

Package ‘pROC’

August 31, 2012

Type Package

Title display and analyze ROC curves

Version 1.5.3

Date 2012-08-31

Encoding UTF-8

Depends plyr, R (>= 2.10), utils

Suggests tcltk, MASS, logcondens

Author Xavier Robin, Natacha Turck, Alexandre Hainard, Natalia Tiberti, Frédérique Lisacek, Jean-Charles Sanchez and Markus Müller

Maintainer Xavier Robin <Xavier.Robin@unige.ch>

Description Tools for visualizing, smoothing and comparing receiver operating characteristic (ROC curves). (Partial) area under the curve (AUC) can be compared with statistical tests based on U-statistics or bootstrap. Confidence intervals can be computed for (p)AUC or ROC curves.

Acknowledgements The authors would like to thank E. S. Venkatraman and Colin B. Begg for their support in the implementation of their test.

License GPL (>= 3)

URL <http://expasy.org/tools/pROC/>

LazyLoad yes

R topics documented:

pROC-package	2
are.paired	6
aSAH	8
auc	9
ci	12

ci.auc	14
ci.se	17
ci.sp	20
ci.thresholds	23
coords	26
cov.roc	30
has.partial.auc	35
lines.roc	36
multiclass.roc	37
plot.ci	39
plot.roc	41
print	46
roc	48
roc.test	52
smooth.roc	59
var.roc	65
Index	69

pROC-package	<i>pROC</i>
--------------	-------------

Description

Tools for visualizing, smoothing and comparing receiver operating characteristic (ROC curves). (Partial) area under the curve (AUC) can be compared with statistical tests based on U-statistics or bootstrap. Confidence intervals can be computed for (p)AUC or ROC curves.

In S+, the package comes with a graphical user interface.

Details

The basic unit of the pROC package is the [roc](#) function. It will build a ROC curve, smooth it if requested (if `smooth=TRUE`), compute the AUC (if `auc=TRUE`), the confidence interval (CI) if requested (if `ci=TRUE`) and plot the curve if requested (if `plot=TRUE`).

The [roc](#) function will call [smooth.roc](#), [auc](#), [ci](#) and [plot](#) as necessary. See these individual functions for the arguments that can be passed to them through [roc](#). These function can be called separately.

Two paired (that is [roc](#) objects with the same response) or unpaired (with different response) ROC curves can be compared with the [roc.test](#) function.

Citation

If you use pROC in published research, please cite the following paper:

Xavier Robin, Natacha Turck, Alexandre Hainard, Natalia Tiberti, Frédérique Lisacek, Jean-Charles Sanchez and Markus Müller (2011). “pROC: an open-source package for R and S+ to analyze and compare ROC curves”. *BMC Bioinformatics*, **12**, p. 77. DOI: [10.1186/1471-2105-12-77](https://doi.org/10.1186/1471-2105-12-77)

Type `citation("pROC")` for a BibTeX entry.

The authors would be glad to hear how pROC is employed. You are kindly encouraged to notify Xavier Robin <Xavier.Robin@unige.ch> about any work you publish.

Abbreviations

The following abbreviations are employed extensively in this package:

- ROC: receiver operating characteristic
- AUC: area under the ROC curve
- pAUC: partial area under the ROC curve
- CI: confidence interval
- SP: specificity
- SE: sensitivity

Functions

<code>roc</code>	Build a ROC curve
<code>are.paired</code>	Determine if two ROC curves are paired
<code>auc</code>	Compute the area under the ROC curve
<code>ci</code>	Compute confidence intervals of a ROC curve
<code>ci.auc</code>	Compute the CI of the AUC
<code>ci.se</code>	Compute the CI of sensitivities at given specificities
<code>ci.sp</code>	Compute the CI of specificities at given sensitivities
<code>ci.thresholds</code>	Compute the CI of specificity and sensitivity of thresholds
<code>coords</code>	Coordinates of a ROC curve
<code>cov</code>	Covariance between two AUCs
<code>has.partial.auc</code>	Determine if the ROC curve have a partial AUC
<code>lines.roc</code>	Add a ROC line to a ROC plot
<code>plot.ci</code>	Plot CIs
<code>plot</code>	Plot a ROC curve
<code>print</code>	Print a ROC curve object
<code>roc.test</code>	Compare the AUC of two ROC curves
<code>smooth</code>	Smooth a ROC curve
<code>var</code>	Variance of the AUC

Dataset

This package comes with a dataset of 141 patients with aneurysmal subarachnoid hemorrhage: [aSAH](#).

Installing and using

To install this package, make sure you are connected to the internet and issue the following command in the R prompt:

```
install.packages("pROC")
```

To load the package in R:

```
library(pROC)
```

Bootstrap

All the bootstrap operations for [significance testing](#) and [confidence interval](#) computation are performed with non-parametric stratified or non-stratified resampling (according to the `stratified` argument) and with the percentile method, as described in Carpenter and Bithell (2000) sections 2.1 and 3.3.

Stratification of bootstrap can be controlled with `boot.stratified`. In stratified bootstrap (the default), each replicate contains the same number of cases and controls than the original sample. Stratification is especially useful if one group has only little observations, or if groups are not balanced.

The number of bootstrap replicates is controlled by `boot.n`. Higher numbers will give a more precise estimate of the significance tests and confidence intervals but take more time to compute. 2000 is recommended by Carpenter and Bithell for confidence intervals. In our experience this is sufficient for a good estimation of the first significant digit only, so we recommend the use of 10000 bootstrap replicates to obtain a good estimate of the second significant digit whenever possible.

Progress bars: A progressbar shows the progress of bootstrap operations. It is handled by the **plyr** package, and is created by the `progress_*` family of functions. Sensible defaults are guessed during the package loading:

- In non-[interactive](#) mode, no progressbar is displayed.
- In embedded GNU Emacs “ESS”, a [txtProgressBar](#)
- In Windows, a `winProgressBar` bar.
- In other systems with a graphical display, a [tkProgressBar](#).
- In systems without a graphical display, a [txtProgressBar](#).

The default can be changed with the option “pROCProgress”. The option must be a list with a `name` item setting the type of progress bar (“none”, “win”, “tk” or “text”). Optional items of the list are “width”, “char” and “style”, corresponding to the arguments to the underlying progressbar functions. For example, to force a text progress bar:

```
options(pROCProgress = list(name = "text", width = NA, char = "=", style = 3))
```

To inhibit the progress bars completely:

```
options(pROCProgress = list(name = "none"))
```

Author(s)

Xavier Robin, Natacha Turck, Jean-Charles Sanchez and Markus Müller

Maintainer: Xavier Robin <Xavier.Robin@unige.ch>

References

Tom Fawcett (2006) “An introduction to ROC analysis”. *Pattern Recognition Letters* **27**, 861–874. DOI: [10.1016/j.patrec.2005.10.010](https://doi.org/10.1016/j.patrec.2005.10.010).

Xavier Robin, Natacha Turck, Alexandre Hainard, *et al.* (2011) “pROC: an open-source package for R and S+ to analyze and compare ROC curves”. *BMC Bioinformatics*, **7**, 77. DOI: [10.1186/1471-2105-12-77](https://doi.org/10.1186/1471-2105-12-77).

See Also

CRAN packages **ROCR**, **verification** or Bioconductor’s **roc** for ROC curves.

CRAN packages **plyr**, **MASS** and **logcondens** employed in this package.

Examples

```
data(aSAH)

# Build a ROC object and compute the AUC
roc(aSAH$outcome, aSAH$s100b)
roc(outcome ~ s100b, aSAH)

# Smooth ROC curve
roc(outcome ~ s100b, aSAH, smooth=TRUE)

# more options, CI and plotting
roc1 <- roc(aSAH$outcome,
            aSAH$s100b, percent=TRUE,
            # arguments for auc
            partial.auc=c(100, 90), partial.auc.correct=TRUE,
            partial.auc.focus="sens",
            # arguments for ci
            ci=TRUE, boot.n=100, ci.alpha=0.9, stratified=FALSE,
            # arguments for plot
            plot=TRUE, auc.polygon=TRUE, max.auc.polygon=TRUE, grid=TRUE,
            print.auc=TRUE, show.thres=TRUE)

# Add to an existing plot. Beware of 'percent' specification!
roc2 <- roc(aSAH$outcome, aSAH$wfns,
            plot=TRUE, add=TRUE, percent=roc1$percent)

## Coordinates of the curve ##
coords(roc1, "best", ret=c("threshold", "specificity", "1-npv"))
coords(roc2, "local maximas", ret=c("threshold", "sens", "spec", "ppv", "npv"))

## Confidence intervals ##

# CI of the AUC
ci(roc2)

## Not run:
# CI of the curve
sens.ci <- ci.se(roc1, specificities=seq(0, 100, 5))
```

```

plot(sens.ci, type="shape", col="lightblue")
plot(sens.ci, type="bars")

## End(Not run)

# need to re-add roc2 over the shape
plot(roc2, add=TRUE)

## Not run:
# CI of thresholds
plot(ci.thresholds(roc2))

## End(Not run)

## Comparisons ##

# Test on the whole AUC
roc.test(roc1, roc2, reuse.auc=FALSE)

## Not run:
# Test on a portion of the whole AUC
roc.test(roc1, roc2, reuse.auc=FALSE, partial.auc=c(100, 90),
         partial.auc.focus="se", partial.auc.correct=TRUE)

# With modified bootstrap parameters
roc.test(roc1, roc2, reuse.auc=FALSE, partial.auc=c(100, 90),
         partial.auc.correct=TRUE, boot.n=1000, boot.stratified=FALSE)

## End(Not run)

```

are.paired

Are two ROC curves paired?

Description

This function determines if two ROC curves can be paired.

Usage

```

are.paired(...)
## S3 method for class 'auc'
are.paired(roc1, roc2, ...)
## S3 method for class 'smooth.roc'
are.paired(roc1, roc2, ...)
## S3 method for class 'roc'
are.paired(roc1, roc2, return.paired.rocs=FALSE,
           reuse.auc = TRUE, reuse.ci = FALSE, reuse.smooth=TRUE, ...)

```

Arguments

`roc1`, `roc2` the two ROC curves to compare. Either “`roc`”, “`auc`” or “`smooth.roc`” objects (types can be mixed).

`return.paired.rocs` if TRUE and the ROC curves can be paired, the two paired ROC curves with NAs removed will be returned.

`reuse.auc`, `reuse.ci`, `reuse.smooth` if `return.paired.rocs=TRUE`, determines if `auc`, `ci` and `smooth.roc` should be re-computed (with the same parameters than the original ROC curves)

... additionnal arguments for `are.paired.roc`. Ignored in `are.paired.roc`

Details

Two ROC curves are paired if they are built on two variables observed on the same sample.

In practice, the paired status is granted if the response vector of both ROC curves are [identical](#). If the responses are different, this can be due to missing values differing between the curves. In this case, the function will strip all NAs in both curves and check for identity again.

It can raise false positives if the responses are identical but correspond to different patients.

Value

TRUE if `roc1` and `roc2` are paired, FALSE otherwise.

In addition, if TRUE and `return.paired.rocs=TRUE`, the following attributes are defined:

`roc1`, `roc2` the two ROC curve with all NAs values removed in both curves.

See Also

[roc](#), [roc.test](#)

Examples

```
data(aSAH)
aSAH.copy <- aSAH

# artificially insert NAs for demonstration purposes
aSAH.copy$outcome[42] <- NA
aSAH.copy$s100b[24] <- NA
aSAH.copy$ndka[1:10] <- NA

# Call roc() on the whole data
roc1 <- roc(aSAH.copy$outcome, aSAH.copy$s100b)
roc2 <- roc(aSAH.copy$outcome, aSAH.copy$ndka)
# are.paired can still find that the curves were paired
are.paired(roc1, roc2) # TRUE

# Removing the NAs manually before passing to roc() un-pairs the ROC curves
nas <- is.na(aSAH.copy$outcome) | is.na(aSAH.copy$ndka)
roc2b <- roc(aSAH.copy$outcome[!nas], aSAH.copy$ndka[!nas])
```

```

are.paired(roc1, roc2b) # FALSE

# Getting the two paired ROC curves with additional smoothing and ci options
roc2$ci <- ci(roc2)
paired <- are.paired(smooth(roc1), roc2, return.paired.rocs=TRUE, reuse.ci=TRUE)
paired.roc1 <- attr(paired, "roc1")
paired.roc2 <- attr(paired, "roc2")

```

aSAH

Subarachnoid hemorrhage data

Description

This dataset summarizes several clinical and one laboratory variable of 113 patients with an aneurysmal subarachnoid hemorrhage.

Usage

```
aSAH
```

Format

A data.frame containing 113 observations of 7 variables.

Source

Natacha Turck, Laszlo Vutskits, Paola Sanchez-Pena, Xavier Robin, Alexandre Hainard, Marianne Gex-Fabry, Catherine Fouda, Hadiji Bassem, Markus Mueller, Frédérique Lisacek, Louis Puybasset and Jean-Charles Sanchez (2010) “A multiparameter panel method for outcome prediction following aneurysmal subarachnoid hemorrhage”. *Intensive Care Medicine* **36**(1), 107–115. DOI: [10.1007/s00134-009-1641-y](https://doi.org/10.1007/s00134-009-1641-y).

See Also

Other examples can be found in all the documentation pages of this package: [roc](#), [auc](#), [ci](#), [ci.auc](#), [ci.se](#), [ci.sp](#), [ci.thresholds](#), [coords](#), [plot.ci](#), [plot.roc](#), [print.roc](#), [roc.test](#) and [smooth](#).

An example analysis with pROC is shown in:

Xavier Robin, Natacha Turck, Alexandre Hainard, *et al.* (2011) “pROC: an open-source package for R and S+ to analyze and compare ROC curves”. *BMC Bioinformatics*, **7**, 77. DOI: [10.1186/1471-2105-12-77](https://doi.org/10.1186/1471-2105-12-77)

Examples

```
# load the dataset
data(aSAH)

# Gender, outcome and set
with(aSAH, table(gender, outcome))

# Age
with(aSAH, by(age, outcome, mean))
with(aSAH, by(age, outcome,
  function(x) sprintf("mean: %.1f (+/- %.1f), median: %.1f (%i-%i)",
    mean(x), sd(x), median(x), min(x), max(x))))

# WFNS score
with(aSAH, table(wfns=ifelse(wfns<=2, "1-2", "3-4-5"), outcome))
```

auc

Compute the area under the ROC curve

Description

This function computes the numeric value of area under the ROC curve (AUC) with the trapezoidal rule. Two syntaxes are possible: one object of class “[roc](#)”, or either two vectors (response, predictor) or a formula (response~predictor) as in the [roc](#) function. By default, the total AUC is computed, but a portion of the ROC curve can be specified with `partial.auc`.

Usage

```
auc(...)
## S3 method for class 'roc'
auc(roc, partial.auc=FALSE, partial.auc.focus=c("specificity",
"sensitivity"), partial.auc.correct=FALSE, ...)
## S3 method for class 'smooth.roc'
auc(smooth.roc, ...)
## S3 method for class 'multiclass.roc'
auc(multiclass.roc, ...)
## S3 method for class 'formula'
auc(formula, data, ...)
## Default S3 method:
auc(response, predictor, ...)
```

Arguments

roc, smooth.roc, multiclass.roc

a “roc” object from the [roc](#) function, a “smooth.roc” object from the [smooth.roc](#) function, or a “multiclass.roc” from the [multiclass.roc](#) function.

<code>response, predictor</code>	arguments for the <code>roc</code> function.
<code>formula, data</code>	a formula (and possibly a data object) of type <code>response~predictor</code> for the <code>roc</code> function.
<code>partial.auc</code>	either FALSE (default: consider total area) or a numeric vector of length 2: boundaries of the AUC to consider in [0,1] (or [0,100] if percent is TRUE).
<code>partial.auc.focus</code>	if <code>partial.auc</code> is not FALSE and a partial AUC is computed, specifies if <code>partial.auc</code> specifies the bounds in terms of specificity (default) or sensitivity. Can be shortened to <code>spec/sens</code> or even <code>sp/se</code> . Ignored if <code>partial.auc=FALSE</code> .
<code>partial.auc.correct</code>	logical indicating if the correction of AUC must be applied in order to have a maximal AUC of 1.0 and a non-discriminant AUC of 0.5 whatever the <code>partial.auc</code> defined. Ignored if <code>partial.auc=FALSE</code> . Default: FALSE.
<code>...</code>	further arguments passed to or from other methods, especially arguments for <code>roc</code> when calling <code>auc.default</code> , <code>auc.formula</code> , <code>auc.smooth.roc</code> . Note that the <code>auc</code> argument of <code>roc</code> is not allowed. Unused in <code>auc.roc</code> .

Details

This function is typically called from `roc` when `auc=TRUE` (default). It is also used by `ci`. When it is called with two vectors (`response, predictor`) or a formula (`response~predictor`) arguments, the `roc` function is called and only the AUC is returned.

By default the total area under the curve is computed, but a partial AUC (pAUC) can be specified with the `partial.auc` argument. It specifies the bounds of specificity or sensitivity (depending on `partial.auc.focus`) between which the AUC will be computed. As it specifies specificities or sensitivities, you must adapt it in relation to the 'percent' specification (see details in `roc`).

`partial.auc.focus` is ignored if `partial.auc=FALSE` (default). If a partial AUC is computed, `partial.auc.focus` specifies if the bounds specified in `partial.auc` must be interpreted as sensitivity or specificity. Any other value will produce an error. It is recommended to `plot` the ROC curve with `auc.polygon=TRUE` in order to make sure the specification is correct.

If a pAUC is defined, it can be standardized (corrected). This correction is controlled by the `partial.auc.correct` argument. If `partial.auc.correct=TRUE`, the correction by McClish will be applied:

$$\frac{1 + \frac{auc - min}{max - min}}{2}$$

where `auc` is the uncorrected pAUC computed in the region defined by `partial.auc`, `min` is the value of the non-discriminant AUC (with an AUC of 0.5 or 50 in the region and `max` is the maximum possible AUC in the region. With this correction, the AUC will be 0.5 if non discriminant and 1.0 if maximal, whatever the region defined. This correction is fully compatible with percent.

There is no difference in the computation of the area under a smoothed ROC curve.

Value

The numeric AUC value, of class `c("auc", "numeric")` (or `c("multiclass.auc", "numeric")` if a “multiclass.roc” was supplied), in fraction of the area or in percent if `percent=TRUE`, with the following attributes:

<code>partial.auc</code>	if the AUC is full (FALSE) or partial (and in this case the bounds), as defined in argument.
<code>partial.auc.focus</code>	only for a partial AUC, if the bound specifies the sensitivity or specificity, as defined in argument.
<code>partial.auc.correct</code>	only for a partial AUC, was it corrected? As defined in argument.
<code>percent</code>	whether the AUC is given in percent or fraction.
<code>roc</code>	the original ROC curve, as a “ roc ”, “ smooth.roc ” or “ multiclass.roc ” object.

Multi-class AUCs

With an object of class “multiclass.roc”, a multi-class AUC is computed as an average AUC as defined by Hand and Till (equation 7).

$$auc = \frac{2}{c(c-1)} \sum aucs$$

with aucs all the pairwise roc curves.

References

- Tom Fawcett (2006) “An introduction to ROC analysis”. *Pattern Recognition Letters* **27**, 861–874. DOI: [10.1016/j.patrec.2005.10.010](#).
- David J. Hand and Robert J. Till (2001). A Simple Generalisation of the Area Under the ROC Curve for Multiple Class Classification Problems. *Machine Learning* **45**(2), p. 171–186. DOI: [10.1023/A:1010920819831](#).
- Donna Katzman McClish (1989) “Analyzing a Portion of the ROC Curve”. *Medical Decision Making* **9**(3), 190–195. DOI: [10.1177/0272989X8900900307](#).
- Xavier Robin, Natacha Turck, Alexandre Hainard, *et al.* (2011) “pROC: an open-source package for R and S+ to analyze and compare ROC curves”. *BMC Bioinformatics*, **7**, 77. DOI: [10.1186/1471-2105-12-77](#).

See Also

[roc](#), [ci.auc](#)

Examples

```
data(aSAH)

# Syntax (response, predictor):
auc(aSAH$outcome, aSAH$s100b)

# With a roc object:
rocobj <- roc(aSAH$outcome, aSAH$s100b)
# Full AUC:
auc(rocobj)
# Partial AUC:
auc(rocobj, partial.auc=c(1, .8), partial.auc.focus="se", partial.auc.correct=TRUE)

# Alternatively, you can get the AUC directly from roc():
roc(aSAH$outcome, aSAH$s100b)$auc
roc(aSAH$outcome, aSAH$s100b,
    partial.auc=c(1, .8), partial.auc.focus="se",
    partial.auc.correct=TRUE)$auc
```

ci

Compute the confidence interval of a ROC curve

Description

This function computes the confidence interval (CI) of a ROC curve. The `of` argument controls the type of CI that will be computed. By default, the 95% CI are computed with 2000 stratified bootstrap replicates.

Usage

```
ci(...)
## S3 method for class 'roc'
ci(roc, of = c("auc", "thresholds", "sp", "se"), ...)
## S3 method for class 'smooth.roc'
ci(smooth.roc, of = c("auc", "sp", "se"), ...)
## S3 method for class 'formula'
ci(formula, data, ...)
## Default S3 method:
ci(response, predictor, ...)
```

Arguments

`roc, smooth.roc`
a “roc” object from the `roc` function, or a “smooth.roc” object from the `smooth.roc` function.

`response, predictor`
arguments for the `roc` function.

formula, data	a formula (and possibly a data object) of type response~predictor for the <code>roc</code> function.
of	The type of confidence interval. One of “auc”, “thresholds”, “sp” or “se”. Note that confidence interval on “thresholds” are not available for smoothed ROC curves.
...	further arguments passed to or from other methods, especially <code>auc</code> , <code>roc</code> , and the specific ci functions <code>ci.auc</code> , <code>ci.se</code> , <code>ci.sp</code> and <code>ci.thresholds</code> .

Details

`ci.formula` and `ci.default` are convenience methods that build the ROC curve (with the `roc` function) before calling `ci.roc`. You can pass them arguments for both `roc` and `ci.roc`. Simply use `ci` that will dispatch to the correct method.

This function is typically called from `roc` when `ci=TRUE` (not by default). Depending on the of argument, the specific ci functions `ci.auc`, `ci.thresholds`, `ci.sp` or `ci.se` are called.

Value

The return value of the specific ci functions `ci.auc`, `ci.thresholds`, `ci.sp` or `ci.se`, depending on the of argument.

References

Xavier Robin, Natacha Turck, Alexandre Hainard, *et al.* (2011) “pROC: an open-source package for R and S+ to analyze and compare ROC curves”. *BMC Bioinformatics*, 7, 77. DOI: [10.1186/1471-2105-12-77](https://doi.org/10.1186/1471-2105-12-77).

See Also

`roc`, `auc`, `ci.auc`, `ci.thresholds`, `ci.sp`, `ci.se`

Examples

```
data(aSAH)

# Syntax (response, predictor):
ci(aSAH$outcome, aSAH$s100b)

# With a roc object:
rocobj <- roc(aSAH$outcome, aSAH$s100b)

# Of an AUC
ci(rocobj)
ci(rocobj, of="auc")
# this is strictly equivalent to:
ci.auc(rocobj)

# Of thresholds, sp, se...
## Not run:
ci(rocobj, of="thresholds")
```

```

ci(rocobj, of="thresholds", thresholds=0.51)
ci(rocobj, of="thresholds", thresholds="all")
ci(rocobj, of="sp", sensitivities=c(.95, .9, .85))
ci(rocobj, of="se")

## End(Not run)

# Alternatively, you can get the CI directly from roc():
rocobj <- roc(aSAH$outcome, aSAH$s100b, ci=TRUE, of="auc")
rocobj$ci

```

ci.auc

*Compute the confidence interval of the AUC***Description**

This function computes the confidence interval (CI) of an area under the curve (AUC). By default, the 95% CI is computed with 2000 stratified bootstrap replicates.

Usage

```

# ci.auc(...)
## S3 method for class 'roc'
ci.auc(roc, conf.level=0.95, method=c("delong",
"bootstrap"), boot.n = 2000, boot.stratified = TRUE, reuse.auc=TRUE,
progress = getOption("pROCProgress")$name, ...)
## S3 method for class 'smooth.roc'
ci.auc(smooth.roc, conf.level=0.95, boot.n=2000,
boot.stratified=TRUE, reuse.auc=TRUE,
progress=getOption("pROCProgress")$name, ...)
## S3 method for class 'auc'
ci.auc(auc, ...)
## S3 method for class 'formula'
ci.auc(formula, data, ...)
## Default S3 method:
ci.auc(response, predictor, ...)

```

Arguments

roc, smooth.roc	a “roc” object from the roc function, or a “smooth.roc” object from the smooth.roc function.
auc	an “auc” object from the auc function.
response, predictor	arguments for the roc function.
formula, data	a formula (and possibly a data object) of type response~predictor for the roc function.

<code>conf.level</code>	the width of the confidence interval as [0,1], never in percent. Default: 0.95, resulting in a 95% CI.
<code>method</code>	the method to use, either “delong” or “bootstrap”. The first letter is sufficient. If omitted, the appropriate method is selected as explained in details.
<code>boot.n</code>	the number of bootstrap replicates. Default: 2000.
<code>boot.stratified</code>	should the bootstrap be stratified (default, same number of cases/controls in each replicate than in the original sample) or not.
<code>reuse.auc</code>	if TRUE (default) and the “roc” object contains an “auc” field, re-use these specifications for the test. If false, use optional ... arguments to auc . See details.
<code>progress</code>	the name of progress bar to display. Typically “none”, “win”, “tk” or “text” (see the name argument to create_progress_bar for more information), but a list as returned by create_progress_bar is also accepted. See also the “Progress bars” section of this package’s documentation .
<code>...</code>	further arguments passed to or from other methods, especially arguments for roc and <code>roc.test.roc</code> when calling <code>roc.test.default</code> or <code>roc.test.formula</code> . Arguments for auc and txtProgressBar (only char and style) if applicable.

Details

This function computes the CI of an AUC. Two methods are available: “delong” and “bootstrap” with the parameters defined in “roc\$auc” to compute a CI. When it is called with two vectors (response, predictor) or a formula (response~predictor) arguments, the [roc](#) function is called to build the ROC curve first.

Default is to use “delong” method except for comparison of partial AUC and smoothed curves, where bootstrap is used. Using “delong” for partial AUC and smoothed ROCs is not supported in pROC (with smoothed ROCs, method is ignored, otherwise for pAUC a warning is produced and “bootstrap” is employed instead).

With `method="bootstrap"`, the function calls [auc](#) `boot.n` times. For more details about the bootstrap, see the Bootstrap section in [this package’s documentation](#).

For [smoothed ROC curves](#), smoothing is performed again at each bootstrap replicate with the parameters originally provided. If a density smoothing was performed with user-provided `density.cases` or `density.controls` the bootstrap cannot be performed and an error is issued.

With `method="delong"`, the variance of the AUC is computed as defined by DeLong *et al.* (1988) and the CI is deduced with [qnorm](#).

Value

A numeric vector of length 3 and class “ci.auc”, with the lower bound, the median and the upper bound of the CI, and the following attributes:

<code>conf.level</code>	the width of the CI, in fraction.
<code>method</code>	the method employed.
<code>boot.n</code>	the number of bootstrap replicates.
<code>boot.stratified</code>	whether or not the bootstrapping was stratified.

auc an object of class “**auc**” stored for reference about the computed AUC details (partial, percent, ...)

The aucs item is not included in this list since version 1.2 for consistency reasons.

AUC specification

The comparison of the CI needs a specification of the AUC. This allows to compute the CI for full or partial AUCs. The specification is defined by:

1. the “auc” field in the “**roc**” object if `reuse.auc` is set to TRUE (default). It is naturally inherited from any call to **roc** and fits most cases.
2. passing the specification to **auc** with ... (arguments `partial.auc`, `partial.auc.correct` and `partial.auc.focus`). In this case, you must ensure either that the **roc** object do not contain an auc field (if you called **roc** with `auc=FALSE`), or set `reuse.auc=FALSE`.

If `reuse.auc=FALSE` the **auc** function will always be called with ... to determine the specification, even if the “**roc**” object do contain an auc field.

As well if the “**roc**” object do not contain an auc field, the **auc** function will always be called with ... to determine the specification.

Warning: if the roc object passed to ci contains an auc field and `reuse.auc=TRUE`, **auc** is not called and arguments such as `partial.auc` are silently ignored.

Warnings

If `method="delong"` and the AUC specification specifies a partial AUC, the warning “Using De-Long’s test for partial AUC is not supported. Using bootstrap test instead.” is issued. The method argument is ignored and “bootstrap” is used instead.

If `boot.stratified=FALSE` and the sample has a large imbalance between cases and controls, it could happen that one or more of the replicates contains no case or control observation, or that there are not enough points for smoothing, producing a NA area. The warning “NA value(s) produced during bootstrap were ignored.” will be issued and the observation will be ignored. If you have a large imbalance in your sample, it could be safer to keep `boot.stratified=TRUE`.

Errors

If `density.cases` and `density.controls` were provided for smoothing, the error “Cannot compute the statistic on ROC curves smoothed with `density.controls` and `density.cases`.” is issued.

References

Elisabeth R. DeLong, David M. DeLong and Daniel L. Clarke-Pearson (1988) “Comparing the areas under two or more correlated receiver operating characteristic curves: a nonparametric approach”. *Biometrics* **44**, 837–845.

James Carpenter and John Bithell (2000) “Bootstrap condence intervals: when, which, what? A practical guide for medical statisticians”. *Statistics in Medicine* **19**, 1141–1164.

Xavier Robin, Natacha Turck, Alexandre Hainard, *et al.* (2011) “pROC: an open-source package for R and S+ to analyze and compare ROC curves”. *BMC Bioinformatics*, **7**, 77. DOI: [10.1186/1471-2105-12-77](https://doi.org/10.1186/1471-2105-12-77).

See Also[roc](#), [auc](#), [ci](#)**Examples**

```

data(aSAH)

# Syntax (response, predictor):
ci.auc(aSAH$outcome, aSAH$s100b)

# With a roc object:
rocobj <- roc(aSAH$outcome, aSAH$s100b)
# default values
ci.auc(rocobj)
ci(rocobj)
ci(auc(rocobj))
ci(rocobj$auc)
ci(rocobj$auc, method="delong")

# Partial AUC and customized bootstrap:
ci.auc(aSAH$outcome, aSAH$s100b,
      boot.n=100, conf.level=0.9, stratified=FALSE, partial.auc=c(1, .8),
      partial.auc.focus="se", partial.auc.correct=TRUE)

# Note that the following will NOT give a CI of the partial AUC:
ci.auc(rocobj, boot.n=500, conf.level=0.9, stratified=FALSE,
      partial.auc=c(1, .8), partial.auc.focus="se", partial.auc.correct=TRUE)
# This is because rocobj$auc is not a partial AUC.
## Not run:
# You can overcome this problem with reuse.auc:
ci.auc(rocobj, boot.n=500, conf.level=0.9, stratified=FALSE,
      partial.auc=c(1, .8), partial.auc.focus="se", partial.auc.correct=TRUE,
      reuse.auc=FALSE)

## End(Not run)

# Alternatively, you can get the CI directly from roc():
rocobj <- roc(aSAH$outcome, aSAH$s100b, ci=TRUE, of="auc")
rocobj$ci

## Not run:
# On a smoothed ROC, the CI is re-computed automatically
smooth(rocobj)
# Or you can compute a new one:
ci.auc(smooth(rocobj, method="density", reuse.ci=FALSE), boot.n=100)

## End(Not run)

```

Description

This function computes the confidence interval (CI) of the sensitivity at the given specificity points. By default, the 95% CI are computed with 2000 stratified bootstrap replicates.

Usage

```
# ci.se(...)
## S3 method for class 'roc'
ci.se(roc, specificities = seq(0, 1, .1) * ifelse(roc$percent,
100, 1), conf.level=0.95, boot.n=2000, boot.stratified=TRUE,
progress=getOption("pROCProgress")$name, ...)
## S3 method for class 'smooth.roc'
ci.se(smooth.roc, specificities = seq(0, 1, .1) *
ifelse(smooth.roc$percent, 100, 1), conf.level=0.95, boot.n=2000,
boot.stratified=TRUE, progress=getOption("pROCProgress")$name, ...)
## S3 method for class 'formula'
ci.se(formula, data, ...)
## Default S3 method:
ci.se(response, predictor, ...)
```

Arguments

roc, smooth.roc	a “roc” object from the roc function, or a “smooth.roc” object from the smooth.roc function.
response, predictor	arguments for the roc function.
formula, data	a formula (and possibly a data object) of type response~predictor for the roc function.
specificities	on which specificities to evaluate the CI.
conf.level	the width of the confidence interval as [0,1], never in percent. Default: 0.95, resulting in a 95% CI.
boot.n	the number of bootstrap replicates. Default: 2000.
boot.stratified	should the bootstrap be stratified (default, same number of cases/controls in each replicate than in the original sample) or not.
progress	the name of progress bar to display. Typically “none”, “win”, “tk” or “text” (see the name argument to create_progress_bar for more information), but a list as returned by create_progress_bar is also accepted. See also the “Progress bars” section of this package’s documentation .
...	further arguments passed to or from other methods, especially arguments for roc and ci.se.roc when calling ci.se.default or ci.se.formula . Arguments for txtProgressBar (only char and style) if applicable.

Details

`ci.se.formula` and `ci.se.default` are convenience methods that build the ROC curve (with the `roc` function) before calling `ci.se.roc`. You can pass them arguments for both `roc` and `ci.se.roc`. Simply use `ci.se` that will dispatch to the correct method.

The `ci.se.roc` function creates `boot.n` bootstrap replicate of the ROC curve, and evaluates the sensitivity at specificities given by the `specificities` argument. Then it computes the confidence interval as the percentiles given by `conf.level`.

For more details about the bootstrap, see the Bootstrap section in [this package's documentation](#).

For [smoothed ROC curves](#), smoothing is performed again at each bootstrap replicate with the parameters originally provided. If a density smoothing was performed with user-provided `density.cases` or `density.controls` the bootstrap cannot be performed and an error is issued.

Value

A matrix of CI for the given sensitivities. Row (names) are the specificities, the first column the lower bound, the 2nd column the median and the 3rd column the upper bound.

Additionally, the list has the following attributes:

<code>conf.level</code>	the width of the CI, in fraction.
<code>boot.n</code>	the number of bootstrap replicates.
<code>boot.stratified</code>	whether or not the bootstrapping was stratified.
<code>specificities</code>	the specificities as given in argument.
<code>roc</code>	the object of class “ <code>roc</code> ” that was used to compute the CI.

Warnings

If `boot.stratified=FALSE` and the sample has a large imbalance between cases and controls, it could happen that one or more of the replicates contains no case or control observation, or that there are not enough points for smoothing, producing a NA area. The warning “NA value(s) produced during bootstrap were ignored.” will be issued and the observation will be ignored. If you have a large imbalance in your sample, it could be safer to keep `boot.stratified=TRUE`.

Errors

If `density.cases` and `density.controls` were provided for smoothing, the error “Cannot compute the statistic on ROC curves smoothed with `density.controls` and `density.cases`.” is issued.

References

- Tom Fawcett (2006) “An introduction to ROC analysis”. *Pattern Recognition Letters* **27**, 861–874. DOI: [10.1016/j.patrec.2005.10.010](https://doi.org/10.1016/j.patrec.2005.10.010).
- James Carpenter and John Bithell (2000) “Bootstrap condence intervals: when, which, what? A practical guide for medical statisticians”. *Statistics in Medicine* **19**, 1141–1164.
- Xavier Robin, Natacha Turck, Alexandre Hainard, *et al.* (2011) “pROC: an open-source package for R and S+ to analyze and compare ROC curves”. *BMC Bioinformatics*, **7**, 77. DOI: [10.1186/1471-2105-12-77](https://doi.org/10.1186/1471-2105-12-77).

See Also

[roc](#), [ci](#), [ci.sp](#), [plot.ci](#)

Examples

```
data(aSAH)

## Not run:
# Syntax (response, predictor):
ci.se(aSAH$outcome, aSAH$s100b)

# With a roc object and less bootstrap:
rocobj <- roc(aSAH$outcome, aSAH$s100b)
ci.se(rocobj, boot.n=100)

# Customized bootstrap and specific specificities:
ci.se(rocobj, c(.95, .9, .85), boot.n=500, conf.level=0.9, stratified=FALSE)

## End(Not run)

# Alternatively, you can get the CI directly from roc():
rocobj <- roc(aSAH$outcome,
              aSAH$s100b, ci=TRUE, of="se", boot.n=100)
rocobj$ci

# Plotting the CI
plot(rocobj)
plot(rocobj$ci)

## Not run:
# On a smoothed ROC, the CI is re-computed automatically
smooth(rocobj)
# Or you can compute a new one:
ci.se(smooth(rocobj, method="density", reuse.ci=FALSE), boot.n=100)

## End(Not run)
```

ci.sp

Compute the confidence interval of specificities at given sensitivities

Description

This function computes the confidence interval (CI) of the specificity at the given sensitivity points. By default, the 95% CI are computed with 2000 stratified bootstrap replicates.

Usage

```
# ci.sp(...)
## S3 method for class 'roc'
```

```

ci.sp(roc, sensitivities = seq(0, 1, .1) * ifelse(roc$percent,
100, 1), conf.level=0.95, boot.n=2000, boot.stratified=TRUE,
progress=getOption("pROCProgress")$name, ...)
## S3 method for class 'smooth.roc'
ci.sp(smooth.roc, sensitivities = seq(0, 1, .1) *
ifelse(smooth.roc$percent, 100, 1), conf.level=0.95, boot.n=2000,
boot.stratified=TRUE, progress=getOption("pROCProgress")$name, ...)
## S3 method for class 'formula'
ci.sp(formula, data, ...)
## Default S3 method:
ci.sp(response, predictor, ...)

```

Arguments

roc, smooth.roc	a “roc” object from the roc function, or a “smooth.roc” object from the smooth.roc function.
response, predictor	arguments for the roc function.
formula, data	a formula (and possibly a data object) of type response~predictor for the roc function.
sensitivities	on which sensitivities to evaluate the CI.
conf.level	the width of the confidence interval as [0,1], never in percent. Default: 0.95, resulting in a 95% CI.
boot.n	the number of bootstrap replicates. Default: 2000.
boot.stratified	should the bootstrap be stratified (default, same number of cases/controls in each replicate than in the original sample) or not.
progress	the name of progress bar to display. Typically “none”, “win”, “tk” or “text” (see the name argument to create_progress_bar for more information), but a list as returned by create_progress_bar is also accepted. See also the “Progress bars” section of this package’s documentation .
...	further arguments passed to or from other methods, especially arguments for roc and ci.sp.roc when calling ci.sp.default or ci.sp.formula . Arguments for txtProgressBar (only char and style) if applicable.

Details

[ci.sp.formula](#) and [ci.sp.default](#) are convenience methods that build the ROC curve (with the [roc](#) function) before calling [ci.sp.roc](#). You can pass them arguments for both [roc](#) and [ci.sp.roc](#). Simply use [ci.sp](#) that will dispatch to the correct method.

The [ci.sp.roc](#) function creates [boot.n](#) bootstrap replicate of the ROC curve, and evaluates the specificity at sensitivities given by the [sensitivities](#) argument. Then it computes the confidence interval as the percentiles given by [conf.level](#).

For more details about the bootstrap, see the Bootstrap section in [this package’s documentation](#).

For [smoothed ROC curves](#), smoothing is performed again at each bootstrap replicate with the parameters originally provided. If a density smoothing was performed with user-provided `density.cases` or `density.controls` the bootstrap cannot be performed and an error is issued.

Value

A matrix of CI for the given specificities. Row (names) are the sensitivities, the first column the lower bound, the 2nd column the median and the 3rd column the upper bound.

Additionally, the list has the following attributes:

<code>conf.level</code>	the width of the CI, in fraction.
<code>boot.n</code>	the number of bootstrap replicates.
<code>boot.stratified</code>	whether or not the bootstrapping was stratified.
<code>sensitivities</code>	the sensitivities as given in argument.
<code>roc</code>	the object of class “ roc ” that was used to compute the CI.

Warnings

If `boot.stratified=FALSE` and the sample has a large imbalance between cases and controls, it could happen that one or more of the replicates contains no case or control observation, or that there are not enough points for smoothing, producing a NA area. The warning “NA value(s) produced during bootstrap were ignored.” will be issued and the observation will be ignored. If you have a large imbalance in your sample, it could be safer to keep `boot.stratified=TRUE`.

Errors

If `density.cases` and `density.controls` were provided for smoothing, the error “Cannot compute the statistic on ROC curves smoothed with `density.controls` and `density.cases`.” is issued.

References

- Tom Fawcett (2006) “An introduction to ROC analysis”. *Pattern Recognition Letters* **27**, 861–874. DOI: [10.1016/j.patrec.2005.10.010](#).
- James Carpenter and John Bithell (2000) “Bootstrap condence intervals: when, which, what? A practical guide for medical statisticians”. *Statistics in Medicine* **19**, 1141–1164.
- Xavier Robin, Natacha Turck, Alexandre Hainard, *et al.* (2011) “pROC: an open-source package for R and S+ to analyze and compare ROC curves”. *BMC Bioinformatics*, **7**, 77. DOI: [10.1186/1471-2105-12-77](#).

See Also

[roc](#), [ci](#), [ci.se](#), [plot.ci](#)

Examples

```

data(aSAH)

## Not run:
# Syntax (response, predictor):
ci.sp(aSAH$outcome, aSAH$s100b)

# With a roc object:
rocobj <- roc(aSAH$outcome, aSAH$s100b)
ci.sp(rocobj)

# Customized bootstrap and specific specificities:
ci.sp(rocobj, c(.95, .9, .85), boot.n=500, conf.level=0.9, stratified=FALSE)

## End(Not run)

# Alternatively, you can get the CI directly from roc():
rocobj <- roc(aSAH$outcome,
              aSAH$s100b, ci=TRUE, of="sp", boot.n=100)
rocobj$ci

# Plotting the CI
plot(rocobj)
plot(rocobj$ci)

## Not run:
# On a smoothed ROC, the CI is re-computed automatically
smooth(rocobj)
# Or you can compute a new one:
ci.sp(smooth(rocobj, method="density", reuse.ci=FALSE), boot.n=100)

## End(Not run)

```

ci.thresholds

Compute the confidence interval of thresholds

Description

This function computes the confidence interval (CI) of the sensitivity and specificity of the thresholds given in argument. By default, the 95% CI are computed with 2000 stratified bootstrap replicates.

Usage

```

# ci.thresholds(...)
## S3 method for class 'roc'
ci.thresholds(roc, conf.level=0.95, boot.n=2000,
boot.stratified=TRUE, thresholds = "local maximas",
progress=getOption("pROCProgress")$name, ...)

```

```
## S3 method for class 'formula'
ci.thresholds(formula, data, ...)
## S3 method for class 'smooth.roc'
ci.thresholds(smooth.roc, ...)
## Default S3 method:
ci.thresholds(response, predictor, ...)
```

Arguments

roc	a “roc” object from the roc function.
smooth.roc	not available for smoothed ROC curves, available only to catch the error and provide a clear error message.
response, predictor	arguments for the roc function.
formula, data	a formula (and possibly a data object) of type response~predictor for the roc function.
conf.level	the width of the confidence interval as [0,1], never in percent. Default: 0.95, resulting in a 95% CI.
boot.n	the number of bootstrap replicates. Default: 2000.
boot.stratified	should the bootstrap be stratified (default, same number of cases/controls in each replicate than in the original sample) or not.
thresholds	on which thresholds to evaluate the CI. Either the numeric values of the thresholds, a logical vector (as index of <code>roc\$thresholds</code>) or a character “all”, “local maximas” or “best”.
progress	the name of progress bar to display. Typically “none”, “win”, “tk” or “text” (see the name argument to create_progress_bar for more information), but a list as returned by create_progress_bar is also accepted. See also the “Progress bars” section of this package’s documentation .
...	further arguments passed to or from other methods, especially arguments for roc and <code>ci.thresholds.roc</code> when calling <code>ci.thresholds.default</code> or <code>ci.thresholds.formula</code> . Arguments for txtProgressBar (only char and style) if applicable.

Details

`ci.thresholds.formula` and `ci.thresholds.default` are convenience methods that build the ROC curve (with the [roc](#) function) before calling `ci.thresholds.roc`. You can pass them arguments for both [roc](#) and `ci.thresholds.roc`. Simply use `ci.thresholds` that will dispatch to the correct method.

This function creates `boot.n` bootstrap replicate of the ROC curve, and evaluates the sensitivity and specificity at thresholds given by the `thresholds` argument. Then it computes the confidence interval as the percentiles given by `conf.level`.

For more details about the bootstrap, see the Bootstrap section in [this package’s documentation](#).

Value

A list of length 2 and class “ci.thresholds”, with the confidence intervals of the CI and the following items:

specificity	a matrix of CI for the specificity. Row (names) are the thresholds, the first column the lower bound, the 2nd column the median and the 3rd column the upper bound.
sensitivity	same than specificity.

Additionally, the list has the following attributes:

conf.level	the width of the CI, in fraction.
boot.n	the number of bootstrap replicates.
boot.stratified	whether or not the bootstrapping was stratified.
thresholds	the thresholds, as given in argument.
roc	the object of class “roc” that was used to compute the CI.

Warnings

If boot.stratified=FALSE and the sample has a large imbalance between cases and controls, it could happen that one or more of the replicates contains no case or control observation, producing a NA area. The warning “NA value(s) produced during bootstrap were ignored.” will be issued and the observation will be ignored. If you have a large imbalance in your sample, it could be safer to keep boot.stratified=TRUE.

References

Tom Fawcett (2006) “An introduction to ROC analysis”. *Pattern Recognition Letters* **27**, 861–874. DOI: [10.1016/j.patrec.2005.10.010](https://doi.org/10.1016/j.patrec.2005.10.010).

James Carpenter and John Bithell (2000) “Bootstrap condence intervals: when, which, what? A practical guide for medical statisticians”. *Statistics in Medicine* **19**, 1141–1164.

Xavier Robin, Natacha Turck, Alexandre Hainard, *et al.* (2011) “pROC: an open-source package for R and S+ to analyze and compare ROC curves”. *BMC Bioinformatics*, **7**, 77. DOI: [10.1186/1471-2105-12-77](https://doi.org/10.1186/1471-2105-12-77).

See Also

[roc](#), [ci](#)

Examples

```
data(aSAH)

## Not run:
# Syntax (response, predictor):
ci.thresholds(aSAH$outcome, aSAH$s100b)
```

```

# With a roc object:
rocobj <- roc(aSAH$outcome, aSAH$s100b)
ci.thresholds(rocobj)

# Customized bootstrap and specific thresholds:
ci.thresholds(aSAH$outcome, aSAH$s100b,
              boot.n=500, conf.level=0.9, stratified=FALSE,
              thresholds=c(0.5, 1, 2))

## End(Not run)

# Alternatively, you can get the CI directly from roc():
rocobj <- roc(aSAH$outcome,
              aSAH$s100b, ci=TRUE, of="thresholds")
rocobj$ci

# Plotting the CI
plot(rocobj)
plot(rocobj$ci)

```

coords

Coordinates of a ROC curve

Description

This function returns the coordinates of the ROC curve at the specified point.

Usage

```

coords(...)
## S3 method for class 'roc'
coords(roc, x, input=c("threshold", "specificity",
                       "sensitivity"), ret=c("threshold", "specificity", "sensitivity"),
       as.list=FALSE, drop=TRUE, best.method=c("youden", "closest.topleft"),
       best.weights=c(1, 0.5), ...)
## S3 method for class 'smooth.roc'
coords(smooth.roc, x, input=c("specificity",
                              "sensitivity"), ret=c("specificity", "sensitivity"), as.list=FALSE,
       drop=TRUE, best.method=c("youden", "closest.topleft"),
       best.weights=c(1, 0.5), ...)

```

Arguments

roc, smooth.roc	a “roc” object from the roc function, or a “smooth.roc” object from the smooth.roc function.
x	the coordinates to look for. Numeric (if so, their meaning is defined by the input argument) or one of “all” (all the points of the ROC curve), “local maximas” (the local maximas of the ROC curve) or “best” (the point with the best sum of sensitivity and specificity).

<code>input</code>	If <code>x</code> is numeric, the kind of input coordinate (<code>x</code>). One of “threshold”, “specificity” or “sensitivity”. Can be shortened (for example to “thr”, “sens” and “spec”, or even to “t”, “se” and “sp”). Note that “threshold” is not allowed in <code>coords.smooth.roc</code> , and that the argument is ignored when <code>x</code> is a character.
<code>ret</code>	The coordinates to return. One or more of “threshold”, “specificity”, “sensitivity”, “accuracy”, “tn” (true negative count), “tp” (true positive count), “fn” (false negative count), “fp” (false positive count), “npv” (negative predictive value), “ppv” (positive predictive value). “1-specificity”, “1-sensitivity”, “1-npv” and “1-ppv” are recognized as well, and must be used verbatim in ROC curves with <code>percent=TRUE</code> (for instance “100-ppv” is never accepted). Values can be shortened (for example to “thr”, “sens” and “spec”, or even to “se”, “sp” or “1-np”). Note that “threshold” is not allowed in <code>coords.smooth.roc</code> . In addition, “npe” is replaced by “1-npv” and “ppe” by “1-ppv” (but they cannot be shortened).
<code>as.list</code>	If the returned object must be a list. If <code>FALSE</code> (default), a named numeric vector is returned.
<code>drop</code>	If <code>TRUE</code> the result is coerced to the lowest possible dimension, as per Extract . With <code>FALSE</code> if <code>x</code> is of length 1, the object returned will have the same format than if <code>x</code> was of length > 1.
<code>best.method</code>	if <code>x="best"</code> , the method to determine the best threshold. See details in the ‘Best thresholds’ section.
<code>best.weights</code>	if <code>x="best"</code> , the weights to determine the best threshold. See details in the ‘Best thresholds’ section.
<code>...</code>	further arguments passed to or from other methods. Ignored.

Details

This function takes a “roc” or “smooth.roc” object as first argument, on which the coordinates will be determined. The coordinates are defined by the `x` and `input` arguments. “threshold” coordinates cannot be determined in a smoothed ROC.

If `input="threshold"`, the coordinates for the threshold are reported, even if the exact threshold do not define the ROC curve. The following convenience characters are allowed: “all”, “local maximas” and “best”. They will return all the thresholds, only the thresholds defining local maximas (upper angles of the ROC curve), or only the threshold(s) corresponding to the best sum of sensitivity + specificity respectively. Note that “best” can return more than one threshold. If `x` is a character, the coordinates are limited to the thresholds within the partial AUC if it has been defined, and not necessarily to the whole curve.

For `input="specificity"` and `input="sensitivity"`, the function checks if the specificity or sensitivity is one of the points of the ROC curve (in `roc$sensitivities` or `roc$specificities`). More than one point may match (in *step* curves), then only the upper-left-most point coordinates are returned. Otherwise, the specificity and sensitivity of the point is interpolated and NA is returned as threshold.

The `coords` function in this package is a generic, but it might be superseded by functions in other packages such as **colorspace** or **spatstat** if they are loaded after **pROC**. In this case, call the `coords.roc` or `coords.smooth.roc` functions directly.

Value

Depending on the length of `x` and `as.list` argument.

	<code>length(x) == 1</code>	<code>length(x) > 1</code> or <code>drop == FALSE</code>
<code>as.list=TRUE</code>	a list of the length of, in the order of, and named after, <code>ret</code> .	a list of the length of, and named
<code>as.list=FALSE</code>	a numeric vector of the length of, in the order of, and named after, <code>ret</code> .	a numeric matrix with one row for

In all cases if `input="specificity"` or `input="sensitivity"` and interpolation was required, `threshold` is returned as NA.

Note that if giving a character as `x` ("all", "local maximas" or "best"), you cannot predict the dimension of the return value unless `drop=FALSE`. Even "best" may return more than one value (for example if the ROC curve is below the identity line, both extreme points).

`coords` may also return NULL when there a partial area is defined but no point of the ROC curve falls within the region.

Best thresholds

If `x="best"`, the `best.method` argument controls how the optimal threshold is determined.

“youden” Youden’s J statistic (Youden, 1950) is employed. The optimal cut-off is the threshold that maximizes the distance to the identity (diagonal) line. Can be shortened to “y”.

The optimality criterion is:

$$\max(sensitivities + specificities)$$

“closest.topleft” The optimal threshold is the point closest to the top-left part of the plot with perfect sensitivity or specificity. Can be shortened to “c” or “t”.

The optimality criterion is:

$$\min((1 - sensitivities)^2 + (1 - specificities)^2)$$

In addition, weights can be supplied if false positive and false negative predictions are not equivalent: a numeric vector of length 2 to the `best.weights` argument. The indices define

1. the cost of of a false negative classification
2. the prevalence, or the proportion of cases in the total population ($\frac{n_{cases}}{n_{controls} + n_{cases}}$).

The optimality criteria are modified as proposed by Perkins and Schisterman:

“youden”

$$\max(sensitivities + r * specificities)$$

“closest.topleft”

$$\min((1 - sensitivities)^2 + r * (1 - specificities)^2)$$

with

$$r = \frac{1 - prevalence}{cost * prevalence}$$

By default, prevalence is 0.5 and cost is 1 so that no weight is applied in effect.

Note that several thresholds might be equally optimal.

References

- Neil J. Perkins, Enrique F. Schisterman (2006) “The Inconsistency of "Optimal" Cutpoints Obtained using Two Criteria based on the Receiver Operating Characteristic Curve”. *American Journal of Epidemiology* **163**(7), 670–675. DOI: [10.1093/aje/kwj063](https://doi.org/10.1093/aje/kwj063).
- Xavier Robin, Natacha Turck, Alexandre Hainard, *et al.* (2011) “pROC: an open-source package for R and S+ to analyze and compare ROC curves”. *BMC Bioinformatics*, **7**, 77. DOI: [10.1186/1471-2105-12-77](https://doi.org/10.1186/1471-2105-12-77).
- W. J. Youden (1950) “Index for rating diagnostic tests”. *Cancer*, **3**, 32–35. DOI: [10.1002/1097-0142\(1950\)3:1<32::AID-CNCR2820030106>3.0.CO;2-3](https://doi.org/10.1002/1097-0142(1950)3:1<32::AID-CNCR2820030106>3.0.CO;2-3).

See Also

[roc](#)

Examples

```
data(aSAH)

# Print a roc object:
rocobj <- roc(aSAH$outcome, aSAH$s100b)

coords(rocobj, 0.55)
coords(rocobj, 0.9, "specificity", as.list=TRUE)
coords(rocobj, 0.5, "se", ret="se")
# fully qualified but identical:
coords(roc=rocobj, x=0.5, input="sensitivity", ret="sensitivity")

# Compare with drop=FALSE
coords(rocobj, 0.55, drop=FALSE)
coords(rocobj, 0.9, "specificity", as.list=TRUE, drop=FALSE)

# Same in percent
rocobj <- roc(aSAH$outcome, aSAH$s100b, percent=TRUE)

coords(rocobj, 0.55)
coords(rocobj, 90, "specificity", as.list=TRUE)
coords(rocobj, x=50, input="sensitivity", ret=c("sen", "spec"))

# And with a smoothed ROC curve
coords(smooth(rocobj), 90, "specificity")
coords(smooth(rocobj), 90, "specificity", drop=FALSE)
coords(smooth(rocobj), 90, "specificity", as.list=TRUE)
coords(smooth(rocobj), 90, "specificity", as.list=TRUE, drop=FALSE)

# Get the sensitivities for all thresholds
sensitivities <- coords(rocobj, rocobj$thresholds, "thr", "se")
# This is equivalent to taking sensitivities from rocobj directly
stopifnot(all.equal(as.vector(rocobj$sensitivities), as.vector(sensitivities)))
# You could also write:
sensitivities <- coords(rocobj, "all", ret="se")
stopifnot(all.equal(as.vector(rocobj$sensitivities), as.vector(sensitivities)))
```

```

# Get the best threshold
coords(rocobj, "b", ret="t")

# Get the best threshold according to different methods
rocobj <- roc(aSAH$outcome, aSAH$ndka, percent=TRUE)
coords(rocobj, "b", ret="t", best.method="youden") # default
coords(rocobj, "b", ret="t", best.method="closest.topleft")
# and with different weights
coords(rocobj, "b", ret="t", best.method="youden", best.weights=c(50, 0.2))
coords(rocobj, "b", ret="t", best.method="closest.topleft", best.weights=c(5, 0.2))
# and plot them
plot(rocobj, print.thres="best", print.thres.best.method="youden")
plot(rocobj, print.thres="best", print.thres.best.method="closest.topleft")
plot(rocobj, print.thres="best", print.thres.best.method="youden",
      print.thres.best.weights=c(50, 0.2))
plot(rocobj, print.thres="best", print.thres.best.method="closest.topleft",
      print.thres.best.weights=c(5, 0.2))

# Return more values:
coords(rocobj, "best", ret=c("threshold", "specificity", "sensitivity", "accuracy",
                             "tn", "tp", "fn", "fp", "npv", "ppv", "1-specificity",
                             "1-sensitivity", "1-npv", "1-ppv"))
coords(smooth(rocobj), "best", ret=c("threshold", "specificity", "sensitivity", "accuracy",
                                     "tn", "tp", "fn", "fp", "npv", "ppv", "1-specificity",
                                     "1-sensitivity", "1-npv", "1-ppv"))
coords(smooth(rocobj), 0.5, ret=c("threshold", "specificity", "sensitivity", "accuracy",
                                  "tn", "tp", "fn", "fp", "npv", "ppv", "1-specificity",
                                  "1-sensitivity", "1-npv", "1-ppv"))

```

cov.roc

Covariance of two paired ROC curves

Description

This function computes the covariance between the AUC of two correlated (or paired) ROC curves.

Usage

```

cov(...)
## Default S3 method:
cov(...)
## S3 method for class 'auc'
cov(roc1, roc2, ...)
## S3 method for class 'smooth.roc'
cov(roc1, roc2, ...)
## S3 method for class 'roc'
cov(roc1, roc2, method=c("delong", "bootstrap"),
     reuse.auc=TRUE, boot.n=2000, boot.stratified=TRUE, boot.return=FALSE,
     progress=getOption("pROCProgress")$name, ...)

```

Arguments

<code>roc1, roc2</code>	the two ROC curves on which to compute the covariance. Either “roc”, “auc” or “smooth.roc” objects (types can be mixed as long as the original ROC curve are paired).
<code>method</code>	the method to use, either “delong” or “bootstrap”. The first letter is sufficient. If omitted, the appropriate method is selected as explained in details.
<code>reuse.auc</code>	if TRUE (default) and the “roc” objects contain an “auc” field, re-use these specifications for the test. See details.
<code>boot.n</code>	for method=“bootstrap” only: the number of bootstrap replicates or permutations. Default: 2000.
<code>boot.stratified</code>	for method=“bootstrap” only: should the bootstrap be stratified (same number of cases/controls in each replicate than in the original sample) or not. Default: TRUE.
<code>boot.return</code>	if TRUE and method=“bootstrap”, also return the bootstrapped values. See the “Value” section for more details.
<code>progress</code>	the name of progress bar to display. Typically “none”, “win”, “tk” or “text” (see the name argument to create_progress_bar for more information), but a list as returned by create_progress_bar is also accepted. See also the “Progress bars” section of this package’s documentation .
<code>...</code>	further arguments passed to or from other methods, especially arguments for <code>cov.roc</code> when calling <code>cov</code> , <code>cov.auc</code> or <code>cov.smooth.roc</code> . Arguments for auc (if <code>reuse.auc=FALSE</code>) and txtProgressBar (only char and style) if applicable.

Details

This function computes the covariance between the AUC of two correlated (or paired, according to the detection of [are.paired](#)) ROC curves. It is typically called with the two [roc](#) objects of interest. Two methods are available: “delong” and “bootstrap” (see “Computational details” section below).

The default is to use “delong” method except with partial AUC and smoothed curves where “bootstrap” is employed. Using “delong” for partial AUC and smoothed ROCs is not supported (a warning is produced and “bootstrap” is employed instead).

For [smoothed ROC curves](#), smoothing is performed again at each bootstrap replicate with the parameters originally provided. If a density smoothing was performed with user-provided `density.cases` or `density.controls` the bootstrap cannot be performed and an error is issued.

`cov.default` forces the usage of the [cov](#) function in the **stats** package, so that other code relying on `cov` should continue to function normally.

Value

The numeric value of the covariance.

If `boot.return=TRUE` and `method=“bootstrap”`, an attribute `resampled.values` is set with the resampled (bootstrapped) values. It contains a matrix with the columns representing the two ROC curves, and the rows the `boot.n` bootstrap replicates.

AUC specification

To compute the covariance of the AUC of the ROC curves, `cov` needs a specification of the AUC. The specification is defined by:

1. the “auc” field in the “roc” objects if `reuse.auc` is set to `TRUE` (default)
2. passing the specification to `auc` with ... (arguments `partial.auc`, `partial.auc.correct` and `partial.auc.focus`). In this case, you must ensure either that the `roc` object do not contain an `auc` field (if you called `roc` with `auc=FALSE`), or set `reuse.auc=FALSE`.

If `reuse.auc=FALSE` the `auc` function will always be called with ... to determine the specification, even if the “roc” objects do contain an `auc` field.

As well if the “roc” objects do not contain an `auc` field, the `auc` function will always be called with ... to determine the specification.

Warning: if the roc object passed to `roc.test` contains an `auc` field and `reuse.auc=TRUE`, `auc` is not called and arguments such as `partial.auc` are silently ignored.

Computation details

With `method="bootstrap"`, the processing is done as follow:

1. `boot.n` bootstrap replicates are drawn from the data. If `boot.stratified` is `TRUE`, each replicate contains exactly the same number of controls and cases than the original sample, otherwise if `FALSE` the numbers can vary.
2. for each bootstrap replicate, the AUC of the two ROC curves are computed and stored.
3. the variance (as per `var.roc`) of the resampled AUCs and their covariance are assessed in a single bootstrap pass.
4. The following formula is used to compute the final covariance: $Var[AUC1] + Var[AUC2] - 2cov[AUC1, AUC2]$

With `method="delong"`, the processing is done as described in Hanley and Hajian-Tilaki (1997).

Errors

If `density.cases` and `density.controls` were provided for smoothing, the error “Cannot compute the covariance on ROC curves smoothed with `density.controls` and `density.cases`.” is issued.

Warnings

If “auc” specifications are different in both roc objects, the warning “Different AUC specifications in the ROC curves. Enforcing the inconsistency, but unexpected results may be produced.” is issued. Unexpected results may be produced.

If one or both ROC curves are “smooth.roc” objects with different smoothing specifications, the warning “Different smoothing parameters in the ROC curves. Enforcing the inconsistency, but unexpected results may be produced.” is issued. This warning can be benign, especially if ROC curves were generated with `roc(..., smooth=TRUE)` with different arguments to other functions (such as `plot`), or if you really want to compare two ROC curves smoothed differently.

If `method="delong"` and the AUC specification specifies a partial AUC, the warning “Using DeLong for partial AUC is not supported. Using bootstrap test instead.” is issued. The `method` argument is ignored and “bootstrap” is used instead.

If `method="delong"` and the ROC curve is smoothed, the warning “Using DeLong for smoothed ROCs is not supported. Using bootstrap instead.” is issued. The `method` argument is ignored and “bootstrap” is used instead.

DeLong ignores the direction of the ROC curve so that if two ROC curves have a different direction, the warning “DeLong should not be applied to ROC curves with a different direction.” is printed. However, the spurious computation is enforced.

If `boot.stratified=FALSE` and the sample has a large imbalance between cases and controls, it could happen that one or more of the replicates contains no case or control observation, or that there are not enough points for smoothing, producing a NA area. The warning “NA value(s) produced during bootstrap were ignored.” will be issued and the observation will be ignored. If you have a large imbalance in your sample, it could be safer to keep `boot.stratified=TRUE`.

Messages

The covariance can only be computed on paired data. This assumption is enforced by [are.paired](#). If the ROC curves are not paired, the covariance is 0 and the message “ROC curves are unpaired.” is printed. If your ROC curves are paired, make sure they fit [are.paired](#) criteria.

References

Elisabeth R. DeLong, David M. DeLong and Daniel L. Clarke-Pearson (1988) “Comparing the areas under two or more correlated receiver operating characteristic curves: a nonparametric approach”. *Biometrics* **44**, 837–845.

James A. Hanley and Karim O. Hajian-Tilaki (1997) “Sampling variability of nonparametric estimates of the areas under receiver operating characteristic curves: An update”. *Academic Radiology* **4**, 49–58. DOI: [10.1016/S1076-6332\(97\)80161-4](https://doi.org/10.1016/S1076-6332(97)80161-4).

See Also

[roc](#), [var.roc](#)

Examples

```
data(aSAH)

# Basic example with 2 roc objects
roc1 <- roc(aSAH$outcome, aSAH$s100b)
roc2 <- roc(aSAH$outcome, aSAH$wfns)
cov(roc1, roc2)

## Not run:
# The latter used DeLong. To use bootstrap:
cov(roc1, roc2, method="bootstrap")
# Decrease boot.n for a faster execution:
cov(roc1, roc2, method="bootstrap", boot.n=1000)
```

```

## End(Not run)

## Not run:
# Comparison can be done on smoothed ROCs
# Smoothing is re-done at each iteration, and execution is slow
cov(smooth(roc1), smooth(roc2))

## End(Not run)
# or from an AUC (no smoothing)
cov(auc(roc1), roc2)

## Not run:
# With bootstrap and return.values, one can compute the variances of the
# ROC curves in one single bootstrap run:
cov.rocs <- cov(roc1, roc2, method="bootstrap", boot.return=TRUE)
# var(roc1):
var(attr(cov.rocs, "resampled.values")[,1])
# var(roc2):
var(attr(cov.rocs, "resampled.values")[,2])

## End(Not run)

## Not run:
# Covariance of partial AUC:
roc3 <- roc(aSAH$outcome, aSAH$s100b, partial.auc=c(1, 0.8), partial.auc.focus="se")
roc4 <- roc(aSAH$outcome, aSAH$wfns, partial.auc=c(1, 0.8), partial.auc.focus="se")
cov(roc3, roc4)
# This is strictly equivalent to:
cov(roc3, roc4, method="bootstrap")

# Alternatively, we could re-use roc1 and roc2 to get the same result:
cov(roc1, roc2, reuse.auc=FALSE, partial.auc=c(1, 0.8), partial.auc.focus="se")

## End(Not run)

# Spurious use of DeLong's test with different direction:
roc5 <- roc(aSAH$outcome, aSAH$s100b, direction="<")
roc6 <- roc(aSAH$outcome, aSAH$s100b, direction=">")
cov(roc5, roc6, method="delong")

## Test data from Hanley and Hajian-Tilaki, 1997
disease.present <- c("Yes", "No", "Yes", "No", "No", "Yes", "Yes", "No", "No", "Yes", "No", "No", "Yes", "No", "No")
field.strength.1 <- c(1, 2, 5, 1, 1, 1, 2, 1, 2, 2, 1, 1, 5, 1, 1)
field.strength.2 <- c(1, 1, 5, 1, 1, 1, 4, 1, 2, 2, 1, 1, 5, 1, 1)
roc7 <- roc(disease.present, field.strength.1)
roc8 <- roc(disease.present, field.strength.2)
# Assess the covariance:
cov(roc7, roc8)

## Not run:
# With bootstrap:
cov(roc7, roc8, method="bootstrap")

```

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

has.partial.auc	<i>Does the ROC curve have a partial AUC?</i>
-----------------	---

Description

This function determines if the ROC curve has a partial AUC.

Usage

```
has.partial.auc(roc)
## S3 method for class 'auc'
has.partial.auc(roc)
## S3 method for class 'smooth.roc'
has.partial.auc(roc)
## S3 method for class 'roc'
has.partial.auc(roc)
```

Arguments

roc the ROC curve to check.

Value

TRUE if the AUC is a partial AUC, FALSE otherwise.

If the AUC is not defined (i. e. if roc was called with AUC=FALSE), returns NULL.

See Also

[auc](#)

Examples

```
data(aSAH)

# Full AUC
roc1 <- roc(aSAH$outcome, aSAH$s100b)
has.partial.auc(roc1)
has.partial.auc(auc(roc1))
has.partial.auc(smooth(roc1))

# Partial AUC
roc2 <- roc(aSAH$outcome, aSAH$s100b, partial.auc = c(1, 0.9))
has.partial.auc(roc2)
has.partial.auc(smooth(roc2))

# No AUC
```

```
roc3 <- roc(aSAH$outcome, aSAH$s100b, auc = FALSE)
has.partial.auc(roc3)
```

lines.roc

Add a ROC line to a ROC plot

Description

This convenience function adds a ROC line to a ROC curve.

Usage

```
## S3 method for class 'roc'
lines(x, ...)
## S3 method for class 'smooth.roc'
lines(x, ...)
## S3 method for class 'roc'
lines.roc(x, lwd=2, ...)
## S3 method for class 'formula'
lines.roc(x, data, ...)
## Default S3 method:
lines.roc(x, predictor, ...)
## S3 method for class 'smooth.roc'
lines.roc(x, ...)
```

Arguments

x	a roc object from the roc function (for plot.roc.roc), a formula (for plot.roc.formula) or a response vector (for plot.roc.default).
predictor, data	arguments for the roc function.
lwd	line width (see par).
...	graphical parameters for lines , and especially type (see plot.default) and arguments for par such as col (color), lty (line type) or line characteristics lend, ljoin and lmitre.

Value

This function returns a list of class “roc” invisibly. See [roc](#) for more details.

See Also

[roc](#), [plot.roc](#)

Examples

```
data(aSAH)

rocobj <- plot.roc(aSAH$outcome, aSAH$s100b, type="n")
lines(rocobj, type="b", pch=21, col="blue", bg="grey")

# Without using 'lines':
rocobj <- plot.roc(aSAH$outcome, aSAH$s100b, type="b", pch=21, col="blue", bg="grey")
```

multiclass.roc

Multi-class AUC

Description

This function builds multiple ROC curve to compute the multi-class AUC as defined by Hand and Till.

Usage

```
multiclass.roc(...)
## S3 method for class 'formula'
multiclass.roc(formula, data, ...)
## Default S3 method:
multiclass.roc(response, predictor,
  levels=base::levels(as.factor(response)),
  percent=FALSE, ...)
```

Arguments

response	a factor, numeric or character vector of responses, typically encoded with 0 (controls) and 1 (cases), as in roc .
predictor	a numeric vector, containing the value of each observation, as in roc .
formula	a formula of the type response~predictor.
data	a matrix or data.frame containing the variables in the formula. See model.frame for more details.
levels	the value of the response for controls and cases respectively. In contrast with levels argument to roc , all the levels are used and combined to compute the multiclass AUC.
percent	if the sensitivities, specificities and AUC must be given in percent (TRUE) or in fraction (FALSE, default).
...	further arguments passed to roc .

Details

This function performs multiclass AUC as defined by Hand and Till (2001). A multiclass AUC is a mean of [auc](#) and cannot be plotted.

This function has been much less tested than the rest of the package and is more subject to bugs. Please report them if you find one. Confidence intervals and comparison tests are not implemented yet.

Value

A list of class “multiclass.roc” with the following fields:

<code>auc</code>	if called with <code>auc=TRUE</code> , a numeric of class “auc” as defined in auc . Note that this is not the standard AUC but the multi-class AUC as defined by Hand and Till.
<code>ci</code>	if called with <code>ci=TRUE</code> , a numeric of class “ci” as defined in ci .
<code>response</code>	the response vector as passed in argument. If NA values were removed, a <code>na.action</code> attribute similar to na.omit stores the row numbers.
<code>predictor</code>	the predictor vector as passed in argument. If NA values were removed, a <code>na.action</code> attribute similar to na.omit stores the row numbers.
<code>levels</code>	the levels of the response as defined in argument.
<code>percent</code>	if the sensitivities, specificities and AUC are reported in percent, as defined in argument.
<code>call</code>	how the function was called. See match.call for more details.

Warnings

If response is an ordered factor and one of the levels specified in `levels` is missing, a warning is issued and the level is ignored.

References

David J. Hand and Robert J. Till (2001). A Simple Generalisation of the Area Under the ROC Curve for Multiple Class Classification Problems. *Machine Learning* **45**(2), p. 171–186. DOI: [10.1023/A:1010920819831](https://doi.org/10.1023/A:1010920819831).

See Also

[auc](#)

Examples

```
data(aSAH)

# Basic example
multiclass.roc(aSAH$gos6, aSAH$s100b)
# Produces an innocuous warning because one level has no observation

# Select only 3 of the aSAH$gos6 levels:
```

```

multiclass.roc(aSAH$gos6, aSAH$s100b, levels=c(3, 4, 5))

# Give the result in percent
multiclass.roc(aSAH$gos6, aSAH$s100b, percent=TRUE)

```

plot.ci

Plot confidence intervals

Description

This function adds confidence intervals to a ROC curve plot, either as bars or as a confidence shape.

Usage

```

## S3 method for class 'ci.thresholds'
plot(x, length=.01*ifelse(attr(x,
  "roc")$percent, 100, 1), col=par("fg"), ...)
## S3 method for class 'ci.sp'
plot(x, type=c("bars", "shape"), length=.01*ifelse(attr(x,
  "roc")$percent, 100, 1), col=ifelse(type=="bars", par("fg"),
  "gainsboro"), no.roc=FALSE, ...)
## S3 method for class 'ci.se'
plot(x, type=c("bars", "shape"), length=.01*ifelse(attr(x,
  "roc")$percent, 100, 1), col=ifelse(type=="bars", par("fg"),
  "gainsboro"), no.roc=FALSE, ...)

```

Arguments

x	a confidence interval object from the functions ci.thresholds , ci.se or ci.sp .
type	type of plot, "bars" or "shape". Can be shortened to "b" or "s". "shape" is only available for ci.se and ci.sp , not for ci.thresholds .
length	the length (as plot coordinates) of the bar ticks. Only if type="bars".
no.roc	if FALSE, the ROC line is re-added over the shape. Otherwise if TRUE, only the shape is plotted. Ignored if type="bars"
col	color of the bars or shape.
...	further arguments for segments (if type="bars") or polygon (if type="shape").

Details

This function adds confidence intervals to a ROC curve plot, either as bars or as a confidence shape, depending on the state of the type argument. The shape is plotted over the ROC curve, so that the curve is re-plotted unless no.roc=TRUE.

Graphical functions are called with [suppressWarnings](#).

Value

This function returns the confidence interval object invisibly.

Warnings

With `type="shape"`, the warning “Low definition shape” is issued when the shape is defined by less than 15 confidence intervals. In such a case, the shape is not well defined and the ROC curve could pass outside the shape. To get a better shape, increase the number of intervals, for example with:

```
plot(ci.sp(rocobj, sensitivities=seq(0, 1, .01)), type="shape")
```

References

Xavier Robin, Natacha Turck, Alexandre Hainard, *et al.* (2011) “pROC: an open-source package for R and S+ to analyze and compare ROC curves”. *BMC Bioinformatics*, **7**, 77. DOI: [10.1186/1471-2105-12-77](https://doi.org/10.1186/1471-2105-12-77).

See Also

[plot.roc](#), [ci.thresholds](#), [ci.sp](#), [ci.se](#)

Examples

```
data(aSAH)
## Not run:
# Start a ROC plot
rocobj <- plot.roc(aSAH$outcome, aSAH$s100b)
plot(rocobj)
# Thresholds
ci.thresholds.obj <- ci.thresholds(rocobj)
plot(ci.thresholds.obj)
# Specificities
plot(rocobj) # restart a new plot
ci.sp.obj <- ci.sp(rocobj, boot.n=500)
plot(ci.sp.obj)
# Sensitivities
plot(rocobj) # restart a new plot
ci.se.obj <- ci(rocobj, of="se", boot.n=500)
plot(ci.se.obj)

# Plotting a shape. We need more
ci.sp.obj <- ci.sp(rocobj, sensitivities=seq(0, 1, .01), boot.n=100)
plot(rocobj) # restart a new plot
plot(ci.sp.obj, type="shape", col="blue")

# Direct syntax (response, predictor):
plot.roc(aSAH$outcome, aSAH$s100b,
        ci=TRUE, of="thresholds")

## End(Not run)
```

plot.roc

*Plot a ROC curve***Description**

This function plots a ROC curve. It can accept many arguments to tweak the appearance of the plot. Two syntaxes are possible: one object of class “`roc`”, or either two vectors (response, predictor) or a formula (response~predictor) as in the `roc` function.

Usage

```
## S3 method for class 'roc'
plot(x, ...)
## S3 method for class 'smooth.roc'
plot(x, ...)
## S3 method for class 'roc'
plot.roc(x, add=FALSE, reuse.auc=TRUE,
axes=TRUE, legacy.axes=FALSE,
# Generic arguments for par:
xlim=if(x$percent){c(100, 0)} else{c(1, 0)},
ylim=if(x$percent){c(0, 100)} else{c(0, 1)},
xlab=ifelse(x$percent, ifelse(legacy.axes, "100 - Specificity (%)", "Specificity (%)", ifelse(legacy.
ylab=ifelse(x$percent, "Sensitivity (%)", "Sensitivity"),
asp=1,
mar=c(4, 4, 2, 2)+.1,
mgp=c(2.5, 1, 0),
# col, lty and lwd for the ROC line only
col=par("col"),
lty=par("lty"),
lwd=2,
type="l",
# Identity line
identity=!add,
identity.col="darkgrey",
identity.lty=1,
identity.lwd=1,
# Print the thresholds on the plot
print.thres=FALSE,
print.thres.pch=20,
print.thres.adj=c(-.05,1.25),
print.thres.col="black",
print.thres.pattern=ifelse(x$percent, "%.1f (%.1f%%, %.1f%%)", "%.3f (%.3f, %.3f)"),
print.thres.cex=par("cex"),
print.thres.pattern.cex=print.thres.cex,
print.thres.best.method=NULL,
print.thres.best.weights=c(1, 0.5),
# Print the AUC on the plot
```

```

print.auc=FALSE,
print.auc.pattern=NULL,
print.auc.x=ifelse(x$percent, 50, .5),
print.auc.y=ifelse(x$percent, 50, .5),
print.auc.adj=c(0,1),
print.auc.col=col,
print.auc.cex=par("cex"),
# Grid
grid=FALSE,
grid.v={if(is.logical(grid) && grid[1]==TRUE){seq(0, 1, 0.1) * ifelse(x$percent, 100, 1)} else if(is.nu
grid.h={if (length(grid) == 1) {grid.v} else if (is.logical(grid) && grid[2]==TRUE){seq(0, 1, 0.1) * ife
grid.lty=3,
grid.lwd=1,
grid.col="#DDDDDD",
# Polygon for the AUC
auc.polygon=FALSE,
auc.polygon.col="gainsboro",
auc.polygon.lty=par("lty"),
auc.polygon.density=NULL,
auc.polygon.angle=45,
auc.polygon.border=NULL,
# Polygon for the maximal AUC possible
max.auc.polygon=FALSE,
max.auc.polygon.col="#EEEEEE",
max.auc.polygon.lty=par("lty"),
max.auc.polygon.density=NULL,
max.auc.polygon.angle=45,
max.auc.polygon.border=NULL,
# Confidence interval
ci=!is.null(x$ci),
ci.type=c("bars", "shape", "no"),
ci.col=ifelse(ci.type=="bars", par("fg"), "gainsboro"),
...)
## S3 method for class 'formula'
plot.roc(x, data, ...)
## Default S3 method:
plot.roc(x, predictor, ...)
## S3 method for class 'smooth.roc'
plot.roc(x, ...)

```

Arguments

<code>x</code>	a roc object from the roc function (for <code>plot.roc.roc</code>), a formula (for <code>plot.roc.formula</code>) or a response vector (for <code>plot.roc.default</code>).
<code>predictor, data</code>	arguments for the roc function.
<code>add</code>	if TRUE, the ROC curve will be added to an existing plot. If FALSE (default),

	a new plot will be created.
<code>reuse.auc</code>	if TRUE (default) and the “roc” object contains an “auc” field, re-use these specifications for the plot (specifically <code>print.auc</code> , <code>auc.polygon</code> and <code>max.auc.polygon</code> arguments). See details.
<code>axes</code>	a logical indicating if the plot axes must be drawn.
<code>legacy.axes</code>	a logical indicating if the specificity axis (x axis) must be plotted as decreasing “specificity” (FALSE, the default) or increasing “1 - specificity” (TRUE) as in most legacy software. This affects only the axis, not the plot coordinates.
<code>xlim, ylim, xlab, ylab, asp, mar, mgp</code>	Generic arguments for the plot. See plot and plot.window for more details. Only used if <code>add=FALSE</code> .
<code>col, lty, lwd</code>	color, line type and line width for the ROC curve. See par for more details.
<code>type</code>	type of plotting as in plot .
<code>identity</code>	logical: whether or not the identity line (no discrimination line) must be displayed. Default: only on new plots.
<code>identity.col, identity.lty, identity.lwd</code>	color, line type and line width for the identity line. Used only if <code>identity=TRUE</code> . See par for more details.
<code>print.thres</code>	Should a selected set of thresholds be displayed on the ROC curve? FALSE, NULL or “no”: no threshold is displayed. TRUE or “best”: the threshold with the highest sum sensitivity + specificity is plotted (this might be more than one threshold). “all”: all the points of the ROC curve. “local maximas”: all the local maximas. Numeric vector: direct definition of the thresholds to display. Note that on a smoothed ROC curve, only “best” is supported.
<code>print.thres.pch, print.thres.adj, print.thres.col, print.thres.cex</code>	the plotting character (pch), text string adjustment (adj), color (col) and character expansion factor (cex) parameters for the printing of the thresholds. See points and par for more details.
<code>print.thres.pattern</code>	the text pattern for the thresholds, as a sprintf format. Three numerics are passed to <code>sprintf</code> : threshold, specificity, sensitivity.
<code>print.thres.pattern.cex</code>	the character expansion factor (cex) for the threshold text pattern. See par for more details.
<code>print.thres.best.method, print.thres.best.weights</code>	if <code>print.thres="best"</code> or <code>print.thres=TRUE</code> , what method must be used to determine which threshold is the best. See argument <code>best.method</code> and <code>best.weights</code> to coords for more details.
<code>print.auc</code>	boolean. Should the numeric value of AUC be printed on the plot?
<code>print.auc.pattern</code>	the text pattern for the AUC, as a sprintf format. If NULL, a reasonable value is computed that takes partial AUC, CI and percent into account. If the CI of the AUC was computed, three numerics are passed to <code>sprintf</code> : AUC, lower CI bound, higher CI bound. Otherwise, only AUC is passed.

`print.auc.x`, `print.auc.y`
 x and y position for the printing of the AUC.

`print.auc.adj`, `print.auc.cex`, `print.auc.col`
 the text adjustment, character expansion factor and color for the printing of the AUC. See [par](#) for more details.

`grid`
 boolean or numeric vector of length 1 or 2. Should a background grid be added to the plot? Numeric: show a grid with the specified interval between each line; Logical: show the grid or not. Length 1: same values are taken for horizontal and vertical lines. Length 2: grid value for vertical (`grid[1]`) and horizontal (`grid[2]`). Note that these values are used to compute `grid.v` and `grid.h`. Therefore if you specify a `grid.h` and `grid.v`, it will be ignored.

`grid.v`, `grid.h`
 numeric. The x and y values at which a vertical or horizontal line (respectively) must be drawn. NULL if no line must be added.

`grid.lty`, `grid.lwd`, `grid.col`
 the line type (`lty`), line width (`lwd`) and color (`col`) of the lines of the grid. See [par](#) for more details. Note that you can pass vectors of length 2, in which case it specifies the vertical (1) and horizontal (2) lines.

`auc.polygon`
 boolean. Whether or not to display the area as a polygon.

`auc.polygon.col`, `auc.polygon.lty`, `auc.polygon.density`, `auc.polygon.angle`, `auc.polygon.border`
 color (`col`), line type (`lty`), density, angle and border for the AUC polygon. See [polygon](#) and [par](#) for more details.

`max.auc.polygon`
 boolean. Whether or not to display the maximal possible area as a polygon.

`max.auc.polygon.col`, `max.auc.polygon.lty`, `max.auc.polygon.density`, `max.auc.polygon.angle`, `max.auc.polygon.border`
 color (`col`), line type (`lty`), density, angle and border for the maximum AUC polygon. See [polygon](#) and [par](#) for more details.

`ci`
 boolean. Should we plot the confidence intervals?

`ci.type`, `ci.col`
 type and col arguments for [plot.ci](#). The special value “no” disables the plotting of confidence intervals.

...
 further arguments passed to or from other methods, especially arguments for [roc](#) and `plot.roc.roc` when calling `plot.roc.default` or `plot.roc.formula`. Note that the `plot` argument for [roc](#) is not allowed. Arguments for [auc](#) and graphical functions [plot](#), [abline](#), [polygon](#), [points](#), [text](#) and [plot.ci](#) if applicable.

Details

This function is typically called from [roc](#) when `plot=TRUE` (not by default). `plot.roc.formula` and `plot.roc.default` are convenience methods that build the ROC curve (with the [roc](#) function) before calling `plot.roc.roc`. You can pass them arguments for both [roc](#) and `plot.roc.roc`. Simply use `plot.roc` that will dispatch to the correct method.

The plotting is done in the following order:

1. A new plot is created if `add=FALSE`.
2. The grid is added if `grid.v` and `grid.h` are not NULL.

3. The maximal AUC polygon is added if `max.auc.polygon=TRUE`.
4. The CI shape is added if `ci=TRUE`, `ci.type="shape"` and `x$ci` isn't a "ci.auc".
5. The AUC polygon is added if `auc.polygon=TRUE`.
6. The identity line if `identity=TRUE`.
7. The actual ROC line is added.
8. The CI bars are added if `ci=TRUE`, `ci.type="bars"` and `x$ci` isn't a "ci.auc".
9. The selected thresholds are printed if `print.thres` is `TRUE` or numeric.
10. The AUC is printed if `print.auc=TRUE`.

Graphical functions are called with [suppressWarnings](#).

Value

This function returns a list of class "roc" invisibly. See [roc](#) for more details.

AUC specification

For `print.auc`, `auc.polygon` and `max.auc.polygon` arguments, an AUC specification is required. By default, the total AUC is plotted, but you may want a partial AUCs. The specification is defined by:

1. the "auc" field in the "roc" object if `reuse.auc` is set to `TRUE` (default). It is naturally inherited from any call to [roc](#) and fits most cases.
2. passing the specification to [auc](#) with `...` (arguments `partial.auc`, `partial.auc.correct` and `partial.auc.focus`). In this case, you must ensure either that the [roc](#) object do not contain an `auc` field (if you called [roc](#) with `auc=FALSE`), or set `reuse.auc=FALSE`.

If `reuse.auc=FALSE` the [auc](#) function will always be called with `...` to determine the specification, even if the "roc" object do contain an `auc` field.

As well if the "roc" object do not contain an `auc` field, the [auc](#) function will always be called with `...` to determine the specification.

Warning: if the roc object passed to `plot.roc` contains an `auc` field and `reuse.auc=TRUE`, [auc](#) is not called and arguments such as `partial.auc` are silently ignored.

References

Xavier Robin, Natacha Turck, Alexandre Hainard, *et al.* (2011) "pROC: an open-source package for R and S+ to analyze and compare ROC curves". *BMC Bioinformatics*, 7, 77. DOI: [10.1186/1471-2105-12-77](https://doi.org/10.1186/1471-2105-12-77).

See Also

[roc](#), [auc](#), [ci](#)

Examples

```
data(aSAH)

# Syntax (response, predictor):
plot.roc(aSAH$outcome, aSAH$s100b)

# With a roc object:
rocobj <- roc(aSAH$outcome, aSAH$s100b)
# identical:
plot(rocobj)
plot.roc(rocobj)

# Add a smoothed ROC:
plot.roc(smooth(rocobj), add=TRUE, col="blue")
legend("bottomright", legend=c("Empirical", "Smoothed"),
      col=c(par("fg"), "blue"), lwd=2)

# With more options:
plot(rocobj, print.auc=TRUE, auc.polygon=TRUE, grid=c(0.1, 0.2),
      grid.col=c("green", "red"), max.auc.polygon=TRUE,
      auc.polygon.col="blue", print.thres=TRUE)

# To plot a different partial AUC, we need to ignore the existing value
# with reuse.auc=FALSE:
plot(rocobj, print.auc=TRUE, auc.polygon=TRUE, partial.auc=c(1, 0.8),
      partial.auc.focus="se", grid=c(0.1, 0.2), grid.col=c("green", "red"),
      max.auc.polygon=TRUE, auc.polygon.col="blue", print.thres=TRUE,
      reuse.auc=FALSE)

# Add a line to the previous plot:
plot.roc(aSAH$outcome, aSAH$wfns, add=TRUE)

# Alternatively, you can get the plot directly from roc():
roc(aSAH$outcome, aSAH$s100b, plot=TRUE)
```

print

Print a ROC curve object

Description

This function prints a ROC curve, AUC or CI object and return it invisibly.

Usage

```
## S3 method for class 'roc'
print(x, digits=max(3, getOption("digits") - 3), call=TRUE, ...)
## S3 method for class 'multiclass.roc'
print(x, digits=max(3, getOption("digits") -
  3), call=TRUE, ...)
```

```
## S3 method for class 'smooth.roc'
print(x, digits=max(3, getOption("digits") - 3),
      call=TRUE, ...)
## S3 method for class 'auc'
print(x, digits=max(3, getOption("digits") - 3), ...)
## S3 method for class 'multiclass.auc'
print(x, digits=max(3, getOption("digits") - 3), ...)
## S3 method for class 'ci.auc'
print(x, digits=max(3, getOption("digits") - 3), ...)
## S3 method for class 'ci.thresholds'
print(x, digits=max(3, getOption("digits") - 3), ...)
## S3 method for class 'ci.se'
print(x, digits=max(3, getOption("digits") - 3), ...)
## S3 method for class 'ci.sp'
print(x, digits=max(3, getOption("digits") - 3), ...)
```

Arguments

<code>x</code>	a roc, auc or ci object, from the roc , auc or ci functions respectively.
<code>call</code>	if the call is printed.
<code>digits</code>	the number of significant figures to print. See signif for more details.
<code>...</code>	further arguments passed to or from other methods. In particular, <code>print.roc</code> calls <code>print.auc</code> and the <code>print.ci</code> variants internally, and a <code>digits</code> argument is propagated. Not used in <code>print.auc</code> and <code>print.ci</code> variants.

Value

These functions return the object they were passed invisibly.

See Also

[roc](#), [auc](#), [ci](#), [coords](#)

Examples

```
data(aSAH)

# Print a roc object:
rocobj <- roc(aSAH$outcome, aSAH$s100b)
print(rocobj)

# Print a smoothed roc object
print(smooth(rocobj))

# implicit printing
roc(aSAH$outcome, aSAH$s100b)

# Print an auc and a ci object, from the ROC object or calling
# the dedicated function:
print(rocobj$auc)
```

```
print(ci(rocobj))
```

 roc

Build a ROC curve

Description

This is the main function of the pROC package. It builds a ROC curve and returns a “roc” object, a list of class “roc”. This object can be printed, plotted, or passed to the functions [auc](#), [ci](#), [smooth.roc](#) and [coords](#). Additionally, two roc objects can be compared with [roc.test](#).

Usage

```
roc(...)
## S3 method for class 'formula'
roc(formula, data, ...)
## Default S3 method:
roc(response, predictor, controls, cases,
     density.controls, density.cases,
     levels=base::levels(as.factor(response)), percent=FALSE, na.rm=TRUE,
     direction=c("auto", "<", ">"), smooth=FALSE, auc=TRUE, ci=FALSE,
     plot=FALSE, smooth.method="binormal", ci.method=NULL, density=NULL, ...)
```

Arguments

response	a factor, numeric or character vector of responses, typically encoded with 0 (controls) and 1 (cases). The object. Only two classes can be used in a ROC curve. If the vector contains more than two unique values, or if their order could be ambiguous, use levels to specify which values must be used as control and case value.
predictor	a numeric vector, containing the value of each observation. An ordered factor is coerced to a numeric.
controls, cases	instead of response, predictor, the data can be supplied as two vectors containing the predictor values for control and case observations.
density.controls, density.cases	a smoothed ROC curve can be built directly from two densities on identical x points, as in smooth.roc .
formula	a formula of the type response~predictor.
data	a matrix or data.frame containing the variables in the formula. See model.frame for more details.
levels	the value of the response for controls and cases respectively. By default, the first two values of levels(as.factor(response)) are taken, and the remaining levels are ignored. It usually captures two-class factor data correctly, but will frequently fail for other data types (response factor with more than 2 levels, or

	for example if your response is coded “controls” and “cases”, the levels will be inverted) and must then be precised here. If your data is coded as 0 and 1 with 0 being the controls, you can safely omit this argument.
percent	if the sensitivities, specificities and AUC must be given in percent (TRUE) or in fraction (FALSE, default).
na.rm	if TRUE, the NA values will be removed.
direction	in which direction to make the comparison? “auto” (default): automatically define in which group the median is higher and take the direction accordingly. “>”: if the predictor values for the control group are higher than the values of the case group. “<”: if the predictor values for the control group are higher or equal than the values of the case group.
smooth	if TRUE, the ROC curve is passed to smooth to be smoothed.
auc	compute the area under the curve (AUC)? If TRUE (default), additional arguments can be passed to auc .
ci	compute the confidence interval (CI)? If TRUE (default), additional arguments can be passed to ci .
plot	plot the ROC curve? If TRUE, additional arguments can be passed to plot.roc .
smooth.method, ci.method	in roc.formula and roc.default , the method arguments to smooth.roc (if smooth=TRUE) and of=”auc”) must be passed as smooth.method and ci.method to avoid confusions.
density	density argument passed to smooth.roc .
...	further arguments passed to or from other methods, and especially: • auc: partial.auc, partial.auc.focus, partial.auc.correct. • ci: of, conf.level, boot.n, boot.stratified, progress • ci.auc: reuse.auc, method • ci.thresholds: thresholds • ci.sp: sensitivities • ci.se: specificities • plot.roc: add, col and most other arguments to the plot.roc function. See plot.roc directly for more details. • smooth: method, n, and all other arguments. See smooth for more details.

Details

This function’s main job is to build a ROC object. See the “Value” section to this page for more details. Before returning, it will call (in this order) the [smooth.roc](#), [auc](#), [ci](#) and [plot.roc](#) functions if smooth auc, ci and plot.roc (respectively) arguments are set to TRUE. By default, only auc is called.

Data can be provided as response, predictor, where the predictor is the numeric (or ordered) level of the evaluated signal, and the response encodes the observation class (control or case). The level argument specifies which response level must be taken as controls (first value of level) or cases (second). It can safely be ignored when the response is encoded as 0 and 1, but it will frequently fail otherwise. By default, the first two values of `levels(as.factor(response))` are

taken, and the remaining levels are ignored. This means that if your response is coded “control” and “case”, the levels will be inverted.

In some cases, it is more convenient to pass the data as `controls`, `cases`, but both arguments are ignored if `response`, `predictor` was specified to non-NULL values. It is also possible to pass density data with `density.controls`, `density.cases`, which will result in a smoothed ROC curve even if `smooth=FALSE`, but are ignored if `response`, `predictor` or `controls`, `cases` are provided.

Specifications for `auc`, `ci` and `plot.roc` are not kept if `auc`, `ci` or `plot` are set to `FALSE`. Especially, in the following case:

```
myRoc <- roc(..., auc.polygon=TRUE, grid=TRUE, plot=FALSE)
plot(myRoc)
```

the plot will not have the AUC polygon nor the grid. Similarly, when comparing “roc” objects, the following is not possible:

```
roc1 <- roc(..., partial.auc=c(1, 0.8), auc=FALSE)
roc2 <- roc(..., partial.auc=c(1, 0.8), auc=FALSE)
roc.test(roc1, roc2)
```

This will produce a test on the full AUC, not the partial AUC. To make a comparison on the partial AUC, you must repeat the specifications when calling `roc.test`:

```
roc.test(roc1, roc2, partial.auc=c(1, 0.8))
```

Note that if `roc` was called with `auc=TRUE`, the latter syntax will not allow redefining the AUC specifications. You must use `reuse.auc=FALSE` for that.

Value

If the data contained any NA value and `na.rm=FALSE`, NA is returned. Otherwise, if `smooth=FALSE`, a list of class “roc” with the following fields:

<code>auc</code>	if called with <code>auc=TRUE</code> , a numeric of class “auc” as defined in auc .
<code>ci</code>	if called with <code>ci=TRUE</code> , a numeric of class “ci” as defined in ci .
<code>response</code>	the response vector as passed in argument. If NA values were removed, a <code>na.action</code> attribute similar to na.omit stores the row numbers.
<code>predictor</code>	the predictor vector converted to numeric as used to build the ROC curve. If NA values were removed, a <code>na.action</code> attribute similar to na.omit stores the row numbers.
<code>original.predictor</code>	the predictor vector as passed in argument.
<code>levels</code>	the levels of the response as defined in argument.

controls	the predictor values for the control observations.
cases	the predictor values for the cases.
percent	if the sensitivities, specificities and AUC are reported in percent, as defined in argument.
direction	the direction of the comparison, as defined in argument.
sensitivities	the sensitivities defining the ROC curve.
specificities	the specificities defining the ROC curve.
thresholds	the thresholds at which the sensitivities and specificities were computed.
call	how the function was called. See match.call for more details.

If smooth=TRUE a list of class “smooth.roc” as returned by [smooth](#), with or without additional elements auc and ci (according to the call).

Errors

If no control or case observation exist for the given levels of response, no ROC curve can be built and an error is triggered with message “No control observation” or “No case observation”.

If the predictor is not a numeric or ordered, as defined by [as.numeric](#) or [as.ordered](#), the message “Predictor must be numeric or ordered” is returned.

The message “No valid data provided” is issued when the data wasn’t properly passed. Remember you need both response and predictor of the same (not null) length, or bot controls and cases. Combinations such as predictor and cases are not valid and will trigger this error.

References

Tom Fawcett (2006) “An introduction to ROC analysis”. *Pattern Recognition Letters* **27**, 861–874. DOI: [10.1016/j.patrec.2005.10.010](#).

Xavier Robin, Natacha Turck, Alexandre Hainard, *et al.* (2011) “pROC: an open-source package for R and S+ to analyze and compare ROC curves”. *BMC Bioinformatics*, **7**, 77. DOI: [10.1186/1471-2105-12-77](#).

See Also

[auc](#), [ci](#), [plot.roc](#), [print.roc](#), [roc.test](#)

Examples

```
data(aSAH)

# Basic example
roc(aSAH$outcome, aSAH$s100b,
    levels=c("Good", "Poor"))
# As levels aSAH$outcome == c("Good", "Poor"),
# this is equivalent to:
roc(aSAH$outcome, aSAH$s100b)
# In some cases, ignoring levels could lead to unexpected results
# Equivalent syntaxes:
```

```

roc(outcome ~ s100b, aSAH)
roc(aSAH$outcome ~ aSAH$s100b)
with(aSAH, roc(outcome, s100b))
with(aSAH, roc(outcome ~ s100b))

# With a formula:
roc(outcome ~ s100b, data=aSAH)

# With controls/cases
roc(controls=aSAH$s100b[aSAH$outcome=="Good"], cases=aSAH$s100b[aSAH$outcome=="Poor"])

# Inverted the levels: "Poor" are now controls and "Good" cases:
roc(aSAH$outcome, aSAH$s100b,
    levels=c("Poor", "Good"))

# The result was exactly the same because of direction="auto".
# The following will give an AUC < 0.5:
roc(aSAH$outcome, aSAH$s100b,
    levels=c("Poor", "Good"), direction="<")

# If we prefer counting in percent:
roc(aSAH$outcome, aSAH$s100b, percent=TRUE)

# Plot and CI (see plot.roc and ci for more options):
roc(aSAH$outcome, aSAH$s100b,
    percent=TRUE, plot=TRUE, ci=TRUE)

# Smoothed ROC curve
roc(aSAH$outcome, aSAH$s100b, smooth=TRUE)
# this is not identical to
smooth(roc(aSAH$outcome, aSAH$s100b))
# because in the latter case, the returned object contains no AUC

```

roc.test

Compare the AUC of two ROC curves

Description

This function compares the AUC or partial AUC of two correlated (or paired) or uncorrelated (unpaired) ROC curves. Several syntaxes are available: two object of class roc (which can be AUC or smoothed ROC), or either three vectors (response, predictor1, predictor2) or a response vector and a matrix or data.frame with two columns (predictors).

Usage

```

# roc.test(...)
## S3 method for class 'roc'
roc.test(roc1, roc2, method=c("delong", "bootstrap",
"venkatraman", "sensitivity", "specificity"), sensitivity = NULL,
specificity = NULL, alternative = c("two.sided", "less", "greater"),

```

```

paired=NULL, reuse.auc=TRUE, boot.n=2000, boot.stratified=TRUE,
ties.method="first", progress=getOption("pROCProgress")$name, ...)
## S3 method for class 'auc'
roc.test(roc1, roc2, ...)
## S3 method for class 'smooth.roc'
roc.test(roc1, roc2, ...)
## S3 method for class 'formula'
roc.test(formula, data, ...)
## Default S3 method:
roc.test(response, predictor1, predictor2=NULL,
na.rm=TRUE, method=NULL, ...)

```

Arguments

roc1, roc2	the two ROC curves to compare. Either “roc”, “auc” or “smooth.roc” objects (types can be mixed).
response	a vector or factor, as for the roc function.
predictor1	a numeric or ordered vector as for the roc function, or a matrix or data.frame with predictors two columns.
predictor2	only if predictor1 was a vector, the second predictor as a numeric vector.
formula	a formula of the type response~predictor1+predictor2.
data	a matrix or data.frame containing the variables in the formula. See model.frame for more details.
na.rm	if TRUE, the observations with NA values will be removed.
method	the method to use, either “delong”, “bootstrap” or “venkatraman”. The first letter is sufficient. If omitted, the appropriate method is selected as explained in details.
sensitivity, specificity	if method=“sensitivity” or method=“specificity”, the respective level where the test must be assessed as a numeric of length 1.
alternative	specifies the alternative hypothesis. Either of “two.sided”, “less” or “greater”. The first letter is sufficient. Default: “two.sided”. Only “two.sided” is available with method=“venkatraman”.
paired	a logical indicating whether you want a paired roc.test. If NULL, the paired status will be auto-detected by are.paired . If TRUE but the paired status cannot be assessed by are.paired will produce an error.
reuse.auc	if TRUE (default) and the “roc” objects contain an “auc” field, re-use these specifications for the test. See the <i>AUC specification</i> section for more details.
boot.n	for method=“bootstrap” and method=“venkatraman” only: the number of bootstrap replicates or permutations. Default: 2000.
boot.stratified	for method=“bootstrap” only: should the bootstrap be stratified (same number of cases/controls in each replicate than in the original sample) or not. Ignored with method=“venkatraman”. Default: TRUE.

<code>ties.method</code>	for <code>method="venkatraman"</code> only: argument for rank specifying how ties are handled. Defaults to "first" as described in the paper.
<code>progress</code>	the name of progress bar to display. Typically "none", "win", "tk" or "text" (see the name argument to create_progress_bar for more information), but a list as returned by create_progress_bar is also accepted. See also the "Progress bars" section of this package's documentation .
<code>...</code>	further arguments passed to or from other methods, especially arguments for roc and <code>roc.test.roc</code> when calling <code>roc.test.default</code> or <code>roc.test.formula</code> . Arguments for auc , and txtProgressBar (only <code>char</code> and <code>style</code>) if applicable.

Details

This function compares two ROC curves. It is typically called with the two [roc](#) objects to compare. `roc.test.default` is provided as a convenience method and creates two [roc](#) objects before calling `roc.test.roc`.

Three methods are available: "delong", "bootstrap" and "venkatraman" (see "Computational details" section below). "delong" and "bootstrap" are tests over the AUC whereas "venkatraman" compares the the ROC curves themselves.

Default is to use "delong" method except for comparison of partial AUC, smoothed curves and curves with different direction, where bootstrap is used. Using "delong" for partial AUC and smoothed ROCs is not supported in pROC (a warning is produced and "bootstrap" is employed instead). It is spurious to use "delong" for [roc](#) with different direction (a warning is issued but the spurious comparison is enforced). "venkatraman"'s test cannot be employed to compare smoothed ROC curves. Additionally, partial AUC specifications are ignored (with a warning), and comparison of ROC curves with different direction should be used with care (a warning is produced as well).

If `alternative="two.sided"`, a two-sided test for difference in AUC is performed. If `alternative="less"`, the alternative is that the AUC of `roc1` is smaller than the AUC of `roc2`. For `method="venkatraman"`, only "two.sided" test is available.

If the `paired` argument is not provided, the [are.paired](#) function is employed to detect the paired status of the ROC curves. It will test if the original response is identical between the two ROC curves (this is always the case if the call is made with `roc.test.default`). This detection is unlikely to raise false positives, but this possibility cannot be excluded entirely. It would require equal sample sizes and response values and order in both ROC curves. If it happens to you, use `paired=FALSE`. If you know the ROC curves are paired you can pass `paired=TRUE`. However this is useless as it will be tested anyway.

For [smoothed ROC curves](#), smoothing is performed again at each bootstrap replicate with the parameters originally provided. If a density smoothing was performed with user-provided `density.cases` or `density.controls` the bootstrap cannot be performed and an error is issued.

Value

A list of class "htest" with following content:

<code>p.value</code>	the p-value of the test.
<code>statistic</code>	the value of the Z (<code>method="delong"</code>) or D (<code>method="bootstrap"</code>) statistics.
<code>alternative</code>	the alternative hypothesis.

method	the character string “DeLong’s test for two correlated ROC curves” (if method=“deLong”) or “Bootstrap test for two correlated ROC curves” (if method=“bootstrap”).
null.value	the expected value of the statistic under the null hypothesis, that is 0.
estimate	the AUC in the two ROC curves.
data.name	the names of the data that was used.
parameter	for method=“bootstrap” only: the values of the boot.n and boot.stratified arguments.

AUC specification

The comparison of the AUC of the ROC curves needs a specification of the AUC. The specification is defined by:

1. the “auc” field in the “roc” objects if reuse.auc is set to TRUE (default)
2. passing the specification to `auc` with ... (arguments `partial.auc`, `partial.auc.correct` and `partial.auc.focus`). In this case, you must ensure either that the `roc` object do not contain an `auc` field (if you called `roc` with `auc=FALSE`), or set `reuse.auc=FALSE`.

If `reuse.auc=FALSE` the `auc` function will always be called with ... to determine the specification, even if the “roc” objects do contain an `auc` field.

As well if the “roc” objects do not contain an `auc` field, the `auc` function will always be called with ... to determine the specification.

The AUC specification is ignored in the Venkatraman test.

Warning: if the roc object passed to `roc.test` contains an `auc` field and `reuse.auc=TRUE`, `auc` is not called and arguments such as `partial.auc` are silently ignored.

Computation details

With method=“bootstrap”, the processing is done as follow:

1. `boot.n` bootstrap replicates are drawn from the data. If `boot.stratified` is `TRUE`, each replicate contains exactly the same number of controls and cases than the original sample, otherwise if `FALSE` the numbers can vary.
2. for each bootstrap replicate, the AUC of the two ROC curves are computed and the difference is stored.
3. The following formula is used:

$$D = \frac{AUC1 - AUC2}{s}$$

where `s` is the standard deviation of the bootstrap differences and `AUC1` and `AUC2` the AUC of the two (original) ROC curves.

4. `D` is then compared to the normal distribution, according to the value of alternative.

See also the Bootstrap section in [this package’s documentation](#).

With method=“deLong”, the processing is done as described in DeLong *et al.* (1988) for paired ROC curves. Only comparison of two ROC curves is implemented. The method has been extended

for unpaired ROC curves where the p-value is computed with an unpaired t-test with unequal sample size and unequal variance.

With `method="venkatraman"`, the processing is done as described in Venkatraman and Begg (1996) (for paired ROC curves) and Venkatraman (2000) (for unpaired ROC curves) with `boot.n` permutation of sample ranks (with ties breaking). For consistency reasons, the same argument `boot.n` as in `bootstrap` defines the number of permutations to execute, even though no bootstrap is performed.

For `method="specificity"`, the test assesses if the sensitivity of the ROC curves are different at the level of specificity given by the `specificity` argument, which must be a numeric of length 1. Bootstrap is employed as with `method="bootstrap"` and `boot.n` and `boot.stratified` are available. This is identical to the test proposed by Pepe *et al.* (2009). The `method="sensitivity"` is very similar, but assesses if the specificity of the ROC curves are different at the level of sensitivity given by the `sensitivity` argument.

Warnings

If “auc” specifications are different in both roc objects, the warning “Different AUC specifications in the ROC curves. Enforcing the inconsistency, but unexpected results may be produced.” is issued. Unexpected results may be produced.

If one or both ROC curves are “smooth.roc” objects with different smoothing specifications, the warning “Different smoothing parameters in the ROC curves. Enforcing the inconsistency, but unexpected results may be produced.” is issued. This warning can be benign, especially if ROC curves were generated with `roc(..., smooth=TRUE)` with different arguments to other functions (such as `plot`), or if you really want to compare two ROC curves smoothed differently.

If `method="delong"` and the AUC specification specifies a partial AUC, the warning “Using DeLong’s test for partial AUC is not supported. Using bootstrap test instead.” is issued. The `method` argument is ignored and “bootstrap” is used instead.

If `method="delong"` and the ROC curve is smoothed, the warning “Using DeLong’s test for smoothed ROCs is not supported. Using bootstrap test instead.” is issued. The `method` argument is ignored and “bootstrap” is used instead.

If `method="venkatraman"`, and the AUC specification specifies a partial AUC, the AUC specification is ignored with the warning “Partial AUC is ignored in Venkatraman’s test.”.

If `method="venkatraman"`, and `alternative` is “less” or “greater”, the warning “Only two-sided tests are available for Venkatraman. Performing two-sided test instead.” is produced and a two tailed test is performed.

Both DeLong and Venkatraman’s test ignores the direction of the ROC curve so that if two ROC curves have a different differ in the value of `direction`, the warning “(DeLong|Venkatraman)’s test should not be applied to ROC curves with different directions.” is printed. However, the spurious test is enforced.

If `boot.stratified=FALSE` and the sample has a large imbalance between cases and controls, it could happen that one or more of the replicates contains no case or control observation, or that there are not enough points for smoothing, producing a NA area. The warning “NA value(s) produced during bootstrap were ignored.” will be issued and the observation will be ignored. If you have a large imbalance in your sample, it could be safer to keep `boot.stratified=TRUE`.

Errors

An error will also occur if you give a predictor2 when predictor1 is a `matrix` or a `data.frame`, if predictor1 has more than two columns, or if you do not give a predictor2 when predictor1 is a vector.

If `density.cases` and `density.controls` were provided for smoothing, the error “Cannot compute the statistic on ROC curves smoothed with `density.controls` and `density.cases`.” is issued.

If `method="venkatraman"` and one of the ROC curves is smoothed, the error “Using Venkatraman’s test for smoothed ROCs is not supported.” is produced.

With `method="specificity"`, the error “Argument ‘specificity’ must be numeric of length 1 for a specificity test.” is given unless the specificity argument is specified as a numeric of length 1. The “Argument ‘sensitivity’ must be numeric of length 1 for a sensitivity test.” message is given for `method="sensitivity"` under similar conditions.

Acknowledgements

We would like to thank E. S. Venkatraman and Colin B. Begg for their support in the implementation of their test.

References

- Elisabeth R. DeLong, David M. DeLong and Daniel L. Clarke-Pearson (1988) “Comparing the areas under two or more correlated receiver operating characteristic curves: a nonparametric approach”. *Biometrics* **44**, 837–845.
- James A. Hanley and Barbara J. McNeil (1982) “The meaning and use of the area under a receiver operating characteristic (ROC) curve”. *Radiology* **143**, 29–36.
- Margaret Pepe, Gary Longton and Holly Janes (2009) “Estimation and Comparison of Receiver Operating Characteristic Curves”. *The Stata journal* **9**, 1.
- Xavier Robin, Natacha Turck, Jean-Charles Sanchez and Markus Müller (2009) “Combination of protein biomarkers”. *useR! 2009*, Rennes. http://www.agrocampus-ouest.fr/math/useR-2009/abstracts/user_author.html
- Xavier Robin, Natacha Turck, Alexandre Hainard, *et al.* (2011) “pROC: an open-source package for R and S+ to analyze and compare ROC curves”. *BMC Bioinformatics*, **7**, 77. DOI: [10.1186/1471-2105-12-77](https://doi.org/10.1186/1471-2105-12-77).
- E. S. Venkatraman and Colin B. Begg (1996) “A distribution-free procedure for comparing receiver operating characteristic curves from a paired experiment”. *Biometrika* **83**, 835–848. DOI: [10.1093/biomet/83.4.835](https://doi.org/10.1093/biomet/83.4.835).
- E. S. Venkatraman (2000) “A Permutation Test to Compare Receiver Operating Characteristic Curves”. *Biometrics* **56**, 1134–1138. DOI: [10.1111/j.0006-341X.2000.01134.x](https://doi.org/10.1111/j.0006-341X.2000.01134.x).

See Also

[roc](#)

Examples

```

data(aSAH)

# Basic example with 2 roc objects
roc1 <- roc(aSAH$outcome, aSAH$s100b)
roc2 <- roc(aSAH$outcome, aSAH$wfns)
roc.test(roc1, roc2)

## Not run:
# The latter used Delong's test. To use bootstrap test:
roc.test(roc1, roc2, method="bootstrap")
# Increase boot.n for a more precise p-value:
roc.test(roc1, roc2, method="bootstrap", boot.n=10000)

## End(Not run)

# Alternative syntaxes
roc.test(aSAH$outcome, aSAH$s100b, aSAH$wfns)
roc.test(aSAH$outcome, data.frame(aSAH$s100b, aSAH$wfns))

# If we had a good a priori reason to think that wfns gives a
# better classification than s100b (in other words, AUC of roc1
# should be lower than AUC of roc2):
roc.test(roc1, roc2, alternative="less")

## Not run:
# Comparison can be done on smoothed ROCs
# Smoothing is re-done at each iteration, and execution is slow
roc.test(smooth(roc1), smooth(roc2))
# or:
roc.test(aSAH$outcome, aSAH$s100b, aSAH$wfns, smooth=TRUE, boot.n=100)

## End(Not run)
# or from an AUC (no smoothing)
roc.test(auc(roc1), roc2)

## Not run:
# Comparison of partial AUC:
roc3 <- roc(aSAH$outcome, aSAH$s100b, partial.auc=c(1, 0.8), partial.auc.focus="se")
roc4 <- roc(aSAH$outcome, aSAH$wfns, partial.auc=c(1, 0.8), partial.auc.focus="se")
roc.test(roc3, roc4)
# This is strictly equivalent to:
roc.test(roc3, roc4, method="bootstrap")

# Alternatively, we could re-use roc1 and roc2 to get the same result:
roc.test(roc1, roc2, reuse.auc=FALSE, partial.auc=c(1, 0.8), partial.auc.focus="se")

# Comparison on specificity and sensitivity
roc.test(roc1, roc2, method="specificity", specificity=0.9)
roc.test(roc1, roc2, method="sensitivity", sensitivity=0.9)

## End(Not run)

```

```

# Spurious use of DeLong's test with different direction:
roc5 <- roc(aSAH$outcome, aSAH$s100b, direction="<")
roc6 <- roc(aSAH$outcome, aSAH$s100b, direction=">")
roc.test(roc5, roc6, method="delong")

## Not run:
# Comparisons of the ROC curves
roc.test(roc1, roc2, method="venkatraman")

## End(Not run)

# Unpaired tests
roc7 <- roc(aSAH$outcome, aSAH$s100b)
# artificially create an roc8 unpaired with roc7
roc8 <- roc(aSAH$outcome[1:100], aSAH$s100b[1:100])
## Not run:
roc.test(roc7, roc8, paired=FALSE, method="delong")
roc.test(roc7, roc8, paired=FALSE, method="bootstrap")
roc.test(roc7, roc8, paired=FALSE, method="venkatraman")
roc.test(roc7, roc8, paired=FALSE, method="specificity", specificity=0.9)

## End(Not run)

```

smooth.roc

Smooth a ROC curve

Description

This function smoothes a ROC curve of numeric predictor. By default, a binormal smoothing is performed, but density or custom smoothings are supported.

Usage

```

smooth(...)
## Default S3 method:
smooth(...)
## S3 method for class 'roc'
smooth(roc,
method=c("binormal", "density", "fitdistr", "logcondens",
"logcondens.smooth"), n=512, bw = "nrd0", density=NULL,
density.controls=density, density.cases=density,
start=NULL, start.controls=start, start.cases=start,
reuse.auc=TRUE, reuse.ci=FALSE, ...)
## S3 method for class 'smooth.roc'
smooth(smooth.roc, ...)

```

Arguments

<code>roc, smooth.roc</code>	a “roc” object from the <code>roc</code> function, or a “smooth.roc” object from the <code>smooth.roc</code> function.
<code>method</code>	“binormal”, “density”, “fitdistr”, “logcondens”, “logcondens.smooth”, or a function returning a list of smoothed sensitivities and specificities.
<code>n</code>	the number of equally spaced points where the smoothed curve will be calculated.
<code>bw</code>	if <code>method="density"</code> and <code>density.controls</code> and <code>density.cases</code> are not provided, <code>bw</code> is passed to <code>density</code> to determine the bandwidth of the density. Can be a character string (“nrd0”, “nrd”, “ucv”, “bcv” or “SJ”, but any name matching a function prefixed with “bw.” is supported) or a numeric value, as described in density . Defaults to “nrd0”.
<code>density, density.controls, density.cases</code>	if <code>method="density"</code> , a numeric value of density (over the y axis) or a function returning a density (such as density . If <code>method="fitdistr"</code> , a <code>densfun</code> argument for fitdistr . If the value is different for control and case observations, <code>density.controls</code> and <code>density.cases</code> can be employed instead, otherwise density will be propagated to both <code>density.controls</code> and <code>density.cases</code> .
<code>start, start.controls, start.cases</code>	if <code>method="fitdistr"</code> , optional start arguments for <code>start.controls</code> and <code>start.cases</code> allows to specify different distributions for controls and cases.
<code>reuse.auc, reuse.ci</code>	if TRUE (default for <code>reuse.auc</code>) and the “roc” objects contain “auc” or “ci” fields, re-use these specifications to regenerate auc or ci on the smoothed ROC curve with the original parameters. If FALSE, the object returned will not contain “auc” or “ci” fields. It is currently not possible to redefine auc and ci options directly: you need to call auc or ci later for that.
<code>...</code>	further arguments passed to or from other methods, and especially to density (only <code>cut</code> , <code>adjust</code> , and <code>kernel</code> , plus <code>window</code> for compatibility with S+) and fitdistr . Also passed to <code>method</code> if it is a function.

Details

If `method="binormal"`, a linear model is fitted to the quantiles of the sensitivities and specificities. Smoothed sensitivities and specificities are then generated from this model on `n` points. This simple approach was found to work well for most ROC curves, but it may produce hooked smooths in some situations (see in Hanley (1988)).

With `method="density"`, the [density](#) function is employed to generate a smooth kernel density of the control and case observations as described by Zhou *et al.* (1997), unless `density.controls` or `density.cases` are provided directly. `bw` can be given to specify a bandwidth to use with [density](#). It can be a numeric value or a character string (“nrd0”, “nrd”, “ucv”, “bcv” or “SJ”, but any name [matching](#) a function prefixed with “bw.” is supported). In the case of a character string, the whole predictor data is employed to determine the numeric value to use on both controls and cases. Depending on your data, it might be a good idea to specify the `kernel` argument for [density](#). By default, “gaussian” is used, but “epanechnikov”, “rectangular”, “triangular”, “biweight”, “cosine”

and “optcosine” are supported. As all the kernels are symetrical, it might help to normalize the data first (that is, before calling `roc`), for example with quantile normalization:

```
norm.x <- qnorm(rank(x)/(length(x)+1))
smooth(roc(response, norm.x, ...), ...)
```

Additionally, density can be a function which must return either a numeric vector of densities over the y axis or a [list](#) with a “y” item like the [density](#) function. It must accept the following input:

```
density.fun(x, n, from, to, bw, kernel, ...)
```

It is important to honour n, from and to in order to have the densities evaluated on the same points for controls and cases. Failing to do so and returning densities of different length will produce an error. It is also a good idea to use a constant smoothing parameter (such as bw) especially when controls and cases have a different number of observations, to avoid producing smoother or rougher densities.

If method=“fitdistr”, the [fitdistr](#) function from the **MASS** package is employed to fit parameters for the density function density with optionnal start parameters start. The density function are fitted separately in control (density.controls, start.controls) and case observations (density.cases, start.cases). density can be one of the character values allowed by [fitdistr](#) or a density function (such as [dnorm](#), [dweibull](#), ...).

The method=“logcondens” and method=“logcondens.smooth” use the **logcondens** package to generate a non smoothed or smoothed (respectively) log-concave density estimate of of the control and case observation with the [logConROC](#) function.

Finally, method can also be a function. It must return a list with exactly 2 elements named “sensitivities” and “specificities”, which must be numeric vectors between 0 and 1 or 100 (depending on the percent argument to [roc](#)). It is passed all the arguments to the smooth function.

smooth.default forces the usage of the [smooth](#) function in the **stats** package, so that other code relying on smooth should continue to function normally.

Smoothed ROC curves can be passed to smooth again. In this case, the smoothing is not re-applied on the smoothed ROC curve but the original “[roc](#)” object will be re-used.

Value

A list of class “smooth.roc” with the following fields:

sensitivities	the smoothed sensitivities defining the ROC curve.
specificities	the smoothed specificities defining the ROC curve.
percent	if the sensitivities, specificities and AUC are reported in percent, as defined in argument.
direction	the direction of the comparison, as defined in argument.
thresholds	the thresholds at which the sensitivities and specificities were computed.
call	how the function was called. See match.call for more details.

<code>smoothing.args</code>	a list of the arguments used for the smoothing. Will serve to apply the smoothing again in further bootstrap operations.
<code>auc</code>	if the original ROC curve contained an AUC, it is computed again on the smoothed ROC.
<code>ci</code>	if the original ROC curve contained a CI, it is computed again on the smoothed ROC.
<code>fit.controls</code> , <code>fit.cases</code>	with <code>method="fitdistr"</code> only: the result of MASS's <code>fitdistr</code> function for controls and cases, with an additional "densfun" item indicating the density function, if possible as character.
<code>logcondens</code>	with <code>method="logcondens"</code> and <code>method="logcondens"</code> only: the result of <code>logcondens</code> 's <code>logConROC</code> function.

Attributes: Additionally, the original `roc` object is stored as a "roc" attribute.

Errors

If `method` is a function, the return values will be checked thoroughly for validity (list with two numeric elements of the same length named "sensitivities" and "specificities" with values in the range of possible values for sensitivities and specificities).

The message "The 'density' function must return a numeric vector or a list with a 'y' item." will be displayed if the density function did not return a valid output. The message "Length of 'density.controls' and 'density.cases' differ." will be displayed if the returned value differ in length.

Binormal smoothing cannot smooth ROC curve defined by only one point. Any such attempt will fail with the error "ROC curve not smoothable (not enough points).".

If the smooth ROC curve was generated by `roc` with `density.controls` and `density.cases` numeric arguments, it cannot be smoothed and the error "Cannot smooth a ROC curve generated directly with numeric 'density.controls' and 'density.cases'." is produced.

`fitdistr` and `density` smoothing methods require a `numeric` predictor. If the ROC curve to smooth was generated with an ordered factor only binormal smoothing can be applied and the message "ROC curves of ordered predictors can be smoothed only with binormal smoothing." is displayed otherwise.

`fitdistr`, `logcondens` and `logcondens.smooth` methods require additional packages. If not available, the following message will be displayed with the required command to install the package: "Package ? not available, required with method=?". Please install it with `'install.packages("?")'`."

References

- James E. Hanley (1988) "The robustness of the "binormal" assumptions used in fitting ROC curves". *Medical Decision Making* **8**, 197–203.
- Lutz Duembgen, Kaspar Rufibach (2011) "logcondens: Computations Related to Univariate Log-Concave Density Estimation". *Journal of Statistical Software*, **39**, 1–28. URL: jstatsoft.org/v39/i06.
- Xavier Robin, Natacha Turck, Alexandre Hainard, *et al.* (2011) "pROC: an open-source package for R and S+ to analyze and compare ROC curves". *BMC Bioinformatics*, **7**, 77. DOI: [10.1186/1471-2105-12-77](https://doi.org/10.1186/1471-2105-12-77).

Kelly H. Zou, W. J. Hall and David E. Shapiro (1997) “Smooth non-parametric receiver operating characteristic (ROC) curves for continuous diagnostic tests”. *Statistics in Medicine* **18**, 2143–2156. DOI: [10.1002/\(SICI\)1097-0258\(19971015\)16:19<2143::AID-SIM655>3.0.CO;2-3](https://doi.org/10.1002/(SICI)1097-0258(19971015)16:19<2143::AID-SIM655>3.0.CO;2-3).

See Also

[roc](#)

Examples

```
data(aSAH)

## Basic example
rocobj <- roc(aSAH$outcome, aSAH$s100b)
smooth(rocobj)
# or directly with roc()
roc(aSAH$outcome, aSAH$s100b, smooth=TRUE)

# plotting
plot(rocobj)
rs <- smooth(rocobj, method="binormal")
plot(rs, add=TRUE, col="green")
rs2 <- smooth(rocobj, method="density")
plot(rs2, add=TRUE, col="blue")
rs3 <- smooth(rocobj, method="fitdistr", density="lognormal")
plot(rs3, add=TRUE, col="magenta")
rs4 <- smooth(rocobj, method="logcondens")
plot(rs4, add=TRUE, col="brown")
rs5 <- smooth(rocobj, method="logcondens.smooth")
plot(rs5, add=TRUE, col="orange")
legend("bottomright", legend=c("Empirical", "Binormal", "Density", "Log-normal",
                              "Log-concave density", "Smoothed log-concave density"),
      col=c("black", "green", "blue", "magenta", "brown", "orange"), lwd=2)

## Advanced smoothing

# if we know the distributions are normal with sd=0.1 and an unknown mean:
smooth(rocobj, method="fitdistr", density=dnorm, start=list(mean=1), sd=.1)
# different distributions for controls and cases:
smooth(rocobj, method="fitdistr", density.controls="normal", density.cases="lognormal")

# with densities
bw <- bw.nrd0(rocobj$predictor)
density.controls <- density(rocobj$controls, from=min(rocobj$predictor) - 3 * bw,
                           to=max(rocobj$predictor) + 3*bw, bw=bw, kernel="gaussian")
density.cases <- density(rocobj$cases, from=min(rocobj$predictor) - 3 * bw,
                        to=max(rocobj$predictor) + 3*bw, bw=bw, kernel="gaussian")
smooth(rocobj, method="density", density.controls=density.controls$y,
      density.cases=density.cases$y)
# which is roughly what is done by a simple:
smooth(rocobj, method="density")
```

```

# With logcondens:
smooth(rocobj, method="logcondens")
smooth(rocobj, method="logcondens.smooth")

## Smoothing artificial ROC curves

rand.unif <- runif(1000, -1, 1)
rand.exp <- rexp(1000)
rand.norm <-
rnorm(1000)

# two normals
roc.norm <- roc(controls=rnorm(1000), cases=rnorm(1000)+1, plot=TRUE)
plot(smooth(roc.norm), col="green", lwd=1, add=TRUE)
plot(smooth(roc.norm, method="density"), col="red", lwd=1, add=TRUE)
plot(smooth(roc.norm, method="fitdistr"), col="blue", lwd=1, add=TRUE)
plot(smooth(roc.norm, method="logcondens"), col="brown", lwd=1, add=TRUE)
plot(smooth(roc.norm, method="logcondens.smooth"), col="orange", lwd=1, add=TRUE)
legend("bottomright", legend=c("empirical", "binormal", "density", "fitdistr",
                              "logcondens", "logcondens.smooth"),
      col=c(par("fg"), "green", "red", "blue", "brown", "orange"), lwd=c(2, 1, 1, 1, 1))

# deviation from the normality
roc.norm.exp <- roc(controls=rnorm(1000), cases=rexp(1000), plot=TRUE)
plot(smooth(roc.norm.exp), col="green", lwd=1, add=TRUE)
plot(smooth(roc.norm.exp, method="density"), col="red", lwd=1, add=TRUE)
# Wrong fitdistr: normality assumed by default
plot(smooth(roc.norm.exp, method="fitdistr"), col="blue", lwd=1, add=TRUE)
# Correct fitdistr
plot(smooth(roc.norm.exp, method="fitdistr", density.controls="normal",
          density.cases="exponential"), col="purple", lwd=1, add=TRUE)
plot(smooth(roc.norm.exp, method="logcondens"), col="brown", lwd=1, add=TRUE)
plot(smooth(roc.norm.exp, method="logcondens.smooth"), col="orange", lwd=1, add=TRUE)
legend("bottomright", legend=c("empirical", "binormal", "density",
                              "wrong fitdistr", "correct fitdistr",
                              "logcondens", "logcondens.smooth"),
      col=c(par("fg"), "green", "red", "blue", "purple", "brown", "orange"), lwd=c(2, 1, 1, 1, 1, 1))

# large deviation from the normality
roc.unif.exp <- roc(controls=runif(1000, 2, 3), cases=rexp(1000)+2, plot=TRUE)
plot(smooth(roc.unif.exp), col="green", lwd=1, add=TRUE)
plot(smooth(roc.unif.exp, method="density"), col="red", lwd=1, add=TRUE)
plot(smooth(roc.unif.exp, method="density", bw="ucv"), col="magenta", lwd=1, add=TRUE)
# Wrong fitdistr: normality assumed by default (uniform distributions not handled)
plot(smooth(roc.unif.exp, method="fitdistr"), col="blue", lwd=1, add=TRUE)
plot(smooth(roc.unif.exp, method="logcondens"), col="brown", lwd=1, add=TRUE)
plot(smooth(roc.unif.exp, method="logcondens.smooth"), col="orange", lwd=1, add=TRUE)
legend("bottomright", legend=c("empirical", "binormal", "density",
                              "density ucv", "wrong fitdistr",
                              "logcondens", "logcondens.smooth"),
      col=c(par("fg"), "green", "red", "magenta", "blue", "brown", "orange"), lwd=c(2, 1, 1, 1, 1, 1))

```

```
# 2 uniform distributions with a custom density function
unif.density <- function(x, n, from, to, bw, kernel, ...) {
  smooth.x <- seq(from=from, to=to, length.out=n)
  smooth.y <- dunif(smooth.x, min=min(x), max=max(x))
  return(smooth.y)
}
roc.unif <- roc(controls=runif(1000, -1, 1), cases=runif(1000, 0, 2), plot=TRUE)
s <- smooth(roc.unif, method="density", density=unif.density)
plot(roc.unif)
plot(s, add=TRUE, col="grey")

## Not run:
# you can bootstrap a ROC curve smoothed with a density function:
ci(s, boot.n=100)

## End(Not run)
```

var.roc	<i>Variance of a ROC curve</i>
---------	--------------------------------

Description

These functions compute the variance of the AUC of a ROC curve.

Usage

```
var(...)
## Default S3 method:
var(...)
## S3 method for class 'auc'
var(auc, ...)
## S3 method for class 'roc'
var(roc, method=c("delong", "bootstrap"),
boot.n = 2000, boot.stratified = TRUE, reuse.auc=TRUE,
progress = getOption("pROCProgress")$name, ...)
## S3 method for class 'smooth.roc'
var(smooth.roc, ...)
```

Arguments

roc, smooth.roc, auc	a “roc” object from the roc function, a “smooth.roc” object from the smooth.roc function or an “auc” object from the auc function.
method	the method to use, either “delong” or “bootstrap”. The first letter is sufficient. If omitted, the appropriate method is selected as explained in details.
reuse.auc	if TRUE (default) and the “roc” objects contain an “auc” field, re-use these specifications for the test. See details.

<code>boot.n</code>	for <code>method="bootstrap"</code> only: the number of bootstrap replicates or permutations. Default: <i>2000</i> .
<code>boot.stratified</code>	for <code>method="bootstrap"</code> only: should the bootstrap be stratified (same number of cases/controls in each replicate than in the original sample) or not. Default: <i>TRUE</i> .
<code>progress</code>	the name of progress bar to display. Typically “none”, “win”, “tk” or “text” (see the name argument to create_progress_bar for more information), but a list as returned by create_progress_bar is also accepted. See also the “Progress bars” section of this package’s documentation .
<code>...</code>	further arguments passed to or from other methods, especially arguments for <code>var.roc</code> when calling <code>var</code> , <code>var.auc</code> and <code>var.smooth.roc</code> . Arguments for auc (if <code>reuse.auc=FALSE</code>) and txtProgressBar (only <code>char</code> and <code>style</code>) if applicable.

Details

The `var` function computes the variance of the AUC of a ROC curve. It is typically called with the [roc](#) object of interest. Two methods are available: “delong” and “bootstrap” (see “Computational details” section below).

The default is to use “delong” method except for with partial AUC and smoothed curves where “bootstrap” is employed. Using “delong” for partial AUC and smoothed ROCs is not supported (a warning is produced and “bootstrap” is employed instead).

For [smoothed ROC curves](#), smoothing is performed again at each bootstrap replicate with the parameters originally provided. If a density smoothing was performed with user-provided `density.cases` or `density.controls` the bootstrap cannot be performed and an error is issued.

`var.default` forces the usage of the [var](#) function in the **stats** package, so that other code relying on `var` should continue to function normally.

Value

The numeric value of the variance.

AUC specification

`var` needs a specification of the AUC to compute the variance of the AUC of the ROC curve. The specification is defined by:

1. the “auc” field in the “[roc](#)” objects if `reuse.auc` is set to *TRUE* (default)
2. passing the specification to [auc](#) with `...(arguments partial.auc, partial.auc.correct and partial.auc.focus)`. In this case, you must ensure either that the [roc](#) object do not contain an `auc` field (if you called [roc](#) with `auc=FALSE`), or set `reuse.auc=FALSE`.

If `reuse.auc=FALSE` the [auc](#) function will always be called with `...` to determine the specification, even if the “[roc](#)” objects do contain an `auc` field.

As well if the “[roc](#)” objects do not contain an `auc` field, the [auc](#) function will always be called with `...` to determine the specification.

Warning: if the roc object passed to roc.test contains an auc field and reuse.auc=TRUE, auc is not called and arguments such as partial.auc are silently ignored.

Computation details

With method="bootstrap", the processing is done as follow:

1. boot.n bootstrap replicates are drawn from the data. If boot.stratified is *TRUE*, each replicate contains exactly the same number of controls and cases than the original sample, otherwise if *FALSE* the numbers can vary.
2. for each bootstrap replicate, the AUC of the ROC curve is computed and stored.
3. the variance of the resampled AUCs are computed and returned.

With method="delong", the processing is done as described in Hanley and Hajian-Tilaki (1997).

Warnings

If method="delong" and the AUC specification specifies a partial AUC, the warning "Using DeLong for partial AUC is not supported. Using bootstrap test instead." is issued. The method argument is ignored and "bootstrap" is used instead.

If method="delong" and the ROC curve is smoothed, the warning "Using DeLong for smoothed ROCs is not supported. Using bootstrap test instead." is issued. The method argument is ignored and "bootstrap" is used instead.

If boot.stratified=FALSE and the sample has a large imbalance between cases and controls, it could happen that one or more of the replicates contains no case or control observation, or that there are not enough points for smoothing, producing a NA area. The warning "NA value(s) produced during bootstrap were ignored." will be issued and the observation will be ignored. If you have a large imbalance in your sample, it could be safer to keep boot.stratified=TRUE.

Errors

If density.cases and density.controls were provided for smoothing, the error "Cannot compute the covariance on ROC curves smoothed with density.controls and density.cases." is issued.

References

Elisabeth R. DeLong, David M. DeLong and Daniel L. Clarke-Pearson (1988) "Comparing the areas under two or more correlated receiver operating characteristic curves: a nonparametric approach". *Biometrics* **44**, 837–845.

James A. Hanley and Karim O. Hajian-Tilaki (1997) "Sampling variability of nonparametric estimates of the areas under receiver operating characteristic curves: An update". *Academic Radiology* **4**, 49–58. DOI: [10.1016/S1076-6332\(97\)80161-4](https://doi.org/10.1016/S1076-6332(97)80161-4).

See Also

[roc](#), [cov.roc](#)

Examples

```

data(aSAH)

## Basic example
roc1 <- roc(aSAH$outcome, aSAH$s100b)
roc2 <- roc(aSAH$outcome, aSAH$wfns)
var(roc1)
var(roc2)

# We could also write it in one line:
var(roc(aSAH$outcome, aSAH$s100b))

## Not run:
# The latter used Delong. To use bootstrap:
var(roc1, method="bootstrap")
# Decrease boot.n for a faster execution
var(roc1, method="bootstrap", boot.n=1000)

## End(Not run)

## Not run:
# Variance of smoothed ROCs:
# Smoothing is re-done at each iteration, and execution is slow
var(smooth(roc1))

## End(Not run)

# or from an AUC (no smoothing)
var(auc(roc1))

## Test data from Hanley and Hajian-Tilaki, 1997
disease.present <- c("Yes", "No", "Yes", "No", "No", "Yes", "Yes", "No", "No", "Yes", "No", "No", "Yes", "No", "No")
field.strength.1 <- c(1, 2, 5, 1, 1, 1, 2, 1, 2, 2, 1, 1, 5, 1, 1)
field.strength.2 <- c(1, 1, 5, 1, 1, 1, 4, 1, 2, 2, 1, 1, 5, 1, 1)
roc3 <- roc(disease.present, field.strength.1)
roc4 <- roc(disease.present, field.strength.2)
# Assess the variance:
var(roc3)
var(roc4)

## Not run:
# With bootstrap:
var(roc3, method="bootstrap")
var(roc4, method="bootstrap")

## End(Not run)

```

Index

- *Topic **aplot**
 - lines.roc, 36
 - plot.ci, 39
 - plot.roc, 41
 - pROC-package, 2
- *Topic **datasets**
 - aSAH, 8
- *Topic **hplot**
 - lines.roc, 36
 - plot.ci, 39
 - plot.roc, 41
 - pROC-package, 2
- *Topic **htest**
 - pROC-package, 2
 - roc.test, 52
- *Topic **logic**
 - are.paired, 6
 - has.partial.auc, 35
- *Topic **multivariate**
 - cov.roc, 30
 - roc.test, 52
- *Topic **nonparametric**
 - auc, 9
 - ci, 12
 - ci.auc, 14
 - ci.se, 17
 - ci.sp, 20
 - ci.thresholds, 23
 - coords, 26
 - cov.roc, 30
 - lines.roc, 36
 - multiclass.roc, 37
 - plot.ci, 39
 - plot.roc, 41
 - print, 46
 - pROC-package, 2
 - roc, 48
 - roc.test, 52
 - smooth.roc, 59
 - var.roc, 65
- *Topic **package**
 - pROC-package, 2
- *Topic **print**
 - print, 46
 - pROC-package, 2
- *Topic **programming**
 - are.paired, 6
 - has.partial.auc, 35
- *Topic **roc**
 - are.paired, 6
 - auc, 9
 - ci, 12
 - ci.auc, 14
 - ci.se, 17
 - ci.sp, 20
 - ci.thresholds, 23
 - coords, 26
 - cov.roc, 30
 - has.partial.auc, 35
 - lines.roc, 36
 - multiclass.roc, 37
 - plot.ci, 39
 - plot.roc, 41
 - print, 46
 - pROC-package, 2
 - roc, 48
 - roc.test, 52
 - smooth.roc, 59
 - var.roc, 65
- *Topic **smooth**
 - smooth.roc, 59
- *Topic **univar**
 - auc, 9
 - ci, 12
 - ci.auc, 14
 - ci.se, 17
 - ci.sp, 20
 - ci.thresholds, 23

- coords, 26
- lines.roc, 36
- multiclass.roc, 37
- plot.ci, 39
- plot.roc, 41
- print, 46
- pROC-package, 2
- roc, 48
- smooth.roc, 59
- var.roc, 65
- *Topic utilities**
 - auc, 9
 - ci, 12
 - ci.auc, 14
 - ci.se, 17
 - ci.sp, 20
 - ci.thresholds, 23
 - coords, 26
 - cov.roc, 30
 - lines.roc, 36
 - multiclass.roc, 37
 - plot.ci, 39
 - plot.roc, 41
 - print, 46
 - pROC-package, 2
 - roc, 48
 - roc.test, 52
 - smooth.roc, 59
 - var.roc, 65
- abline, 44
- are.paired, 3, 6, 31, 33, 53, 54
- as.numeric, 51
- as.ordered, 51
- aSAH, 3, 8
- auc, 2, 3, 7, 8, 9, 13–17, 31, 32, 35, 38, 44, 45, 47–51, 53–55, 60, 65–67
- ci, 2, 3, 7, 8, 10, 12, 17, 20, 22, 25, 38, 45, 47–51, 60
- ci.auc, 3, 8, 11, 13, 14, 49
- ci.se, 3, 8, 13, 17, 22, 39, 40, 49
- ci.sp, 3, 8, 13, 20, 20, 39, 40, 49
- ci.thresholds, 3, 8, 13, 23, 39, 40, 49
- combined, 37
- confidence interval, 4
- coords, 3, 8, 26, 43, 47, 48
- cov, 3, 31
- cov (cov.roc), 30
- cov.roc, 30, 67
- create_progress_bar, 15, 18, 21, 24, 31, 54, 66
- data.frame, 57
- density, 60, 61
- dnorm, 61
- dweibull, 61
- Extract, 27
- fitdistr, 60–62
- has.partial.auc, 3, 35
- identical, 7
- interactive, 4
- lines, 36
- lines.roc, 3, 36
- lines.smooth.roc (lines.roc), 36
- list, 61
- logConROC, 61, 62
- match.call, 38, 51, 61
- matching, 60
- matrix, 57
- model.frame, 37, 48, 53
- multiclass.roc, 9, 11, 37
- na.omit, 38, 50
- nrd0, 60
- numeric, 62
- par, 36, 43, 44
- plot, 2, 3, 10, 43, 44
- plot.ci, 3, 8, 20, 22, 39, 44
- plot.default, 36
- plot.roc, 8, 36, 40, 41, 49–51
- plot.smooth.roc (plot.roc), 41
- plot.window, 43
- points, 43, 44
- polygon, 39, 44
- print, 3, 46
- print.roc, 8, 51
- pROC (pROC-package), 2
- pROC-package, 2
- qnorm, 15
- rank, 54

`roc`, 2, 3, 7–22, 24–26, 29, 31–33, 36, 37, 41,
42, 44, 45, 47, 48, 53–55, 57, 60–63,
65–67
`roc.test`, 2, 3, 7, 8, 48, 50, 51, 52

`segments`, 39
`signif`, 47
significance testing, 4
`smooth`, 3, 8, 49, 51, 61
`smooth(smooth.roc)`, 59
`smooth.roc`, 2, 7, 9, 11, 12, 14, 18, 21, 26, 31,
48, 49, 53, 59, 60, 65
`smoothed`, 24
smoothed ROC curves, 15, 19, 22, 31, 54, 66
`sprintf`, 43
`suppressWarnings`, 39, 45

`text`, 44
this package's documentation, 15, 18, 19,
21, 24, 31, 54, 55, 66
`tkProgressBar`, 4
`txtProgressBar`, 4, 15, 18, 21, 24, 31, 54, 66

`var`, 3, 66
`var(var.roc)`, 65
`var.roc`, 32, 33, 65