the percentiles alpha / 2 and 1 - alpha / 2. The metrics in the bootstrap data frames of `cutpointr` are suffixed with `_b` and `_oob` to indicate in-bag and out-of-bag, respectively. For example, to calculate quantiles of the in-bag AUC `variable = AUC_b` should be set.

Usage

```
boot_ci(x, variable, in_bag = TRUE, alpha = 0.05)
```

Arguments

x	A cutpointr object with bootstrap results
variable	Variable to calculate CI for
in_bag	Whether the in-bag or out-of-bag results should be used for testing
alpha	Alpha level. Quantiles of the bootstrapped values are returned for (alpha / 2) and 1 - (alpha / 2).

Value

A data frame with the columns quantile and value

See Also

Other main cutpointr functions: add_metric(), boot_test(), cutpointr(), multi_cutpointr(), predict.cutpointr(), roc()

Examples

```
## Not run:
opt_cut <- cutpointr(suicide, dsi, suicide, gender,
  metric = youden, boot_runs = 1000)
boot_ci(opt_cut, optimal_cutpoint, in_bag = FALSE, alpha = 0.05)
boot_ci(opt_cut, acc, in_bag = FALSE, alpha = 0.05)
boot_ci(opt_cut, cohens_kappa, in_bag = FALSE, alpha = 0.05)
boot_ci(opt_cut, AUC, in_bag = TRUE, alpha = 0.05)

## End(Not run)
```

---

boot_test *Test for equivalence of a metric*

Description

This function performs a significance test based on the bootstrap results of cutpointr to test whether a chosen metric is equal between subgroups or between two cutpointr objects. The test statistic is calculated as the standardized difference of the metric between groups. If x contains subgroups, the test is run on all possible pairings of subgroups. An additional adjusted p-value is returned in that case.

Usage

```
boot_test(x, y = NULL, variable = "AUC", in_bag = TRUE, correction = "holm")
```

Arguments

x	A cutpointr object with bootstrap results
y	If x does not contain subgroups another cutpointr object
variable	The variable for testing
in_bag	Whether the in-bag or out-of-bag results should be used for testing
correction	The type of correction for multiple testing. Possible values are as in p.adjust.methods

Details

The variable name is looked up in the columns of the bootstrap results where the suffixes _b and _oob indicate in-bag and out-of-bag estimates, respectively (controlled via the in_bag argument). Possible values are optimal_cutpoint, AUC, acc, sensitivity, specificity, and the metric that was selected in cutpointr. Note that there is no "out-of-bag optimal cutpoint", so when selecting variable = optimal_cutpoint the test will be based on the in-bag data.

The test statistic is calculated as z = (t1 - t2) / sd(t1 - t2) where t1 and t2 are the metric values on the full sample and sd(t1 - t2) is the standard deviation of the differences of the metric values per bootstrap repetition. The test is two-sided.

If two cutpointr objects are compared and the numbers of bootstrap repetitions differ, the smaller number will be used.

Since pairwise differences are calculated for this test, the test function does not support multiple optimal cutpoints, because it is unclear how the differences should be calculated in that case.

Value

A data.frame (a tibble) with the columns test_var, p, d, sd_d, z and in_bag. If a grouped cutpointr object was tested, the additional columns subgroup1, subgroup2 and p_adj are returned.

Source

Robin, X., Turck, N., Hainard, A., Tiberti, N., Lisacek, F., Sanchez, J.-C., & Müller, M. (2011). pROC: An open-source package for R and S+ to analyze and compare ROC curves. BMC Bioinformatics, 12(1), 77. https://doi.org/10.1186/1471-2105-12-77

See Also

Other main cutpointr functions: add_metric(), boot_ci(), cutpointr(), multi_cutpointr(), predict.cutpointr(), roc()

Examples

```
## Not run:
library(cutpointr)
library(dplyr)
set.seed(734)
cp_f <- cutpointr(suicide %>% filter(gender == "female"), dsi, suicide,
  boot_runs = 1000, boot_stratify = TRUE)
set.seed(928)
```

```
cp_m <- cutpointr(suicide %>% filter(gender == "male"), dsi, suicide,
  boot_runs = 1000, boot_stratify = TRUE)
# No significant differences:
boot_test(cp_f, cp_m, AUC, in_bag = TRUE)
boot_test(cp_f, cp_m, sum_sens_spec, in_bag = FALSE)

set.seed(135)
cp <- cutpointr(suicide, dsi, suicide, gender, boot_runs = 1000,
  boot_stratify = TRUE)
# Roughly same result as above:
boot_test(cp, variable = AUC, in_bag = TRUE)
boot_test(cp, variable = sum_sens_spec, in_bag = FALSE)

## End(Not run)
```

---

cohens_kappa *Calculate Cohen's Kappa*

Description

Calculate the Kappa metric from true positives, false positives, true negatives and false negatives. The inputs must be vectors of equal length.

mrg_a = ((tp + fn) * (tp + fp)) / (tp + fn + fp + tn)
mrg_b = ((fp + tn) * (fn + tn)) / (tp + fn + fp + tn)
expec_agree = (mrg_a + mrg_b) / (tp + fn + fp + tn)
obs_agree = (tp + tn) / (tp + fn + fp + tn)
cohens_kappa = (obs_agree - expec_agree) / (1 - expec_agree)

Usage

```
cohens_kappa(tp, fp, tn, fn, ...)
```

Arguments

tp	(numeric) number of true positives.
fp	(numeric) number of false positives.
tn	(numeric) number of true negatives.
fn	(numeric) number of false negatives.
...	for capturing additional arguments passed by method.

Value

A numeric matrix with the column name "cohens_kappa".

See Also

Other metric functions: F1_score(), Jaccard(), abs_d_ppv_npv(), abs_d_sens_spec(), accuracy(), cutpoint(), false_omission_rate(), metric_constrain(), misclassification_cost(), npv(), odds_ratio(), p_chisquared(), plr(), ppv(), precision(), prod_ppv_npv(), prod_sens_spec(), recall(), risk_ratio(), roc01(), sensitivity(), specificity(), sum_ppv_npv(), sum_sens_spec(), total_utility(), tpr(), tp(), youden()

Examples

```
cohens_kappa(10, 5, 20, 10)
cohens_kappa(c(10, 8), c(5, 7), c(20, 12), c(10, 18))
```

---

cutpoint *Extract the cutpoints from a ROC curve generated by cutpointr*

Description

This is a utility function for extracting the cutpoints from a roc_cutpointr object. Mainly useful in conjunction with the plot_cutpointr function if cutpoints are to be plotted on the x-axis.

Usage

```
cutpoint(x, ...)

cutpoints(x, ...)
```

Arguments

x	A roc_cutpointr object.
...	Further arguments.

See Also

Other metric functions: F1_score(), Jaccard(), abs_d_ppv_npv(), abs_d_sens_spec(), accuracy(), cohens_kappa(), false_omission_rate(), metric_constrain(), misclassification_cost(), npv(), odds_ratio(), p_chisquared(), plr(), ppv(), precision(), prod_ppv_npv(), prod_sens_spec(), recall(), risk_ratio(), roc01(), sensitivity(), specificity(), sum_ppv_npv(), sum_sens_spec(), total_utility(), tpr(), tp(), youden()

Examples

```
oc <- cutpointr(suicide, dsi, suicide, gender)
plot_cutpointr(oc, cutpoint, accuracy)
```

`cutpointr` *Determine and evaluate optimal cutpoints*

Description

Using predictions (or e.g. biological marker values) and binary class labels, this function will determine "optimal" cutpoints using various selectable methods. The methods for cutpoint determination can be evaluated using bootstrapping. An estimate of the cutpoint variability and the out-of-sample performance can then be returned with `summary` or `plot`. For an introduction to the package please see `vignette("cutpointr",package = "cutpointr")`.

Usage

```
cutpointr(...)

## Default S3 method:
cutpointr(
  data,
  x,
  class,
  subgroup = NULL,
  method = maximize_metric,
  metric = sum_sens_spec,
  pos_class = NULL,
  neg_class = NULL,
  direction = NULL,
  boot_runs = 0,
  boot_stratify = FALSE,
  use_midpoints = FALSE,
  break_ties = median,
  na.rm = FALSE,
  allowParallel = FALSE,
  silent = FALSE,
  tol_metric = 1e-06,
  ...
)

## S3 method for class 'numeric'
cutpointr(
  x,
  class,
  subgroup = NULL,
  method = maximize_metric,
  metric = sum_sens_spec,
  pos_class = NULL,
  neg_class = NULL,
  direction = NULL,
```

```
  boot_runs = 0,
  boot_stratify = FALSE,
  use_midpoints = FALSE,
  break_ties = median,
  na.rm = FALSE,
  allowParallel = FALSE,
  silent = FALSE,
  tol_metric = 1e-06,
  ...
)
```

Arguments

...	Further optional arguments that will be passed to method. minimize_metric and maximize_metric pass ... to metric.
data	A data.frame with the data needed for x, class and optionally subgroup.
x	The variable name to be used for classification, e.g. predictions. The raw vector of values if the data argument is unused.
class	The variable name indicating class membership. If the data argument is unused, the vector of raw numeric values.
subgroup	An additional covariate that identifies subgroups or the raw data if data = NULL. Separate optimal cutpoints will be determined per group. Numeric, character and factor are allowed.
method	(function) A function for determining cutpoints. Can be user supplied or use some of the built in methods. See details.
metric	(function) The function for computing a metric when using maximize_metric or minimize_metric as method and and for the out-of-bag values during bootstrapping. A way of internally validating the performance. User defined functions can be supplied, see details.
pos_class	(optional) The value of class that indicates the positive class.
neg_class	(optional) The value of class that indicates the negative class.
direction	(character, optional) Use ">=" or "<=" to indicate whether x is supposed to be larger or smaller for the positive class.
boot_runs	(numerical) If positive, this number of bootstrap samples will be used to assess the variability and the out-of-sample performance.
boot_stratify	(logical) If the bootstrap is stratified, bootstrap samples are drawn separately in both classes and then combined, keeping the proportion of positives and negatives constant in every resample.
use_midpoints	(logical) If TRUE (default FALSE) the returned optimal cutpoint will be the mean of the optimal cutpoint and the next highest observation (for direction = ">=") or the next lowest observation (for direction = "<=") which avoids biasing the optimal cutpoint.
break_ties	If multiple cutpoints are found, they can be summarized using this function, e.g. mean or median. To return all cutpoints use c as the function.

na.rm	(logical) Set to TRUE (default FALSE) to keep only complete cases of x, class and subgroup (if specified). Missing values with na.rm = FALSE will raise an error.
allowParallel	(logical) If TRUE, the bootstrapping will be parallelized using foreach. A local cluster, for example, should be started manually beforehand.
silent	(logical) If TRUE suppresses all messages.
tol_metric	All cutpoints will be returned that lead to a metric value in the interval [m_max - tol_metric, m_max + tol_metric] where m_max is the maximum achievable metric value. This can be used to return multiple decent cutpoints and to avoid floating-point problems. Not supported by all method functions, see details.

Details

If direction and/or pos_class and neg_class are not given, the function will assume that higher values indicate the positive class and use the class with a higher median as the positive class.

This function uses tidyeval to support unquoted arguments. For programming with cutpointr the operator !! can be used to unquote an argument, see the examples.

Different methods can be selected for determining the optimal cutpoint via the method argument. The package includes the following method functions:

- maximize_metric: Maximize the metric function
- minimize_metric: Minimize the metric function
- maximize_loess_metric: Maximize the metric function after LOESS smoothing
- minimize_loess_metric: Minimize the metric function after LOESS smoothing
- maximize_spline_metric: Maximize the metric function after spline smoothing
- minimize_spline_metric: Minimize the metric function after spline smoothing
- maximize_boot_metric: Maximize the metric function as a summary of the optimal cutpoints in bootstrapped samples
- minimize_boot_metric: Minimize the metric function as a summary of the optimal cutpoints in bootstrapped samples
- oc_youden_kernel: Maximize the Youden-Index after kernel smoothing the distributions of the two classes
- oc_youden_normal: Maximize the Youden-Index parametrically assuming normally distributed data in both classes
- oc_manual: Specify the cutpoint manually

User-defined functions can be supplied to method, too. As a reference, the code of all included method functions can be accessed by simply typing their name. To define a new method function, create a function that may take as input(s):

- data: A data.frame or tbl_df
- x: (character) The name of the predictor or independent variable
- class: (character) The name of the class or dependent variable
- metric_func: A function for calculating a metric, e.g. accuracy

- pos_class: The positive class
- neg_class: The negative class
- direction: ">=" if the positive class has higher x values, "<=" otherwise
- tol_metric: (numeric) In the built-in methods a tolerance around the optimal metric value
- use_midpoints: (logical) In the built-in methods whether to use midpoints instead of exact optimal cutpoints
- ... Further arguments

The ... argument can be used to avoid an error if not all of the above arguments are needed or in order to pass additional arguments to method. The function should return a data.frame or tbl_df with one row, the column "optimal_cutpoint", and an optional column with an arbitrary name with the metric value at the optimal cutpoint.

Built-in metric functions include:

- accuracy: Fraction correctly classified
- youden: Youden- or J-Index = sensitivity + specificity - 1
- sum_sens_spec: sensitivity + specificity
- sum_ppv_npv: The sum of positive predictive value (PPV) and negative predictive value (NPV)
- prod_sens_spec: sensitivity * specificity
- prod_ppv_npv: The product of positive predictive value (PPV) and negative predictive value (NPV)
- cohens_kappa: Cohen's Kappa
- abs_d_sens_spec: The absolute difference between sensitivity and specificity
- roc01: Distance to the point (0,1) on ROC space
- abs_d_ppv_npv: The absolute difference between positive predictive value (PPV) and negative predictive value (NPV)
- p_chisquared: The p-value of a chi-squared test on the confusion matrix of predictions and observations
- odds_ratio: The odds ratio calculated as (TP / FP) / (FN / TN)
- risk_ratio: The risk ratio (relative risk) calculated as (TP / (TP + FN)) / (FP / (FP + TN))
- positive and negative likelihood ratio calculated as plr = true positive rate / false positive rate and nlr = false negative rate / true negative rate
- misclassification_cost: The sum of the misclassification cost of false positives and false negatives fp * cost_fp + fn * cost_fn. Additional arguments to cutpointr: cost_fp, cost_fn
- total_utility: The total utility of true / false positives / negatives calculated as utility_tp * TP + utility_tn * TN - cost_fp * FP - cost_fn * FN. Additional arguments to cutpointr: utility_tp, utility_tn, cost_fp, cost_fn
- F1_score: The F1-score (2 * TP) / (2 * TP + FP + FN)
- sens_constrain: Maximize sensitivity given a minimal value of specificity
- spec_constrain: Maximize specificity given a minimal value of sensitivity

- `metric_constrain`: Maximize a selected metric given a minimal value of another selected metric

Furthermore, the following functions are included which can be used as metric functions but are more useful for plotting purposes, for example in plot_cutpointr, or for defining new metric functions: `tp`, `fp`, `tn`, `fn`, `tpr`, `fpr`, `tnr`, `fnr`, `false_omission_rate`, `false_discovery_rate`, `ppv`, `npv`, `precision`, `recall`, `sensitivity`, and `specificity`.

User defined metric functions can be created as well which can accept the following inputs as vectors:

- `tp`: Vector of true positives
- `fp`: Vector of false positives
- `tn`: Vector of true negatives
- `fn`: Vector of false negatives
- `...` If the metric function is used in conjunction with any of the maximize / minimize methods, further arguments can be passed

The function should return a numeric vector or a matrix or a `data.frame` with one column. If the column is named, the name will be included in the output and plots. Avoid using names that are identical to the column names that are by default returned by **cutpointr**.

If `boot_runs` is positive, that number of bootstrap samples will be drawn and the optimal cutpoint using `method` will be determined. Additionally, as a way of internal validation, the function in `metric` will be used to score the out-of-bag predictions using the cutpoints determined by `method`. Various default metrics are always included in the bootstrap results.

If multiple optimal cutpoints are found, the column optimal_cutpoint becomes a list that contains the vector(s) of the optimal cutpoints.

If `use_midpoints = TRUE` the mean of the optimal cutpoint and the next highest or lowest possible cutpoint is returned, depending on `direction`.

The `tol_metric` argument can be used to avoid floating-point problems that may lead to exclusion of cutpoints that achieve the optimally achievable metric value. Additionally, by selecting a large tolerance multiple cutpoints can be returned that lead to decent metric values in the vicinity of the optimal metric value. `tol_metric` is passed to metric and is only supported by the maximization and minimization functions, i.e. `maximize_metric`, `minimize_metric`, `maximize_loess_metric`, `minimize_loess_metric`, `maximize_spline_metric`, and `minimize_spline_metric`. In `maximize_boot_metric` and `minimize_boot_metric` multiple optimal cutpoints will be passed to the `summary_func` of these two functions.

Value

A cutpointr object which is also a data.frame and tbl_df.

See Also

Other main cutpointr functions: `add_metric()`, `boot_ci()`, `boot_test()`, `multi_cutpointr()`, `predict.cutpointr()`, `roc()`

Examples

```
library(cutpointr)

## Optimal cutpoint for dsi
data(suicide)
opt_cut <- cutpointr(suicide, dsi, suicide)
opt_cut
s_opt_cut <- summary(opt_cut)
plot(opt_cut)

## Not run:
## Predict class for new observations
predict(opt_cut, newdata = data.frame(dsi = 0:5))

## Supplying raw data, same result
cutpointr(x = suicide$dsi, class = suicide$suicide)

## direction, class labels, method and metric can be defined manually
## Again, same result
cutpointr(suicide, dsi, suicide, direction = ">=", pos_class = "yes",
          method = maximize_metric, metric = youden)

## Optimal cutpoint for dsi, as before, but for the separate subgroups
opt_cut <- cutpointr(suicide, dsi, suicide, gender)
opt_cut
(s_opt_cut <- summary(opt_cut))
tibble:::print.tbl(s_opt_cut)

## Bootstrapping also works on individual subgroups
set.seed(30)
opt_cut <- cutpointr(suicide, dsi, suicide, gender, boot_runs = 1000,
  boot_stratify = TRUE)
opt_cut
summary(opt_cut)
plot(opt_cut)

## Parallelized bootstrapping
  library(doParallel)
  library(doRNG)
  cl <- makeCluster(2) # 2 cores
  registerDoParallel(cl)
  registerDoRNG(12) # Reproducible parallel loops using doRNG
  opt_cut <- cutpointr(suicide, dsi, suicide, gender,
                       boot_runs = 1000, allowParallel = TRUE)
  stopCluster(cl)
  opt_cut
  plot(opt_cut)

## Robust cutpoint method using kernel smoothing for optimizing Youden-Index
opt_cut <- cutpointr(suicide, dsi, suicide, gender,
                     method = oc_youden_kernel)
opt_cut
```

```
## End(Not run)
```

`cutpointr_` *The standard evaluation version of cutpointr (deprecated)*

Description

This function is equivalent to `cutpointr` but takes only quoted arguments for `x`, `class` and `subgroup`. This was useful before `cutpointr` supported tidyeval.

Usage

```
cutpointr_(
  data,
  x,
  class,
  subgroup = NULL,
  method = maximize_metric,
  metric = sum_sens_spec,
  pos_class = NULL,
  neg_class = NULL,
  direction = NULL,
  boot_runs = 0,
  boot_stratify = FALSE,
  use_midpoints = FALSE,
  break_ties = median,
  na.rm = FALSE,
  allowParallel = FALSE,
  silent = FALSE,
  tol_metric = 1e-06,
  ...
)
```

Arguments

`data`	A data.frame with the data needed for x, class and optionally subgroup.
`x`	(character) The variable name to be used for classification, e.g. predictions or test values.
`class`	(character) The variable name indicating class membership.
`subgroup`	(character) The variable name of an additional covariate that identifies subgroups. Separate optimal cutpoints will be determined per group.
`method`	(function) A function for determining cutpoints. Can be user supplied or use some of the built in methods. See details.

metric	(function) The function for computing a metric when using maximize_metric or minimize_metric as method and and for the out-of-bag values during bootstrapping. A way of internally validating the performance. User defined functions can be supplied, see details.
pos_class	(optional) The value of class that indicates the positive class.
neg_class	(optional) The value of class that indicates the negative class.
direction	(character, optional) Use ">=" or "<=" to indicate whether x is supposed to be larger or smaller for the positive class.
boot_runs	(numerical) If positive, this number of bootstrap samples will be used to assess the variability and the out-of-sample performance.
boot_stratify	(logical) If the bootstrap is stratified, bootstrap samples are drawn separately in both classes and then combined, keeping the proportion of positives and negatives constant in every resample.
use_midpoints	(logical) If TRUE (default FALSE) the returned optimal cutpoint will be the mean of the optimal cutpoint and the next highest observation (for direction = ">=") or the next lowest observation (for direction = "<=") which avoids biasing the optimal cutpoint.
break_ties	If multiple cutpoints are found, they can be summarized using this function, e.g. mean or median. To return all cutpoints use c as the function.
na.rm	(logical) Set to TRUE (default FALSE) to keep only complete cases of x, class and subgroup (if specified). Missing values with na.rm = FALSE will raise an error.
allowParallel	(logical) If TRUE, the bootstrapping will be parallelized using foreach. A local cluster, for example, should be started manually beforehand.
silent	(logical) If TRUE suppresses all messages.
tol_metric	All cutpoints will be returned that lead to a metric value in the interval [m_max - tol_metric, m_max + tol_metric] where m_max is the maximum achievable metric value. This can be used to return multiple decent cutpoints and to avoid floating-point problems. Not supported by all method functions, see details.
...	Further optional arguments that will be passed to method. minimize_metric and maximize_metric pass ... to metric.

Examples

```
library(cutpointr)

## Optimal cutpoint for dsi
data(suicide)
opt_cut <- cutpointr_(suicide, "dsi", "suicide")
opt_cut
summary(opt_cut)
plot(opt_cut)
predict(opt_cut, newdata = data.frame(dsi = 0:5))
```

cutpoint_knots *Calculate number of knots to use in spline smoothing*

Description

This function calculates the number of knots when using smoothing splines for smoothing a function of metric values per cutpoint value. The function for calculating the number of knots is equal to `stats::.nknots_smspl` but uses the number of unique cutpoints in the data as n.

Usage

```
cutpoint_knots(data, x)
```

Arguments

`data`	A data frame
`x`	(character) The name of the predictor variable

Examples

```
cutpoint_knots(suicide, "dsi")
```

F1_score *Calculate the F1-score*

Description

Calculate the F1-score from true positives, false positives, true negatives and false negatives. The inputs must be vectors of equal length.

F1_score = (2 * tp) / (2 * tp + fp + fn)

Usage

```
F1_score(tp, fp, tn, fn, ...)
```

Arguments

`tp`	(numeric) number of true positives.
`fp`	(numeric) number of false positives.
`tn`	(numeric) number of true negatives.
`fn`	(numeric) number of false negatives.
`...`	for capturing additional arguments passed by method.

See Also

Other metric functions: Jaccard(), abs_d_ppv_npv(), abs_d_sens_spec(), accuracy(), cohens_kappa(), cutpoint(), false_omission_rate(), metric_constrain(), misclassification_cost(), npv(), odds_ratio(), p_chisquared(), plr(), ppv(), precision(), prod_ppv_npv(), prod_sens_spec(), recall(), risk_ratio(), roc01(), sensitivity(), specificity(), sum_ppv_npv(), sum_sens_spec(), total_utility(), tpr(), tp(), youden()

Examples

```
F1_score(10, 5, 20, 10)
F1_score(c(10, 8), c(5, 7), c(20, 12), c(10, 18))
```

---

false_omission_rate *Calculate the false omission and false discovery rate*

Description

Calculate the false omission rate or false discovery rate from true positives, false positives, true negatives and false negatives. The inputs must be vectors of equal length.

false_omission_rate = fn / (tn + fn) = 1 - npv false_discovery_rate = fp / (tp + fp) = 1 - ppv

Usage

```
false_omission_rate(tp, fp, tn, fn, ...)

false_discovery_rate(tp, fp, tn, fn, ...)
```

Arguments

tp	(numeric) number of true positives.
fp	(numeric) number of false positives.
tn	(numeric) number of true negatives.
fn	(numeric) number of false negatives.
...	for capturing additional arguments passed by method.

See Also

Other metric functions: F1_score(), Jaccard(), abs_d_ppv_npv(), abs_d_sens_spec(), accuracy(), cohens_kappa(), cutpoint(), metric_constrain(), misclassification_cost(), npv(), odds_ratio(), p_chisquared(), plr(), ppv(), precision(), prod_ppv_npv(), prod_sens_spec(), recall(), risk_ratio(), roc01(), sensitivity(), specificity(), sum_ppv_npv(), sum_sens_spec(), total_utility(), tpr(), tp(), youden()

Examples

```
false_omission_rate(10, 5, 20, 10)
false_omission_rate(c(10, 8), c(5, 7), c(20, 12), c(10, 18))
```

`Jaccard` *Calculate the Jaccard Index*

Description

Calculate the Jaccard Index from true positives, false positives, true negatives and false negatives. The inputs must be vectors of equal length.

Jaccard = (tp) / (tp + fp + fn)

Usage

```
Jaccard(tp, fp, tn, fn, ...)
```

Arguments

`tp`	(numeric) number of true positives.
`fp`	(numeric) number of false positives.
`tn`	(numeric) number of true negatives.
`fn`	(numeric) number of false negatives.
`...`	for capturing additional arguments passed by method.

See Also

Other metric functions: `F1_score()`, `abs_d_ppv_npv()`, `abs_d_sens_spec()`, `accuracy()`, `cohens_kappa()`, `cutpoint()`, `false_omission_rate()`, `metric_constrain()`, `misclassification_cost()`, `npv()`, `odds_ratio()`, `p_chisquared()`, `plr()`, `ppv()`, `precision()`, `prod_ppv_npv()`, `prod_sens_spec()`, `recall()`, `risk_ratio()`, `roc01()`, `sensitivity()`, `specificity()`, `sum_ppv_npv()`, `sum_sens_spec()`, `total_utility()`, `tpr()`, `tp()`, `youden()`

Examples

```
Jaccard(10, 5, 20, 10)
Jaccard(c(10, 8), c(5, 7), c(20, 12), c(10, 18))
```

maximize_boot_metric *Optimize a metric function in binary classification after bootstrapping*

Description

Given a function for computing a metric in `metric_func`, these functions bootstrap the data `boot_cut` times and maximize or minimize the metric by selecting an optimal cutpoint. The returned optimal cutpoint is the result of applying `summary_func`, e.g. the mean, to all optimal cutpoints that were determined in the bootstrap samples. The `metric` function should accept the following inputs:

- `tp`: vector of number of true positives
- `fp`: vector of number of false positives
- `tn`: vector of number of true negatives
- `fn`: vector of number of false negatives

Usage

```
maximize_boot_metric(
  data,
  x,
  class,
  metric_func = youden,
  pos_class = NULL,
  neg_class = NULL,
  direction,
  summary_func = mean,
  boot_cut = 50,
  boot_stratify,
  inf_rm = TRUE,
  tol_metric,
  use_midpoints,
  ...
)

minimize_boot_metric(
  data,
  x,
  class,
  metric_func = youden,
  pos_class = NULL,
  neg_class = NULL,
  direction,
  summary_func = mean,
  boot_cut = 50,
  boot_stratify,
  inf_rm = TRUE,
```

```
  tol_metric,
  use_midpoints,
  ...
)
```

Arguments

data	A data frame or tibble in which the columns that are given in x and class can be found.
x	(character) The variable name to be used for classification, e.g. predictions or test values.
class	(character) The variable name indicating class membership.
metric_func	(function) A function that computes a single number metric to be maximized. See description.
pos_class	The value of class that indicates the positive class.
neg_class	The value of class that indicates the negative class.
direction	(character) Use ">=" or "<=" to select whether an x value >= or <= the cutoff predicts the positive class.
summary_func	(function) After obtaining the bootstrapped optimal cutpoints this function, e.g. mean or median, is applied to arrive at a single cutpoint.
boot_cut	(numeric) Number of bootstrap repetitions over which the mean optimal cutpoint is calculated.
boot_stratify	(logical) If the bootstrap is stratified, bootstrap samples are drawn in both classes and then combined, keeping the number of positives and negatives constant in every resample.
inf_rm	(logical) whether to remove infinite cutpoints before calculating the summary.
tol_metric	All cutpoints will be passed to summary_func that lead to a metric value in the interval [m_max - tol_metric, m_max + tol_metric] where m_max is the maximum achievable metric value. This can be used to return multiple decent cutpoints and to avoid floating-point problems.
use_midpoints	(logical) If TRUE (default FALSE) the returned optimal cutpoint will be the mean of the optimal cutpoint and the next highest observation (for direction = ">") or the next lowest observation (for direction = "<") which avoids biasing the optimal cutpoint.
...	To capture further arguments that are always passed to the method function by cutpointr. The cutpointr function passes data, x, class, metric_func, direction, pos_class and neg_class to the method function.

Details

The above inputs are arrived at by using all unique values in x, Inf, and -Inf as possible cutpoints for classifying the variable in class. The reported metric represents the usual in-sample performance of the determined cutpoint.

Value

A tibble with the column `optimal_cutpoint`

See Also

Other method functions: `maximize_gam_metric()`, `maximize_loess_metric()`, `maximize_metric()`, `maximize_spline_metric()`, `oc_manual()`, `oc_mean()`, `oc_median()`, `oc_youden_kernel()`, `oc_youden_normal()`

Examples

```
set.seed(100)
cutpointr(suicide, dsi, suicide, method = maximize_boot_metric,
          metric = accuracy, boot_cut = 30)
set.seed(100)
cutpointr(suicide, dsi, suicide, method = minimize_boot_metric,
          metric = abs_d_sens_spec, boot_cut = 30)
```

`maximize_gam_metric` *Optimize a metric function in binary classification after smoothing via generalized additive models*

Description

Given a function for computing a metric in `metric_func`, these functions smooth the function of metric value per cutpoint using generalized additive models (as implemented in **mgcv**), then maximize or minimize the metric by selecting an optimal cutpoint. For further details on the GAM smoothing see `?mgcv::gam`. The `metric` function should accept the following inputs:

- `tp`: vector of number of true positives
- `fp`: vector of number of false positives
- `tn`: vector of number of true negatives
- `fn`: vector of number of false negatives

Usage

```
maximize_gam_metric(
  data,
  x,
  class,
  metric_func = youden,
  pos_class = NULL,
  neg_class = NULL,
  direction,
  formula = m ~ s(x.sorted),
  optimizer = c("outer", "newton"),
  tol_metric,
```

```
  use_midpoints,
  ...
)

minimize_gam_metric(
  data,
  x,
  class,
  metric_func = youden,
  pos_class = NULL,
  neg_class = NULL,
  direction,
  formula = m ~ s(x.sorted),
  optimizer = c("outer", "newton"),
  tol_metric,
  use_midpoints,
  ...
)
```

Arguments

data	A data frame or tibble in which the columns that are given in x and class can be found.
x	(character) The variable name to be used for classification, e.g. predictions or test values.
class	(character) The variable name indicating class membership.
metric_func	(function) A function that computes a metric to be maximized. See description.
pos_class	The value of class that indicates the positive class.
neg_class	The value of class that indicates the negative class.
direction	(character) Use ">=" or "<=" to select whether an x value >= or <= the cutoff predicts the positive class.
formula	A GAM formula. See `help("gam",package = "mgcv")` for details.
optimizer	An array specifying the numerical optimization method to use to optimize the smoothing parameter estimation criterion (given by method). See `help("gam",package = "mgcv")` for details.
tol_metric	All cutpoints will be returned that lead to a metric value in the interval [m_max - tol_metric, m_max + tol_metric] where m_max is the maximum achievable metric value. This can be used to return multiple decent cutpoints and to avoid floating-point problems.
use_midpoints	(logical) If TRUE (default FALSE) the returned optimal cutpoint will be the mean of the optimal cutpoint and the next highest observation (for direction = ">") or the next lowest observation (for direction = "<") which avoids biasing the optimal cutpoint.
...	Further arguments that will be passed to metric_func or the GAM smoother.

Details

The above inputs are arrived at by using all unique values in x, Inf, and -Inf as possible cutpoints for classifying the variable in class.

Value

A tibble with the columns optimal_cutpoint, the corresponding metric value and roc_curve, a nested tibble that includes all possible cutoffs and the corresponding numbers of true and false positives / negatives and all corresponding metric values.

See Also

Other method functions: maximize_boot_metric(), maximize_loess_metric(), maximize_metric(), maximize_spline_metric(), oc_manual(), oc_mean(), oc_median(), oc_youden_kernel(), oc_youden_normal()

Examples

```
oc <- cutpointr(suicide, dsi, suicide, gender, method = maximize_gam_metric,
metric = accuracy)
plot_metric(oc)
oc <- cutpointr(suicide, dsi, suicide, gender, method = minimize_gam_metric,
metric = abs_d_sens_spec)
plot_metric(oc)
```

---

maximize_loess_metric *Optimize a metric function in binary classification after LOESS smoothing*

Description

Given a function for computing a metric in metric_func, these functions smooth the function of metric value per cutpoint using LOESS, then maximize or minimize the metric by selecting an optimal cutpoint. For further details on the LOESS smoothing see ?fANCOVA::loess.as. The metric function should accept the following inputs:

- tp: vector of number of true positives
- fp: vector of number of false positives
- tn: vector of number of true negatives
- fn: vector of number of false negatives

Usage

```
maximize_loess_metric(
  data,
  x,
  class,
  metric_func = youden,
  pos_class = NULL,
  neg_class = NULL,
  direction,
  criterion = "aicc",
  degree = 1,
  family = "symmetric",
  user.span = NULL,
  tol_metric,
  use_midpoints,
  ...
)

minimize_loess_metric(
  data,
  x,
  class,
  metric_func = youden,
  pos_class = NULL,
  neg_class = NULL,
  direction,
  criterion = "aicc",
  degree = 1,
  family = "symmetric",
  user.span = NULL,
  tol_metric,
  use_midpoints,
  ...
)
```

Arguments

data	A data frame or tibble in which the columns that are given in x and class can be found.
x	(character) The variable name to be used for classification, e.g. predictions or test values.
class	(character) The variable name indicating class membership.
metric_func	(function) A function that computes a metric to be maximized. See description.
pos_class	The value of class that indicates the positive class.
neg_class	The value of class that indicates the negative class.
direction	(character) Use ">=" or "<=" to select whether an x value >= or <= the cutoff predicts the positive class.

criterion	the criterion for automatic smoothing parameter selection: "aicc" denotes bias-corrected AIC criterion, "gcv" denotes generalized cross-validation.
degree	the degree of the local polynomials to be used. It can be 0, 1 or 2.
family	if "gaussian" fitting is by least-squares, and if "symmetric" a re-descending M estimator is used with Tukey's biweight function.
user.span	The user-defined parameter which controls the degree of smoothing
tol_metric	All cutpoints will be returned that lead to a metric value in the interval [m_max - tol_metric, m_max + tol_metric] where m_max is the maximum achievable metric value. This can be used to return multiple decent cutpoints and to avoid floating-point problems.
use_midpoints	(logical) If TRUE (default FALSE) the returned optimal cutpoint will be the mean of the optimal cutpoint and the next highest observation (for direction = ">") or the next lowest observation (for direction = "<") which avoids biasing the optimal cutpoint.
...	Further arguments that will be passed to metric_func or the loess smoother.

Details

The above inputs are arrived at by using all unique values in x, Inf, and -Inf as possible cutpoints for classifying the variable in class.

Value

A tibble with the columns optimal_cutpoint, the corresponding metric value and roc_curve, a nested tibble that includes all possible cutoffs and the corresponding numbers of true and false positives / negatives and all corresponding metric values.

Source

Xiao-Feng Wang (2010). fANCOVA: Nonparametric Analysis of Covariance. https://CRAN.R-project.org/package=fANCOVA

Leeflang, M. M., Moons, K. G., Reitsma, J. B., & Zwinderman, A. H. (2008). Bias in sensitivity and specificity caused by data-driven selection of optimal cutoff values: mechanisms, magnitude, and solutions. Clinical Chemistry, (4), 729–738.

See Also

Other method functions: maximize_boot_metric(), maximize_gam_metric(), maximize_metric(), maximize_spline_metric(), oc_manual(), oc_mean(), oc_median(), oc_youden_kernel(), oc_youden_normal()

Examples

```
oc <- cutpointr(suicide, dsi, suicide, gender, method = maximize_loess_metric,
criterion = "aicc", family = "symmetric", degree = 2, user.span = 0.7,
metric = accuracy)
plot_metric(oc)
oc <- cutpointr(suicide, dsi, suicide, gender, method = minimize_loess_metric,
```

```
criterion = "aicc", family = "symmetric", degree = 2, user.span = 0.7,
metric = misclassification_cost, cost_fp = 1, cost_fn = 10)
plot_metric(oc)
```

`maximize_metric` *Optimize a metric function in binary classification*

Description

Given a function for computing a metric in `metric_func`, these functions maximize or minimize that metric by selecting an optimal cutpoint. The metric function should accept the following inputs:

- `tp`: vector of number of true positives
- `fp`: vector of number of false positives
- `tn`: vector of number of true negatives
- `fn`: vector of number of false negatives

Usage

```
maximize_metric(
  data,
  x,
  class,
  metric_func = youden,
  pos_class = NULL,
  neg_class = NULL,
  direction,
  tol_metric,
  use_midpoints,
  ...
)

minimize_metric(
  data,
  x,
  class,
  metric_func = youden,
  pos_class = NULL,
  neg_class = NULL,
  direction,
  tol_metric,
  use_midpoints,
  ...
)
```

Arguments

data	A data frame or tibble in which the columns that are given in x and class can be found.
x	(character) The variable name to be used for classification, e.g. predictions or test values.
class	(character) The variable name indicating class membership.
metric_func	(function) A function that computes a metric to be maximized. See description.
pos_class	The value of class that indicates the positive class.
neg_class	The value of class that indicates the negative class.
direction	(character) Use ">=" or "<=" to select whether an x value >= or <= the cutoff predicts the positive class.
tol_metric	All cutpoints will be returned that lead to a metric value in the interval [m_max - tol_metric, m_max + tol_metric] where m_max is the maximum achievable metric value. This can be used to return multiple decent cutpoints and to avoid floating-point problems.
use_midpoints	(logical) If TRUE (default FALSE) the returned optimal cutpoint will be the mean of the optimal cutpoint and the next highest observation (for direction = ">") or the next lowest observation (for direction = "<") which avoids biasing the optimal cutpoint.
...	Further arguments that will be passed to metric_func.

Details

The above inputs are arrived at by using all unique values in x, Inf, or -Inf as possible cutpoints for classifying the variable in class.

Value

A tibble with the columns optimal_cutpoint, the corresponding metric value and roc_curve, a nested tibble that includes all possible cutoffs and the corresponding numbers of true and false positives / negatives and all corresponding metric values.

See Also

Other method functions: maximize_boot_metric(), maximize_gam_metric(), maximize_loess_metric(), maximize_spline_metric(), oc_manual(), oc_mean(), oc_median(), oc_youden_kernel(), oc_youden_normal()

Examples

```
cutpointr(suicide, dsi, suicide, method = maximize_metric, metric = accuracy)
cutpointr(suicide, dsi, suicide, method = minimize_metric, metric = abs_d_sens_spec)
```

`maximize_spline_metric` *Optimize a metric function in binary classification after spline smoothing*

Description

Given a function for computing a metric in `metric_func`, this function smoothes the function of metric value per cutpoint using smoothing splines. Then it optimizes the metric by selecting an optimal cutpoint. For further details on the smoothing spline see `?stats::smooth.spline`. The `metric` function should accept the following inputs:

- `tp`: vector of number of true positives
- `fp`: vector of number of false positives
- `tn`: vector of number of true negatives
- `fn`: vector of number of false negatives

Usage

```
maximize_spline_metric(
  data,
  x,
  class,
  metric_func = youden,
  pos_class = NULL,
  neg_class = NULL,
  direction,
  w = NULL,
  df = NULL,
  spar = 1,
  nknots = cutpoint_knots,
  df_offset = NULL,
  penalty = 1,
  control_spar = list(),
  tol_metric,
  use_midpoints,
  ...
)

minimize_spline_metric(
  data,
  x,
  class,
  metric_func = youden,
  pos_class = NULL,
  neg_class = NULL,
```

```
  direction,
  w = NULL,
  df = NULL,
  spar = 1,
  nknots = cutpoint_knots,
  df_offset = NULL,
  penalty = 1,
  control_spar = list(),
  tol_metric,
  use_midpoints,
  ...
)
```

Arguments

data	A data frame or tibble in which the columns that are given in x and class can be found.
x	(character) The variable name to be used for classification, e.g. predictions or test values.
class	(character) The variable name indicating class membership.
metric_func	(function) A function that computes a metric to be optimized. See description.
pos_class	The value of class that indicates the positive class.
neg_class	The value of class that indicates the negative class.
direction	(character) Use ">=" or "<=" to select whether an x value >= or <= the cutoff predicts the positive class.
w	Optional vector of weights of the same length as x; defaults to all 1.
df	The desired equivalent number of degrees of freedom (trace of the smoother matrix). Must be in (1,nx], nx the number of unique x values.
spar	Smoothing parameter, typically (but not necessarily) in (0,1]. When spar is specified, the coefficient lambda of the integral of the squared second derivative in the fit (penalized log likelihood) criterion is a monotone function of spar.
nknots	Integer or function giving the number of knots. The function should accept data and x (the name of the predictor variable) as inputs. By default nknots = 0.1 * log(n_dat / n_cut) * n_cut where n_dat is the number of observations and n_cut the number of unique predictor values.
df_offset	Allows the degrees of freedom to be increased by df_offset in the GCV criterion.
penalty	The coefficient of the penalty for degrees of freedom in the GCV criterion.
control_spar	Optional list with named components controlling the root finding when the smoothing parameter spar is computed, i.e., NULL. See help("smooth.spline") for further information.
tol_metric	All cutpoints will be returned that lead to a metric value in the interval [m_max - tol_metric, m_max + tol_metric] where m_max is the maximum achievable metric value. This can be used to return multiple decent cutpoints and to avoid floating-point problems.

use_midpoints (logical) If TRUE (default FALSE) the returned optimal cutpoint will be the mean of the optimal cutpoint and the next highest observation (for direction = ">") or the next lowest observation (for direction = "<") which avoids biasing the optimal cutpoint.

... Further arguments that will be passed to metric_func.

Details

The above inputs are arrived at by using all unique values in x, Inf, and -Inf as possible cutpoints for classifying the variable in class.

Value

A tibble with the columns optimal_cutpoint, the corresponding metric value and roc_curve, a nested tibble that includes all possible cutoffs and the corresponding numbers of true and false positives / negatives and all corresponding metric values.

See Also

Other method functions: maximize_boot_metric(), maximize_gam_metric(), maximize_loess_metric(), maximize_metric(), oc_manual(), oc_mean(), oc_median(), oc_youden_kernel(), oc_youden_normal()

Examples

```
oc <- cutpointr(suicide, dsi, suicide, gender, method = maximize_spline_metric,
df = 5, metric = accuracy)
plot_metric(oc)
```

---

metric_constrain *Metrics that are constrained by another metric*

Description

For example, calculate sensitivity where a lower bound (minimal desired value) for specificty can be defined. All returned metric values for cutpoints that lead to values of the constraining metric below the specified minimum will be zero. The inputs must be vectors of equal length.

Usage

```
metric_constrain(
  tp,
  fp,
  tn,
  fn,
  main_metric = sensitivity,
  constrain_metric = specificity,
  min_constrain = 0.5,
  suffix = "_constrain",
```

```
  ...
)

sens_constrain(
  tp,
  fp,
  tn,
  fn,
  constrain_metric = specificity,
  min_constrain = 0.5,
  ...
)

spec_constrain(
  tp,
  fp,
  tn,
  fn,
  constrain_metric = sensitivity,
  min_constrain = 0.5,
  ...
)

acc_constrain(
  tp,
  fp,
  tn,
  fn,
  constrain_metric = sensitivity,
  min_constrain = 0.5,
  ...
)
```

Arguments

tp	(numeric) number of true positives.
fp	(numeric) number of false positives.
tn	(numeric) number of true negatives.
fn	(numeric) number of false negatives.
main_metric	Metric to be optimized.
constrain_metric	Metric for constraint.
min_constrain	Minimum desired value of constrain_metric.
suffix	Character string to be added to the name of main_metric.
...	for capturing additional arguments passed by method.

See Also

Other metric functions: F1_score(), Jaccard(), abs_d_ppv_npv(), abs_d_sens_spec(), accuracy(), cohens_kappa(), cutpoint(), false_omission_rate(), misclassification_cost(), npv(), odds_ratio(), p_chisquared(), plr(), ppv(), precision(), prod_ppv_npv(), prod_sens_spec(), recall(), risk_ratio(), roc01(), sensitivity(), specificity(), sum_ppv_npv(), sum_sens_spec(), total_utility(), tpr(), tp(), youden()

Examples

```
## Maximum sensitivity when Positive Predictive Value (PPV) is at least 75%
library(dplyr)
library(purrr)
library(cutpointr)
cp <- cutpointr(data = suicide, x = dsi, class = suicide,
method = maximize_metric,
metric = sens_constrain,
constrain_metric = ppv,
min_constrain = 0.75)
## All metric values (m) where PPV < 0.75 are zero
plot_metric(cp)
cp$roc_curve
## We can confirm that PPV is indeed >= 0.75
cp %>%
    add_metric(list(ppv))
## We can also do so for the complete ROC curve(s)
cp %>%
    pull(roc_curve) %>%
    map(~ add_metric(., list(sensitivity, ppv)))

## Use the metric_constrain function for a combination of any two metrics
## Estimate optimal cutpoint for precision given a recall of at least 70%
cp <- cutpointr(data = suicide, x = dsi, class = suicide,
                subgroup = gender,
                method = maximize_metric,
                metric = metric_constrain,
                main_metric = precision,
                suffix = "_constrained",
                constrain_metric = recall,
                min_constrain = 0.70)
## All metric values (m) where recall < 0.7 are zero
plot_metric(cp)
## We can confirm that recall is indeed >= 0.70 and that precision_constrain
## is identical to precision for the estimated cutpoint
cp %>%
    add_metric(list(recall, precision))
## We can also do so for the complete ROC curve(s)
cp %>%
    pull(roc_curve) %>%
    map(~ add_metric(., list(recall, precision)))
```

misclassification_cost *Calculate the misclassification cost*

Description

Calculate the misclassification cost from true positives, false positives, true negatives and false negatives. The inputs must be vectors of equal length.

misclassification_cost = cost_fp * fp + cost_fn * fn

Usage

```
misclassification_cost(tp, fp, tn, fn, cost_fp = 1, cost_fn = 1, ...)
```

Arguments

tp	(numeric) number of true positives.
fp	(numeric) number of false positives.
tn	(numeric) number of true negatives.
fn	(numeric) number of false negatives.
cost_fp	(numeric) the cost of a false positive
cost_fn	(numeric) the cost of a false negative
...	for capturing additional arguments passed by method.

See Also

Other metric functions: F1_score(), Jaccard(), abs_d_ppv_npv(), abs_d_sens_spec(), accuracy(), cohens_kappa(), cutpoint(), false_omission_rate(), metric_constrain(), npv(), odds_ratio(), p_chisquared(), plr(), ppv(), precision(), prod_ppv_npv(), prod_sens_spec(), recall(), risk_ratio(), roc01(), sensitivity(), specificity(), sum_ppv_npv(), sum_sens_spec(), total_utility(), tpr(), tp(), youden()

Examples

```
misclassification_cost(10, 5, 20, 10, cost_fp = 1, cost_fn = 5)
misclassification_cost(c(10, 8), c(5, 7), c(20, 12), c(10, 18),
                       cost_fp = 1, cost_fn = 5)
```

`multi_cutpointr` *Calculate optimal cutpoints and further statistics for multiple predictors*

Description

Runs `cutpointr` over multiple predictor variables. Tidyeval via `!!` is supported for `class` and `subgroup`. If `x = NULL`, `cutpointr` will be run using all numeric columns in the data set as predictors except for the variable in `class` and, if given, `subgroup`.

Usage

```
multi_cutpointr(data, x = NULL, class, subgroup = NULL, silent = FALSE, ...)
```

Arguments

`data`	A data frame.
`x`	Character vector of predictor variables. If NULL all numeric columns.
`class`	The name of the outcome / independent variable.
`subgroup`	An additional covariate that identifies subgroups. Separate optimal cutpoints will be determined per group.
`silent`	Whether to suppress messages.
`...`	Further arguments to be passed to cutpointr, e.g., boot_runs

Details

The automatic determination of positive / negative classes and `direction` will be carried out separately for every predictor variable. That way, if `direction` and the classes are not specified, the reported AUC for every variable will be >= 0.5. AUC may be < 0.5 if subgroups are specified as `direction` is equal within every subgroup.

Value

A data frame.

See Also

Other main cutpointr functions: `add_metric()`, `boot_ci()`, `boot_test()`, `cutpointr()`, `predict.cutpointr()`, `roc()`

Examples

```
library(cutpointr)

multi_cutpointr(suicide, x = c("age", "dsi"), class = suicide,
                pos_class = "yes")
```

```
mcp <- multi_cutpointr(suicide, x = c("age", "dsi"), class = suicide,
                       subgroup = gender, pos_class = "yes")
mcp

(scp <- summary(mcp))
## Not run:
## The result is a data frame
tibble:::print.tbl(scp)

## End(Not run)
```

---

npv *Calculate the negative predictive value*

Description

Calculate the negative predictive value (NPV) from true positives, false positives, true negatives and false negatives. The inputs must be vectors of equal length.

npv = tn / (tn + fn)

Usage

```
npv(tp, fp, tn, fn, ...)
```

Arguments

tp	(numeric) number of true positives.
fp	(numeric) number of false positives.
tn	(numeric) number of true negatives.
fn	(numeric) number of false negatives.
...	for capturing additional arguments passed by method.

See Also

Other metric functions: F1_score(), Jaccard(), abs_d_ppv_npv(), abs_d_sens_spec(), accuracy(), cohens_kappa(), cutpoint(), false_omission_rate(), metric_constrain(), misclassification_cost(), odds_ratio(), p_chisquared(), plr(), ppv(), precision(), prod_ppv_npv(), prod_sens_spec(), recall(), risk_ratio(), roc01(), sensitivity(), specificity(), sum_ppv_npv(), sum_sens_spec(), total_utility(), tpr(), tp(), youden()

Examples

```
npv(10, 5, 20, 10)
npv(c(10, 8), c(5, 7), c(20, 12), c(10, 18))
```

oc_manual *Set a manual cutpoint for use with cutpointr*

Description

This function simply returns `cutpoint` as the optimal cutpoint. Mainly useful if bootstrap estimates of the out-of-bag performance of a given cutpoint are desired, e.g. taking a cutpoint value from the literature.

Usage

```
oc_manual(cutpoint, ...)
```

Arguments

`cutpoint`	(numeric) The fixed cutpoint.
`...`	To capture further arguments that are always passed to the method function by cutpointr. The cutpointr function passes data, x, class, metric_func, direction, pos_class and neg_class to the method function.

See Also

Other method functions: `maximize_boot_metric()`, `maximize_gam_metric()`, `maximize_loess_metric()`, `maximize_metric()`, `maximize_spline_metric()`, `oc_mean()`, `oc_median()`, `oc_youden_kernel()`, `oc_youden_normal()`

Examples

```
cutpointr(suicide, dsi, suicide, method = oc_manual, cutpoint = 4)
```

oc_mean *Use the sample mean as cutpoint*

Description

The sample mean is calculated and returned as the optimal cutpoint.

Usage

```
oc_mean(data, x, trim = 0, ...)
```

Arguments

data	A data frame or tibble in which the columns that are given in x and class can be found.
x	(character) The variable name to be used for classification, e.g. predictions or test values.
trim	The fraction (0 to 0.5) of observations to be trimmed from each end of x before the mean is computed. Values of trim outside that range are taken as the nearest endpoint.
...	To capture further arguments that are always passed to the method function by cutpointr. The cutpointr function passes data, x, class, metric_func, direction, pos_class and neg_class to the method function.

See Also

Other method functions: maximize_boot_metric(), maximize_gam_metric(), maximize_loess_metric(), maximize_metric(), maximize_spline_metric(), oc_manual(), oc_median(), oc_youden_kernel(), oc_youden_normal()

Examples

```
data(suicide)
oc_mean(suicide, "dsi")
cutpointr(suicide, dsi, suicide, method = oc_mean)
```

---

oc_median *Use the sample median as cutpoint*

Description

The sample median is calculated and returned as the optimal cutpoint.

Usage

```
oc_median(data, x, ...)
```

Arguments

data	A data frame or tibble in which the columns that are given in x and class can be found.
x	(character) The variable name to be used for classification, e.g. predictions or test values.
...	To capture further arguments that are always passed to the method function by cutpointr. The cutpointr function passes data, x, class, metric_func, direction, pos_class and neg_class to the method function.

See Also

Other method functions: maximize_boot_metric(), maximize_gam_metric(), maximize_loess_metric(), maximize_metric(), maximize_spline_metric(), oc_manual(), oc_mean(), oc_youden_kernel(), oc_youden_normal()

Examples

```
data(suicide)
oc_median(suicide, "dsi")
cutpointr(suicide, dsi, suicide, method = oc_median)
```

`oc_youden_kernel` *Determine an optimal cutpoint maximizing the Youden-Index based on kernel smoothed densities*

Description

Instead of searching for an optimal cutpoint to maximize (sensitivity + specificity - 1) on the ROC curve, this function first smoothes the empirical distributions of x per class. The smoothing is done using a binned kernel density estimate. The bandwidth is automatically selected using the direct plug-in method.

Usage

```
oc_youden_kernel(data, x, class, pos_class, neg_class, direction, ...)
```

Arguments

`data`	A data frame or tibble in which the columns that are given in x and class can be found.
`x`	(character) The variable name to be used for classification, e.g. predictions or test values.
`class`	(character) The variable name indicating class membership.
`pos_class`	The value of class that indicates the positive class.
`neg_class`	The value of class that indicates the negative class.
`direction`	(character) Use ">=" or "<=" to select whether an x value >= or <= the cutoff predicts the positive class.
`...`	To capture further arguments that are always passed to the method function by cutpointr. The cutpointr function passes data, x, class, metric_func, direction, pos_class and neg_class to the method function.

Details

The functions for calculating the kernel density estimate and the bandwidth are both from **KernSmooth** with default parameters, except for the bandwidth selection, which uses the standard deviation as scale estimate.

The cutpoint is estimated as the cutpoint that maximizes the Youden-Index given by $J = max_c F_N(c) - G_N(c)$ where J and G are the smoothed distribution functions.

Source

Fluss, R., Faraggi, D., & Reiser, B. (2005). Estimation of the Youden Index and its associated cutoff point. Biometrical Journal, 47(4), 458–472.

Matt Wand (2015). KernSmooth: Functions for Kernel Smoothing Supporting Wand & Jones (1995). R package version 2.23-15. https://CRAN.R-project.org/package=KernSmooth

See Also

Other method functions: `maximize_boot_metric()`, `maximize_gam_metric()`, `maximize_loess_metric()`, `maximize_metric()`, `maximize_spline_metric()`, `oc_manual()`, `oc_mean()`, `oc_median()`, `oc_youden_normal()`

Examples

```
data(suicide)
if (require(KernSmooth)) {
  oc_youden_kernel(suicide, "dsi", "suicide", oc_metric = "Youden",
  pos_class = "yes", neg_class = "no", direction = ">=")
  ## Within cutpointr
  cutpointr(suicide, dsi, suicide, method = oc_youden_kernel)
}
```

`oc_youden_normal` *Determine an optimal cutpoint for the Youden-Index assuming normal distributions*

Description

An optimal cutpoint maximizing the Youden- or J-Index (sensitivity + specificity - 1) is calculated parametrically assuming normal distributions per class.

Usage

```
oc_youden_normal(
  data,
  x,
  class,
  pos_class = NULL,
  neg_class = NULL,
  direction,
  ...
)
```

Arguments

`data`	A data frame or tibble in which the columns that are given in x and class can be found.

x	(character) The variable name to be used for classification, e.g. predictions or test values.
class	(character) The variable name indicating class membership.
pos_class	The value of class that indicates the positive class.
neg_class	The value of class that indicates the negative class.
direction	(character) Use ">=" or "<=" to select whether an x value >= or <= the cutoff predicts the positive class.
...	To capture further arguments that are always passed to the method function by cutpointr. The cutpointr function passes data, x, class, metric_func, direction, pos_class and neg_class to the method function.

See Also

Other method functions: maximize_boot_metric(), maximize_gam_metric(), maximize_loess_metric(), maximize_metric(), maximize_spline_metric(), oc_manual(), oc_mean(), oc_median(), oc_youden_kernel()

Examples

```
data(suicide)
oc_youden_normal(suicide, "dsi", "suicide",
  pos_class = "yes", neg_class = "no", direction = ">=")
cutpointr(suicide, dsi, suicide, method = oc_youden_normal)
```

---

odds_ratio *Calculate the odds ratio*

Description

Calculate the (diagnostic) odds ratio from true positives, false positives, true negatives and false negatives. The inputs must be vectors of equal length.

odds_ratio = (tp / fp) / (fn / tn)

Usage

```
odds_ratio(tp, fp, tn, fn, ...)
```

Arguments

tp	(numeric) number of true positives.
fp	(numeric) number of false positives.
tn	(numeric) number of true negatives.
fn	(numeric) number of false negatives.
...	for capturing additional arguments passed by method.

See Also

Other metric functions: F1_score(), Jaccard(), abs_d_ppv_npv(), abs_d_sens_spec(), accuracy(), cohens_kappa(), cutpoint(), false_omission_rate(), metric_constrain(), misclassification_cost(), npv(), p_chisquared(), plr(), ppv(), precision(), prod_ppv_npv(), prod_sens_spec(), recall(), risk_ratio(), roc01(), sensitivity(), specificity(), sum_ppv_npv(), sum_sens_spec(), total_utility(), tpr(), tp(), youden()

Examples

```
odds_ratio(10, 5, 20, 10)
odds_ratio(c(10, 8), c(5, 7), c(20, 12), c(10, 18))
```

---

plot.cutpointr *Plot cutpointr objects*

Description

The plot layout depends on whether subgroups were defined and whether bootstrapping was run.

Usage

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

Arguments

x	A cutpointr object.
...	Further arguments.

Details

The ... argument can be used to apply **ggplot2** functions to every individual plot, for example for changing the theme.

See Also

Other cutpointr plotting functions: plot_cut_boot(), plot_cutpointr(), plot_metric_boot(), plot_metric(), plot_precision_recall(), plot_roc(), plot_sensitivity_specificity(), plot_x()

Examples

```
opt_cut <- cutpointr(suicide, dsi, suicide, gender)
plot(opt_cut)
plot(opt_cut, ggplot2::theme_bw())
```

plot.multi_cutpointr *Plotting multi_cutpointr objects is currently not supported*

Description

You can try plotting the data manually instead.

Usage

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

Arguments

x	A multi_cutpointr object.
...	Further arguments.

plot.roc_cutpointr *Plot ROC curve from a cutpointr or roc_cutpointr object*

Description

Given a cutpointr object this function plots the ROC curve(s) per subgroup, if given. Also plots a ROC curve from the output of roc().

Usage

```
## S3 method for class 'roc_cutpointr'
plot(x, type = "line", ...)
```

Arguments

x	A cutpointr or roc_cutpointr object.
type	"line" for line plot (default) or "step" for step plot.
...	Additional arguments (unused).

See Also

Other cutpointr plotting functions: plot.cutpointr(), plot_cut_boot(), plot_cutpointr(), plot_metric_boot(), plot_metric(), plot_precision_recall(), plot_sensitivity_specificity(), plot_x()

Examples

```
opt_cut <- cutpointr(suicide, dsi, suicide)
plot_roc(opt_cut, display_cutpoint = FALSE)

opt_cut_2groups <- cutpointr(suicide, dsi, suicide, gender)
plot_roc(opt_cut_2groups, display_cutpoint = TRUE)

roc_curve <- roc(suicide, x = dsi, class = suicide, pos_class = "yes",
  neg_class = "no", direction = ">=")
plot(roc_curve)
auc(roc_curve)
```

---

plot_cutpointr *General purpose plotting function for cutpointr or roc_cutpointr objects*

Description

Flexibly plot various metrics against all cutpoints or any other metric. The function can plot any metric based on a `cutpointr` or `roc_cutpointr` object. If `cutpointr` was run with bootstrapping, bootstrapped confidence intervals can be plotted. These represent the quantiles of the distribution of the y-variable grouped by x-variable over all bootstrap repetitions.

Usage

```
plot_cutpointr(
  x,
  xvar = cutpoint,
  yvar = sum_sens_spec,
  conf_lvl = 0.95,
  aspect_ratio = NULL
)
```

Arguments

`x`	A `cutpointr` or `roc_cutpointr` object.
`xvar`	A function, typically `cutpoint` or a metric function.
`yvar`	A function, typically a metric function.
`conf_lvl`	(numeric) If bootstrapping was run and x is a cutpointr object, a confidence interval at the level of conf_lvl can be plotted. To plot no confidence interval set conf_lvl = 0.
`aspect_ratio`	(numeric) Set to 1 to obtain a quadratic plot, e.g. for plotting a ROC curve.

Details

The arguments to xvar and yvar should be metric functions. Any metric function that is suitable for cutpointr can also be used in plot_cutpointr. Anonymous functions are also allowed. To plot all possible cutpoints, the utility function cutpoint can be used.

The functions for xvar and yvar may accept any or all of the arguments tp, fp, tn, or fn and return a numeric vector, a matrix or a data.frame. For more details on metric functions see vignette("cutpointr").

Note that confidence intervals can only be correctly plotted if the values of xvar are constant across bootstrap samples. For example, confidence intervals for tpr by fpr (a ROC curve) cannot be plotted, as the values of the false positive rate vary per bootstrap sample.

See Also

Other cutpointr plotting functions: plot.cutpointr(), plot_cut_boot(), plot_metric_boot(), plot_metric(), plot_precision_recall(), plot_roc(), plot_sensitivity_specificity(), plot_x()

Examples

```
set.seed(1)
oc <- cutpointr(suicide, dsi, suicide, boot_runs = 10)

plot_cutpointr(oc, cutpoint, F1_score)

## ROC curve
plot_cutpointr(oc, fpr, tpr, aspect_ratio = 1)

## Custom function
plot_cutpointr(oc, cutpoint, function(tp, tn, fp, fn, ...) tp / fp) +
  ggplot2::ggtitle("Custom metric") + ggplot2::ylab("value")
```

---

plot_cut_boot *Plot the bootstrapped distribution of optimal cutpoints from a cutpointr object*

Description

Given a cutpointr object this function plots the bootstrapped distribution of optimal cutpoints. cutpointr has to be run with boot_runs‘ > 0 to enable bootstrapping.

Usage

```
plot_cut_boot(x, ...)
```

Arguments

x	A cutpointr object.
...	Additional arguments (unused).

See Also

Other cutpointr plotting functions: plot.cutpointr(), plot_cutpointr(), plot_metric_boot(), plot_metric(), plot_precision_recall(), plot_roc(), plot_sensitivity_specificity(), plot_x()

Examples

```
set.seed(100)
opt_cut <- cutpointr(suicide, dsi, suicide, boot_runs = 10)
plot_cut_boot(opt_cut)
```

---

plot_metric *Plot a metric over all possible cutoffs from a cutpointr object*

Description

If maximize_metric is used as method function in cutpointr the computed metric values over all possible cutoffs can be plotted. Generally, this works for method functions that return a ROC-curve including the metric value for every cutpoint along with the optimal cutpoint.

Usage

```
plot_metric(x, conf_lvl = 0.95, add_unsmoothed = TRUE)
```

Arguments

x	A cutpointr object.
conf_lvl	The confidence level of the bootstrap confidence interval. Set to 0 to draw no bootstrap confidence interval.
add_unsmoothed	Add the line of unsmoothed metric values to the plot. Applicable for some smoothing methods, e.g. maximize_gam_metric.

See Also

Other cutpointr plotting functions: plot.cutpointr(), plot_cut_boot(), plot_cutpointr(), plot_metric_boot(), plot_precision_recall(), plot_roc(), plot_sensitivity_specificity(), plot_x()

Other cutpointr plotting functions: plot.cutpointr(), plot_cut_boot(), plot_cutpointr(), plot_metric_boot(), plot_precision_recall(), plot_roc(), plot_sensitivity_specificity(), plot_x()

Examples

```
opt_cut <- cutpointr(suicide, dsi, suicide)
plot_metric(opt_cut)
```

`plot_metric_boot` *Plot the bootstrapped metric distribution from a cutpointr object*

Description

Given a `cutpointr` object this function plots the bootstrapped metric distribution, i.e. the distribution of out-of-bag metric values. The metric depends on the function that was supplied to `metric` in the call to `cutpointr`. The `cutpointr` function has to be run with `boot_runs` > 0 to enable bootstrapping.

Usage

```
plot_metric_boot(x, ...)
```

Arguments

`x`	A cutpointr object.
`...`	Additional arguments (unused)

See Also

Other cutpointr plotting functions: `plot.cutpointr()`, `plot_cut_boot()`, `plot_cutpointr()`, `plot_metric()`, `plot_precision_recall()`, `plot_roc()`, `plot_sensitivity_specificity()`, `plot_x()`

Examples

```
set.seed(300)
opt_cut <- cutpointr(suicide, dsi, suicide, boot_runs = 10)
plot_metric_boot(opt_cut)
```

`plot_precision_recall` *Precision recall plot from a cutpointr object*

Description

Given a `cutpointr` object this function plots the precision recall curve(s) per subgroup, if given.

Usage

```
plot_precision_recall(x, display_cutpoint = TRUE, ...)
```

Arguments

x	A cutpointr object.
display_cutpoint	(logical) Whether or not to display the optimal cutpoint as a dot on the precision recall curve.
...	Additional arguments (unused).

See Also

Other cutpointr plotting functions: plot.cutpointr(), plot_cut_boot(), plot_cutpointr(), plot_metric_boot(), plot_metric(), plot_roc(), plot_sensitivity_specificity(), plot_x()

Examples

```
library(cutpointr)

## Optimal cutpoint for dsi
data(suicide)
opt_cut <- cutpointr(suicide, dsi, suicide)
plot_precision_recall(opt_cut)
```

---

plot_roc *Plot ROC curve from a cutpointr or roc_cutpointr object*

Description

Given a cutpointr object this function plots the ROC curve(s) per subgroup, if given. Also plots a ROC curve from the output of roc().

Usage

```
plot_roc(x, ...)

## S3 method for class 'cutpointr'
plot_roc(x, display_cutpoint = TRUE, type = "line", ...)

## S3 method for class 'roc_cutpointr'
plot_roc(x, type = "line", ...)
```

Arguments

x	A cutpointr or roc_cutpointr object.
...	Additional arguments (unused).
display_cutpoint	(logical) Whether or not to display the optimal cutpoint as a dot on the ROC curve for cutpointr objects.
type	"line" for line plot (default) or "step" for step plot.

See Also

Other cutpointr plotting functions: plot.cutpointr(), plot_cut_boot(), plot_cutpointr(), plot_metric_boot(), plot_metric(), plot_precision_recall(), plot_sensitivity_specificity(), plot_x()

Examples

```
opt_cut <- cutpointr(suicide, dsi, suicide)
plot_roc(opt_cut, display_cutpoint = FALSE)

opt_cut_2groups <- cutpointr(suicide, dsi, suicide, gender)
plot_roc(opt_cut_2groups, display_cutpoint = TRUE)

roc_curve <- roc(suicide, x = dsi, class = suicide, pos_class = "yes",
  neg_class = "no", direction = ">=")
plot(roc_curve)
auc(roc_curve)
```

---

plot_sensitivity_specificity

Sensitivity and specificity plot from a cutpointr object

Description

Given a cutpointr object this function plots the sensitivity and specificity curve(s) per subgroup, if the latter is given.

Usage

```
plot_sensitivity_specificity(x, display_cutpoint = TRUE, ...)
```

Arguments

x	A cutpointr object.
display_cutpoint	(logical) Whether or not to display the optimal cutpoint as a dot on the precision recall curve.
...	Additional arguments (unused).

See Also

Other cutpointr plotting functions: plot.cutpointr(), plot_cut_boot(), plot_cutpointr(), plot_metric_boot(), plot_metric(), plot_precision_recall(), plot_roc(), plot_x()

Examples

```
library(cutpointr)

## Optimal cutpoint for dsi
data(suicide)
opt_cut <- cutpointr(suicide, dsi, suicide)
plot_sensitivity_specificity(opt_cut)
```

`plot_x` *Plot the distribution of the independent variable per class from a cutpointr object*

Description

Given a `cutpointr` object this function plots the distribution(s) of the independent variable(s) and the respective cutpoints per class.

Usage

```
plot_x(x, display_cutpoint = TRUE, ...)
```

Arguments

`x`	A cutpointr object.
`display_cutpoint`	(logical) Whether or not to display the optimal cutpoint as a vertical line.
`...`	Additional arguments (unused).

See Also

Other cutpointr plotting functions: `plot.cutpointr()`, `plot_cut_boot()`, `plot_cutpointr()`, `plot_metric_boot()`, `plot_metric()`, `plot_precision_recall()`, `plot_roc()`, `plot_sensitivity_specificity()`

Examples

```
opt_cut <- cutpointr(suicide, dsi, suicide)
plot_x(opt_cut)

## With subgroup
opt_cut_2groups <- cutpointr(suicide, dsi, suicide, gender)
plot_x(opt_cut_2groups)
```

plr *Calculate the positive or negative likelihood ratio*

Description

Calculate the positive or negative likelihood ratio from true positives, false positives, true negatives and false negatives. The inputs must be vectors of equal length.

plr = tpr / fpr
nlr = fnr / tnr

Usage

```
plr(tp, fp, tn, fn, ...)

nlr(tp, fp, tn, fn, ...)
```

Arguments

tp	(numeric) number of true positives.
fp	(numeric) number of false positives.
tn	(numeric) number of true negatives.
fn	(numeric) number of false negatives.
...	for capturing additional arguments passed by method.

See Also

Other metric functions: F1_score(), Jaccard(), abs_d_ppv_npv(), abs_d_sens_spec(), accuracy(), cohens_kappa(), cutpoint(), false_omission_rate(), metric_constrain(), misclassification_cost(), npv(), odds_ratio(), p_chisquared(), ppv(), precision(), prod_ppv_npv(), prod_sens_spec(), recall(), risk_ratio(), roc01(), sensitivity(), specificity(), sum_ppv_npv(), sum_sens_spec(), total_utility(), tpr(), tp(), youden()

Examples

```
plr(10, 5, 20, 10)
plr(c(10, 8), c(5, 7), c(20, 12), c(10, 18))
```

ppv *Calculate the positive predictive value*

Description

Calculate the positive predictive value (PPV) from true positives, false positives, true negatives and false negatives. The inputs must be vectors of equal length.

ppv = tp / (tp + fp)

Usage

```
ppv(tp, fp, tn, fn, ...)
```

Arguments

`tp`	(numeric) number of true positives.
`fp`	(numeric) number of false positives.
`tn`	(numeric) number of true negatives.
`fn`	(numeric) number of false negatives.
`...`	for capturing additional arguments passed by method.

See Also

Other metric functions: `F1_score()`, `Jaccard()`, `abs_d_ppv_npv()`, `abs_d_sens_spec()`, `accuracy()`, `cohens_kappa()`, `cutpoint()`, `false_omission_rate()`, `metric_constrain()`, `misclassification_cost()`, `npv()`, `odds_ratio()`, `p_chisquared()`, `plr()`, `precision()`, `prod_ppv_npv()`, `prod_sens_spec()`, `recall()`, `risk_ratio()`, `roc01()`, `sensitivity()`, `specificity()`, `sum_ppv_npv()`, `sum_sens_spec()`, `total_utility()`, `tpr()`, `tp()`, `youden()`

Examples

```
ppv(10, 5, 20, 10)
ppv(c(10, 8), c(5, 7), c(20, 12), c(10, 18))
```

`precision` *Calculate precision*

Description

Calculate precision (equal to the positive predictive value) from true positives, false positives, true negatives and false negatives. The inputs must be vectors of equal length.

precision = tp / (tp + fp)

Usage

```
precision(tp, fp, tn, fn, ...)
```

Arguments

`tp`	(numeric) number of true positives.
`fp`	(numeric) number of false positives.
`tn`	(numeric) number of true negatives.
`fn`	(numeric) number of false negatives.
`...`	for capturing additional arguments passed by method.

See Also

Other metric functions: `F1_score()`, `Jaccard()`, `abs_d_ppv_npv()`, `abs_d_sens_spec()`, `accuracy()`, `cohens_kappa()`, `cutpoint()`, `false_omission_rate()`, `metric_constrain()`, `misclassification_cost()`, `npv()`, `odds_ratio()`, `p_chisquared()`, `plr()`, `ppv()`, `prod_ppv_npv()`, `prod_sens_spec()`, `recall()`, `risk_ratio()`, `roc01()`, `sensitivity()`, `specificity()`, `sum_ppv_npv()`, `sum_sens_spec()`, `total_utility()`, `tpr()`, `tp()`, `youden()`

Examples

```
precision(10, 5, 20, 10)
precision(c(10, 8), c(5, 7), c(20, 12), c(10, 18))
```

predict.cutpointr *Predict using a cutpointr object*

Description

Predictions are made on the `data.frame` in `newdata` using either the variable name or by applying the same transformation to the data as in `cutpointr`. The class of the output will be identical to the class of the predictor.

Usage

```
## S3 method for class 'cutpointr'
predict(object, newdata, cutpoint_nr = 1, ...)
```

Arguments

`object`	a cutpointr object.
`newdata`	a data.frame with a column that contains the predictor variable.
`cutpoint_nr`	if multiple optimal cutpoints were found this parameter defines which one should be used for predictions. Can be a vector if different cutpoint numbers are desired for different subgroups.
`...`	further arguments.

See Also

Other main cutpointr functions: `add_metric()`, `boot_ci()`, `boot_test()`, `cutpointr()`, `multi_cutpointr()`, `roc()`

Examples

```
oc <- cutpointr(suicide, dsi, suicide)
## Return in-sample predictions
predict(oc, newdata = data.frame(dsi = oc$data[[1]]$dsi))
```

print.cutpointr *Print cutpointr objects*

Description

Prints the `cutpointr` object with full width like a `tbl_df`.

Usage

```
## S3 method for class 'cutpointr'
print(x, width = 1000, n = 50, sigfig = 6, ...)
```

Arguments

x	a cutpointr object.
width	width of output.
n	number of rows to print.
sigfig	Number of significant digits to print. Temporarily overrides options("pillar.sigfig").
...	further arguments.

Source

Kirill Müller and Hadley Wickham (2017). tibble: Simple Data Frames. https://CRAN.R-project.org/package=tibble

---

print.multi_cutpointr *Print multi_cutpointr objects*

Description

Prints the multi_cutpointr object with infinite width like a tbl_df.

Usage

```
## S3 method for class 'multi_cutpointr'
print(x, n = Inf, ...)
```

Arguments

x	a multi_cutpointr object.
n	number of rows to print.
...	further arguments.

Source

Kirill Müller and Hadley Wickham (2017). tibble: Simple Data Frames. https://CRAN.R-project.org/package=tibble

prod_ppv_npv *Calculate the product of positive and negative predictive value*

Description

Calculate the product of positive predictive value (PPV) and negative predictive value (NPV) from true positives, false positives, true negatives and false negatives. The inputs must be vectors of equal length.

ppv = tp / (tp + fp)
npv = tn / (tn + fn)
prod_ppv_npv = ppv * npv

Usage

```
prod_ppv_npv(tp, fp, tn, fn, ...)
```

Arguments

tp	(numeric) number of true positives.
fp	(numeric) number of false positives.
tn	(numeric) number of true negatives.
fn	(numeric) number of false negatives.
...	for capturing additional arguments passed by method.

See Also

Other metric functions: F1_score(), Jaccard(), abs_d_ppv_npv(), abs_d_sens_spec(), accuracy(), cohens_kappa(), cutpoint(), false_omission_rate(), metric_constrain(), misclassification_cost(), npv(), odds_ratio(), p_chisquared(), plr(), ppv(), precision(), prod_sens_spec(), recall(), risk_ratio(), roc01(), sensitivity(), specificity(), sum_ppv_npv(), sum_sens_spec(), total_utility(), tpr(), tp(), youden()

Examples

```
prod_ppv_npv(10, 5, 20, 10)
prod_ppv_npv(c(10, 8), c(5, 7), c(20, 12), c(10, 18))
```

`prod_sens_spec` *Calculate the product of sensitivity and specificity*

Description

Calculate the product of sensitivity and specificity from true positives, false positives, true negatives and false negatives. The inputs must be vectors of equal length.

sensitivity = tp / (tp + fn)
specificity = tn / (tn + fp)
prod_sens_spec = sensitivity * specificity

Usage

```
prod_sens_spec(tp, fp, tn, fn, ...)
```

Arguments

`tp`	(numeric) number of true positives.
`fp`	(numeric) number of false positives.
`tn`	(numeric) number of true negatives.
`fn`	(numeric) number of false negatives.
`...`	for capturing additional arguments passed by method.

See Also

Other metric functions: `F1_score()`, `Jaccard()`, `abs_d_ppv_npv()`, `abs_d_sens_spec()`, `accuracy()`, `cohens_kappa()`, `cutpoint()`, `false_omission_rate()`, `metric_constrain()`, `misclassification_cost()`, `npv()`, `odds_ratio()`, `p_chisquared()`, `plr()`, `ppv()`, `precision()`, `prod_ppv_npv()`, `recall()`, `risk_ratio()`, `roc01()`, `sensitivity()`, `specificity()`, `sum_ppv_npv()`, `sum_sens_spec()`, `total_utility()`, `tpr()`, `tp()`, `youden()`

Examples

```
prod_sens_spec(10, 5, 20, 10)
prod_sens_spec(c(10, 8), c(5, 7), c(20, 12), c(10, 18))
```

prostate_nodal *Nodal involvement and acid phosphatase levels in 53 prostate cancer patients*

Description

Prostatic acid phosphatase (PAP) emerged as the first clinically useful tumor marker in the 1940s and 1950s. This data set contains the serum levels of acid phosphatase of 53 patients that were confirmed to have prostate cancer and whether the neighboring lymph nodes were involved.

Usage

```
prostate_nodal
```

Format

A data frame with 53 rows and 2 variables:

acid_phosphatase (numeric) Blood serum level of acid phosphatase

nodal_involvement (logical) Whether neighboring lymph nodes were involved

Source

Le CT (2006). A solution for the most basic optimization problem associated with an ROC curve. Statistical methods in medical research 15: 571–584

p_chisquared *Calculate the p-value of a chi-squared test*

Description

Calculate the p-value of a chi-squared test from true positives, false positives, true negatives and false negatives. The inputs must be vectors of equal length.

Usage

```
p_chisquared(tp, fp, tn, fn, ...)
```

Arguments

tp	(numeric) number of true positives.
fp	(numeric) number of false positives.
tn	(numeric) number of true negatives.
fn	(numeric) number of false negatives.
...	for capturing additional arguments passed by method.

See Also

Other metric functions: F1_score(), Jaccard(), abs_d_ppv_npv(), abs_d_sens_spec(), accuracy(), cohens_kappa(), cutpoint(), false_omission_rate(), metric_constrain(), misclassification_cost(), npv(), odds_ratio(), plr(), ppv(), precision(), prod_ppv_npv(), prod_sens_spec(), recall(), risk_ratio(), roc01(), sensitivity(), specificity(), sum_ppv_npv(), sum_sens_spec(), total_utility(), tpr(), tp(), youden()

Examples

```
p_chisquared(10, 5, 20, 10)
p_chisquared(c(10, 8), c(5, 7), c(20, 12), c(10, 18))
```

---

recall *Calculate recall*

Description

Calculate recall (equal to sensitivity) from true positives, false positives, true negatives and false negatives. The inputs must be vectors of equal length.

recall = tp / (tp + fn)

Usage

```
recall(tp, fp, tn, fn, ...)
```

Arguments

tp	(numeric) number of true positives.
fp	(numeric) number of false positives.
tn	(numeric) number of true negatives.
fn	(numeric) number of false negatives.
...	for capturing additional arguments passed by method.

See Also

Other metric functions: F1_score(), Jaccard(), abs_d_ppv_npv(), abs_d_sens_spec(), accuracy(), cohens_kappa(), cutpoint(), false_omission_rate(), metric_constrain(), misclassification_cost(), npv(), odds_ratio(), p_chisquared(), plr(), ppv(), precision(), prod_ppv_npv(), prod_sens_spec(), risk_ratio(), roc01(), sensitivity(), specificity(), sum_ppv_npv(), sum_sens_spec(), total_utility(), tpr(), tp(), youden()

Examples

```
recall(10, 5, 20, 10)
recall(c(10, 8), c(5, 7), c(20, 12), c(10, 18))
```

`risk_ratio` *Calculate the risk ratio (relative risk)*

Description

Calculate the risk ratio (or relative risk) from true positives, false positives, true negatives and false negatives. The inputs must be vectors of equal length.

risk_ratio = (tp / (tp + fn)) / (fp / (fp + tn))

Usage

```
risk_ratio(tp, fp, tn, fn, ...)
```

Arguments

`tp`	(numeric) number of true positives.
`fp`	(numeric) number of false positives.
`tn`	(numeric) number of true negatives.
`fn`	(numeric) number of false negatives.
`...`	for capturing additional arguments passed by method.

See Also

Other metric functions: `F1_score()`, `Jaccard()`, `abs_d_ppv_npv()`, `abs_d_sens_spec()`, `accuracy()`, `cohens_kappa()`, `cutpoint()`, `false_omission_rate()`, `metric_constrain()`, `misclassification_cost()`, `npv()`, `odds_ratio()`, `p_chisquared()`, `plr()`, `ppv()`, `precision()`, `prod_ppv_npv()`, `prod_sens_spec()`, `recall()`, `roc01()`, `sensitivity()`, `specificity()`, `sum_ppv_npv()`, `sum_sens_spec()`, `total_utility()`, `tpr()`, `tp()`, `youden()`

Examples

```
risk_ratio(10, 5, 20, 10)
risk_ratio(c(10, 8), c(5, 7), c(20, 12), c(10, 18))
```

`roc` *Calculate a ROC curve*

Description

Given a `data.frame` with a numeric predictor variable and a binary outcome variable this function returns a `data.frame` that includes all elements of the confusion matrix (true positives, false positives, true negatives, and false negatives) for every unique value of the predictor variable. Additionally, the true positive rate (tpr), false positive rate (fpr), true negative rate (tnr) and false negative rate (fnr) are returned.

Usage

```
roc(data, x, class, pos_class, neg_class, direction = ">=", silent = FALSE)
```

Arguments

`data`	A data.frame or matrix. Will be converted to a data.frame.
`x`	The name of the numeric predictor variable.
`class`	The name of the binary outcome variable.
`pos_class`	The value of 'class' that represents the positive cases.
`neg_class`	The value of 'class' that represents the negative cases.
`direction`	(character) One of ">=" or "<=". Specifies if the positive class is associated with higher values of x (default).
`silent`	If FALSE and the ROC curve contains no positives or negatives, a warning is generated.

Details

To enable classifying all observations as belonging to only one class the predictor values will be augmented by Inf or -Inf. The returned object can be plotted with plot_roc.

This function uses tidyeval to support unquoted arguments. For programming with `roc` the operator `!!` can be used to unquote an argument, see the examples.

Value

A data frame with the columns x.sorted, tp, fp, tn, fn, tpr, tnr, fpr, and fnr.

Source

Forked from the **ROCR** package

See Also

Other main cutpointr functions: `add_metric()`, `boot_ci()`, `boot_test()`, `cutpointr()`, `multi_cutpointr()`, `predict.cutpointr()`

Examples

```
roc_curve <- roc(data = suicide, x = dsi, class = suicide,
  pos_class = "yes", neg_class = "no", direction = ">=")
roc_curve
plot_roc(roc_curve)
auc(roc_curve)

## Unquoting an argument
myvar <- "dsi"
roc(suicide, x = !!myvar, suicide, pos_class = "yes", neg_class = "no")
```

`roc01` *Calculate the distance between points on the ROC curve and (0,1)*

Description

Calculate the distance on the ROC space between points on the ROC curve and the point of perfect discrimination from true positives, false positives, true negatives and false negatives. The inputs must be vectors of equal length. To be used with `method = minimize_metric`.

sensitivity = tp / (tp + fn)
specificity = tn / (tn + fp)
roc01 = sqrt((1 - sensitivity)^2 + (1 - specificity)^2)

Usage

```
roc01(tp, fp, tn, fn, ...)
```

Arguments

`tp`	(numeric) number of true positives.
`fp`	(numeric) number of false positives.
`tn`	(numeric) number of true negatives.
`fn`	(numeric) number of false negatives.
`...`	for capturing additional arguments passed by method.

See Also

Other metric functions: `F1_score()`, `Jaccard()`, `abs_d_ppv_npv()`, `abs_d_sens_spec()`, `accuracy()`, `cohens_kappa()`, `cutpoint()`, `false_omission_rate()`, `metric_constrain()`, `misclassification_cost()`, `npv()`, `odds_ratio()`, `p_chisquared()`, `plr()`, `ppv()`, `precision()`, `prod_ppv_npv()`, `prod_sens_spec()`, `recall()`, `risk_ratio()`, `sensitivity()`, `specificity()`, `sum_ppv_npv()`, `sum_sens_spec()`, `total_utility()`, `tpr()`, `tp()`, `youden()`

Examples

```
roc01(10, 5, 20, 10)
roc01(c(10, 8), c(5, 7), c(20, 12), c(10, 18))
oc <- cutpointr(suicide, dsi, suicide,
  method = minimize_metric, metric = roc01)
plot_roc(oc)
```

`sensitivity` *Calculate sensitivity*

Description

Calculate sensitivity from true positives, false positives, true negatives and false negatives. The inputs must be vectors of equal length.

sensitivity = tp / (tp + fn)

Usage

```
sensitivity(tp, fn, ...)
```

Arguments

`tp`	(numeric) number of true positives.
`fn`	(numeric) number of false negatives.
`...`	for capturing additional arguments passed by method.

See Also

Other metric functions: `F1_score()`, `Jaccard()`, `abs_d_ppv_npv()`, `abs_d_sens_spec()`, `accuracy()`, `cohens_kappa()`, `cutpoint()`, `false_omission_rate()`, `metric_constrain()`, `misclassification_cost()`, `npv()`, `odds_ratio()`, `p_chisquared()`, `plr()`, `ppv()`, `precision()`, `prod_ppv_npv()`, `prod_sens_spec()`, `recall()`, `risk_ratio()`, `roc01()`, `specificity()`, `sum_ppv_npv()`, `sum_sens_spec()`, `total_utility()`, `tpr()`, `tp()`, `youden()`

Examples

```
sensitivity(10, 5, 20, 10)
sensitivity(c(10, 8), c(5, 7), c(20, 12), c(10, 18))
```

specificity *Calculate specificity*

Description

Calculate specificity from true positives, false positives, true negatives and false negatives. The inputs must be vectors of equal length.

specificity = tn / (tn + fp)

Usage

```
specificity(fp, tn, ...)
```

Arguments

fp	(numeric) number of false positives.
tn	(numeric) number of true negatives.
...	for capturing additional arguments passed by method.

See Also

Other metric functions: F1_score(), Jaccard(), abs_d_ppv_npv(), abs_d_sens_spec(), accuracy(), cohens_kappa(), cutpoint(), false_omission_rate(), metric_constrain(), misclassification_cost(), npv(), odds_ratio(), p_chisquared(), plr(), ppv(), precision(), prod_ppv_npv(), prod_sens_spec(), recall(), risk_ratio(), roc01(), sensitivity(), sum_ppv_npv(), sum_sens_spec(), total_utility(), tpr(), tp(), youden()

Examples

```
specificity(10, 5, 20, 10)
specificity(c(10, 8), c(5, 7), c(20, 12), c(10, 18))
```

suicide *Suicide attempts and DSI sum scores of 532 subjects*

Description

Various personality and clinical psychological characteristics were assessed as part of an online-study preventing suicide. To identify persons at risk for attempting suicide, various demographic and clinical characteristics were assessed. Depressive Symptom Inventory - Suicidality Subscale (DSA-SS) sum scores and past suicide attempts from 532 subjects are included as a demonstration set to calculate optimal cutpoints. Two additional demographic variables (age, gender) are also included to test for group differences.

Usage

```
suicide
```

Format

A data frame with 532 rows and 4 variables:

age (numeric) Age of participants in years

gender (factor) Gender

dsi (numeric) Sum-score (0 = low suicidality, 12 = high suicidality)

suicide (factor) Past suicide attempt (no = no attempt, yes = at least one attempt)

Source

von Glischinski, M., Teisman, T., Prinz, S., Gebauer, J., and Hirschfeld, G. (2017). Depressive Symptom Inventory- Suicidality Subscale: Optimal cut points for clinical and non-clinical samples. Clinical Psychology & Psychotherapy

---

sum_ppv_npv *Calculate the sum of positive and negative predictive value*

Description

Calculate the sum of positive predictive value (PPV) and negative predictive value (NPV) from true positives, false positives, true negatives and false negatives. The inputs must be vectors of equal length.

ppv = tp / (tp + fp)
npv = tn / (tn + fn)
sum_ppv_npv = ppv + npv

Usage

```
sum_ppv_npv(tp, fp, tn, fn, ...)
```

Arguments

`tp`	(numeric) number of true positives.
`fp`	(numeric) number of false positives.
`tn`	(numeric) number of true negatives.
`fn`	(numeric) number of false negatives.
`...`	for capturing additional arguments passed by method.

See Also

Other metric functions: F1_score(), Jaccard(), abs_d_ppv_npv(), abs_d_sens_spec(), accuracy(), cohens_kappa(), cutpoint(), false_omission_rate(), metric_constrain(), misclassification_cost(), npv(), odds_ratio(), p_chisquared(), plr(), ppv(), precision(), prod_ppv_npv(), prod_sens_spec(), recall(), risk_ratio(), roc01(), sensitivity(), specificity(), sum_sens_spec(), total_utility(), tpr(), tp(), youden()

Examples

```
sum_ppv_npv(10, 5, 20, 10)
sum_ppv_npv(c(10, 8), c(5, 7), c(20, 12), c(10, 18))
```

---

sum_sens_spec *Calculate the sum of sensitivity and specificity*

Description

Calculate the sum of sensitivity and specificity from true positives, false positives, true negatives and false negatives. The inputs must be vectors of equal length.

sensitivity = tp / (tp + fn)
specificity = tn / (tn + fp)
sum_sens_spec = sensitivity + specificity

Usage

```
sum_sens_spec(tp, fp, tn, fn, ...)
```

Arguments

tp	(numeric) number of true positives.
fp	(numeric) number of false positives.
tn	(numeric) number of true negatives.
fn	(numeric) number of false negatives.
...	for capturing additional arguments passed by method.

See Also

Other metric functions: F1_score(), Jaccard(), abs_d_ppv_npv(), abs_d_sens_spec(), accuracy(), cohens_kappa(), cutpoint(), false_omission_rate(), metric_constrain(), misclassification_cost(), npv(), odds_ratio(), p_chisquared(), plr(), ppv(), precision(), prod_ppv_npv(), prod_sens_spec(), recall(), risk_ratio(), roc01(), sensitivity(), specificity(), sum_ppv_npv(), total_utility(), tpr(), tp(), youden()

Examples

```
sum_sens_spec(10, 5, 20, 10)
sum_sens_spec(c(10, 8), c(5, 7), c(20, 12), c(10, 18))
```

`total_utility` *Calculate the total utility*

Description

Calculate the total utility from true positives, false positives, true negatives and false negatives.

total_utility = utility_tp * tp + utility_tn * tn - cost_fp * fp - cost_fn * fn

The inputs must be vectors of equal length.

Usage

```
total_utility(
  tp,
  fp,
  tn,
  fn,
  utility_tp = 1,
  utility_tn = 1,
  cost_fp = 1,
  cost_fn = 1,
  ...
)
```

Arguments

`tp`	(numeric) number of true positives.
`fp`	(numeric) number of false positives.
`tn`	(numeric) number of true negatives.
`fn`	(numeric) number of false negatives.
`utility_tp`	(numeric) the utility of a true positive
`utility_tn`	(numeric) the utility of a true negative
`cost_fp`	(numeric) the cost of a false positive
`cost_fn`	(numeric) the cost of a false negative
`...`	for capturing additional arguments passed by method.

See Also

Other metric functions: F1_score(), Jaccard(), abs_d_ppv_npv(), abs_d_sens_spec(), accuracy(), cohens_kappa(), cutpoint(), false_omission_rate(), metric_constrain(), misclassification_cost(), npv(), odds_ratio(), p_chisquared(), plr(), ppv(), precision(), prod_ppv_npv(), prod_sens_spec(), recall(), risk_ratio(), roc01(), sensitivity(), specificity(), sum_ppv_npv(), sum_sens_spec(), tpr(), tp(), youden()

Examples

```
total_utility(10, 5, 20, 10, utility_tp = 3, utility_tn = 3, cost_fp = 1, cost_fn = 5)
total_utility(c(10, 8), c(5, 7), c(20, 12), c(10, 18),
              utility_tp = 3, utility_tn = 3, cost_fp = 1, cost_fn = 5)
```

---

tp *Extract number true / false positives / negatives*

Description

Extract the number of true positives (tp), false positives (fp), true negatives (tn), or false negatives (fn). The inputs must be vectors of equal length. Mainly useful for plot_cutpointr.

Usage

```
tp(tp, ...)

tn(tn, ...)

fp(fp, ...)

fn(fn, ...)
```

Arguments

tp	(numeric) number of true positives.
...	for capturing additional arguments passed by method.
tn	(numeric) number of true negatives.
fp	(numeric) number of false positives.
fn	(numeric) number of false negatives.

See Also

Other metric functions: F1_score(), Jaccard(), abs_d_ppv_npv(), abs_d_sens_spec(), accuracy(), cohens_kappa(), cutpoint(), false_omission_rate(), metric_constrain(), misclassification_cost(), npv(), odds_ratio(), p_chisquared(), plr(), ppv(), precision(), prod_ppv_npv(), prod_sens_spec(), recall(), risk_ratio(), roc01(), sensitivity(), specificity(), sum_ppv_npv(), sum_sens_spec(), total_utility(), tpr(), youden()

Examples

```
tp(10, 5, 20, 10)
tp(c(10, 8), c(5, 7), c(20, 12), c(10, 18))
fp(10, 5, 20, 10)
tn(10, 5, 20, 10)
fn(10, 5, 20, 10)
```

`tpr` *Calculate true / false positive / negative rate*

Description

Calculate the true positive rate (tpr, equal to sensitivity and recall), the false positive rate (fpr, equal to fall-out), the true negative rate (tnr, equal to specificity), or the false negative rate (fnr) from true positives, false positives, true negatives and false negatives. The inputs must be vectors of equal length.

tpr = tp / (tp + fn)
fpr = fp / (fp + tn)
tnr = tn / (tn + fp)
fnr = fn / (fn + tp)

Usage

```
tpr(tp, fn, ...)

fpr(fp, tn, ...)

tnr(fp, tn, ...)

fnr(tp, fn, ...)
```

Arguments

`tp`	(numeric) number of true positives.
`fn`	(numeric) number of false negatives.
`...`	for capturing additional arguments passed by method.
`fp`	(numeric) number of false positives.
`tn`	(numeric) number of true negatives.

See Also

Other metric functions: `F1_score()`, `Jaccard()`, `abs_d_ppv_npv()`, `abs_d_sens_spec()`, `accuracy()`, `cohens_kappa()`, `cutpoint()`, `false_omission_rate()`, `metric_constrain()`, `misclassification_cost()`, `npv()`, `odds_ratio()`, `p_chisquared()`, `plr()`, `ppv()`, `precision()`, `prod_ppv_npv()`, `prod_sens_spec()`, `recall()`, `risk_ratio()`, `roc01()`, `sensitivity()`, `specificity()`, `sum_ppv_npv()`, `sum_sens_spec()`, `total_utility()`, `tp()`, `youden()`

Examples

```
tpr(10, 5, 20, 10)
tpr(c(10, 8), c(5, 7), c(20, 12), c(10, 18))
```

`user_span_cutpointr` *Calculate bandwidth for LOESS smoothing of metric functions by rule of thumb*

Description

This function implements a rule of thumb for selecting the bandwidth when smoothing a function of metric values per cutpoint value, particularly in `maximize_loess_metric` and `minimize_loess_metric`.

Usage

```
user_span_cutpointr(data, x)
```

Arguments

`data`	A data frame
`x`	The predictor variable

Details

The function used for calculating the bandwidth is 0.1 * xsd / sqrt(xn), where xsd is the standard deviation of the unique values of the predictor variable (i.e. all cutpoints) and xn is the number of unique predictor values.

`youden` *Calculate the Youden-Index*

Description

Calculate the Youden-Index (J-Index) from true positives, false positives, true negatives and false negatives. The inputs must be vectors of equal length.

sensitivity = tp / (tp + fn)
specificity = tn / (tn + fp)
youden_index = sensitivity + specificity - 1

Usage

```
youden(tp, fp, tn, fn, ...)
```

Arguments

tp	(numeric) number of true positives.
fp	(numeric) number of false positives.
tn	(numeric) number of true negatives.
fn	(numeric) number of false negatives.
...	for capturing additional arguments passed by method.

See Also

Other metric functions: F1_score(), Jaccard(), abs_d_ppv_npv(), abs_d_sens_spec(), accuracy(), cohens_kappa(), cutpoint(), false_omission_rate(), metric_constrain(), misclassification_cost(), npv(), odds_ratio(), p_chisquared(), plr(), ppv(), precision(), prod_ppv_npv(), prod_sens_spec(), recall(), risk_ratio(), roc01(), sensitivity(), specificity(), sum_ppv_npv(), sum_sens_spec(), total_utility(), tpr(), tp()

Examples

```
youden(10, 5, 20, 10)
youden(c(10, 8), c(5, 7), c(20, 12), c(10, 18))
```

Index