

Package ‘NSCA’

October 26, 2026

Type Package

Title Necessary and Sufficient Condition Analysis

Version 0.5.2

Date 2026-10-26

Description Tests whether a condition is both necessary and sufficient for an outcome, that is, whether the outcome cannot be reached without the condition and is guaranteed in its presence. Such a claim implies that two diagonally opposite corners of the condition-outcome scatter plot contain no observations. The package estimates both boundaries with the same data, scope and frontier technique, reports the effect size and permutation test of each, and indicates joint support only when both components are supported. The two components are summarized by the weaker effect, a balanced combination and the proportion of the scope that the two claims jointly rule out. A dual-threshold table gives, for each outcome level, the condition level it requires, the level sufficient for it, and the interval between them. Results can be compared with regression slopes. The package also provides plots, influential-case diagnostics, scope-sensitivity and resampling checks, and simulation and power analysis for planning studies. The two sides are estimated with the 'NCA' and 'SCAtools' packages.

License GPL (>= 3)

Encoding UTF-8

Depends R (>= 3.5.0)

Imports ggplot2 (>= 3.4.0), NCA (>= 5.0.2), SCAtools (>= 0.5.2), stats, utils

Suggests testthat (>= 3.0.0)

Config/testthat/edition 3

URL <https://github.com/youngchanresearcher/NSCA>

BugReports <https://github.com/youngchanresearcher/NSCA/issues>

NeedsCompilation no

Config/roxygen2/version 8.1.1

Author Young Chan [aut, cre]

Maintainer Young Chan <youngchanresearcher@gmail.com>

Repository CRAN

Date/Publication 2026-10-26 19:40:02 UTC

Contents

NSCA-package	3
nsca	6
nsca_analysis	7
nsca_compare	12
nsca_corners	14
nsca_direction_map	14
nsca_extract	15
nsca_joint	16
nsca_legacy_names	17
nsca_locationshift	18
nsca_magnitude	19
nsca_normalize	20
nsca_outliers	21
nsca_output	23
nsca_plot	24
nsca_power	25
nsca_powerplot	29
nsca_random	30
nsca_reference	32
nsca_resample	33
nsca_rescale	34
nsca_results	36
nsca_scales	37
nsca_scope	38
nsca_scope_sensitivity	39
nsca_shared_plot	41
nsca_table	42
nsca_terms	47
nsca_test_plot	48
nsca_thresholds	49

Index	52
--------------	-----------

Description

Tests whether a condition is both necessary and sufficient for an outcome, that is, whether the outcome cannot be reached without the condition and is guaranteed in its presence. Such a claim implies that two diagonally opposite corners of the condition-outcome scatter plot contain no observations. The package estimates both boundaries with the same data, scope and frontier technique, reports the effect size and permutation test of each, and indicates joint support only when both components are supported. The two components are summarized by the weaker effect, a balanced combination and the proportion of the scope that the two claims jointly rule out. A dual-threshold table gives, for each outcome level, the condition level it requires, the level sufficient for it, and the interval between them. Results can be compared with regression slopes. The package also provides plots, influential-case diagnostics, scope-sensitivity and resampling checks, and simulation and power analysis for planning studies. The two sides are estimated with the 'NCA' and 'SCAtools' packages.

The method

A condition is necessary for an outcome when the outcome does not occur without it, and sufficient when the condition guarantees the outcome. In a scatter plot of condition against outcome, each claim rules out one corner. Necessary Condition Analysis (NCA; Dul, 2016) estimates the empty corner of a necessity claim, and Sufficient Condition Analysis, implemented in the SCAtools package, estimates the empty corner of a sufficiency claim. A necessary-and-sufficient claim rules out both corners, which lie diagonally opposite each other, so NSCA estimates both on the same observations, scope and frontier technique and assesses them together. The band between the two frontiers is the admissible region: the narrower it is, the closer the relationship comes to a strict necessary-and-sufficient one. The framework, and the vocabulary the tables use, follow Chan (in press), *From Necessity to Sufficiency and Necessity-and-Sufficiency: Extending Condition Analysis in Degree*.

What it is for

- Testing the claim that a condition is both required and sufficient for an outcome; the joint claim is supported only when both components are.
- Estimating the thresholds: for each outcome level, the condition level the outcome requires, the level that is sufficient for it, and the interval between them ([nsca_thresholds\(\)](#)).
- Describing how closely the relationship approaches a necessary-and-sufficient one, through both component effect sizes and three summaries that combine them ([nsca_table\(\)](#)).
- Comparing the result with regression ([nsca_compare\(\)](#)) and planning the sample size of a study ([nsca_power\(\)](#)).

How to use it

1. The claim is fitted with `nsca_analysis()`, which takes the condition, the outcome, the direction and, if theory provides one, the scope. `test.rep` sets the number of permutations for the two permutation tests; `nsca()` runs 1000 permutations and draws the plot in one step.
2. The results are reported by `nsca_results()`, `nsca_table()`, `nsca_thresholds()` and `nsca_plot()`; `nsca_output()` prints and draws them together.
3. `nsca_compare()` and `nsca_locationshift()` set the result beside regression.
4. `nsca_test_plot()`, `nsca_scope()`, `nsca_scope_sensitivity()`, `nsca_resample()` and `nsca_outliers()` provide diagnostic checks.

Joint support

The joint claim is supported only when both components are: both directional permutation tests reject, combined by the intersection-union rule (Berger, 1982), $p_{\text{nsca_iut}} = \max(p_{\text{nec}}, p_{\text{suf}})$; any pre-specified relevance threshold is met; and the two estimated frontiers form an acceptable geometry. The two tests can be based on one shared permutation sequence, which also allows statistics involving both components to be tested: the smaller of the two empty areas ($p_{\text{weakest_perm}}$) and the two together ($p_{\text{sum_perm}}$). Each p-value addresses a different question, and we suggest choosing the one that corresponds to the research question before examining the results. Each component effect size can be read against the conventional bands of Dul (2016). The joint summaries are constructions specific to this package; they may be calibrated for the design at hand with `nsca_power()` and `nsca_resample()`.

Reading the result

A fitted model describes the empty-space pattern in the sample under the stated scope and frontier technique, tested against random pairing of the condition and the outcome, and its thresholds are empirical rules for the population and design studied. Claims beyond the sample – that the condition causes the outcome, or that future cases will follow the frontiers – depend on the research design. Each direction predicts one empty corner on each side, which assumes that the relationship is monotone; it is therefore advisable to plot the data before choosing a direction.

What the permutation tests compare

Each p-value in this package compares the observed empty areas with the areas produced by randomly re-pairing the same values of the condition and the outcome, the type of test that NCA also uses. A small p-value therefore indicates that a corner is emptier than random pairing would leave it; on its own it does not establish that the corner is empty in the population.

This is most relevant when one corner is largely empty and the other is not. A strong pattern on one side changes how the values of the outcome are distributed; random pairing then fills the opposite corner more than the data do, so that corner may appear emptier than under random pairing even where it is not empty, and its test may reject more often than the nominal level. A simple correlation can have the same effect on both corners. In our simulations this tendency was noticeable, and its size depended on the strength of the other side, the sample size and the distributions involved; the scripts and their results are installed with the package, in `system.file("simulations", package = "NSCA")`. The intersection-union test does not remove it, since it combines the two component tests as they are, and the two supplementary tests are more susceptible to it (see `nsca_table()`).

Researchers may wish to keep this in mind in three ways:

- reading each p-value together with the effect sizes, the scatter plot and theory, especially when one side is much stronger than the other;
- checking a planned design by simulating it with one side absent, for example `nsca_power(d_nec = 0.25, d_suf = 0)`, where every rejection of the joint claim is a false positive (see [nsca_power\(\)](#));
- stating in a report that the tests compare the data with random pairing.

These are offered as considerations rather than rules; how much weight they deserve depends on the design and the claim.

Regression as a reference

An ordinary least-squares or median regression line can be drawn beside the two frontiers as a central-tendency reference. It summarizes the average effect and is not a component of either claim. Chan (in press), *Mean Effects Are Not Sufficiency Effects: How Sufficient Condition Analysis Complements Regression Analysis*, explains why the two kinds of effect are reported side by side.

Author(s)

Maintainer: Young Chan <youngchanresearcher@gmail.com>

Authors:

- Young Chan <youngchanresearcher@gmail.com>

References

- Berger, R. L. (1982). Multiparameter hypothesis testing and acceptance sampling. *Technometrics*, 24(4), 295-300.
- Chan, Y. (in press). *From Necessity to Sufficiency and Necessity-and-Sufficiency: Extending Condition Analysis in Degree*. University of Leeds.
- Chan, Y. (in press). *Mean Effects Are Not Sufficiency Effects: How Sufficient Condition Analysis Complements Regression Analysis*. University of Leeds.
- Dul, J. (2016). Necessary Condition Analysis (NCA): Logic and methodology of "necessary but not sufficient" causality. *Organizational Research Methods*, 19(1), 10-52. doi:10.1177/1094428115584005

See Also

[nsca_analysis\(\)](#), [nsca_table\(\)](#), [nsca_thresholds\(\)](#), [nsca_plot\(\)](#), [nsca_terms\(\)](#)

nsca

*Quick necessary and sufficient condition analysis***Description**

A convenience wrapper around `nsca_analysis()` that also runs the permutation tests.

Usage

```
nsca(
  data,
  x,
  y,
  direction = "HH",
  ceilings = c("ce_fdh", "cr_fdh"),
  reference = NULL,
  test.rep = 1000,
  test.seed = NULL
)
```

Arguments

<code>data</code>	A data frame or object coercible to a data frame.
<code>x</code>	Columns containing one or more conditions.
<code>y</code>	A single outcome column.
<code>direction</code>	Necessary-and-sufficient direction(s): "HH", "LH", "HL", or "LL". The first letter is the condition level and the second the outcome level.
<code>ceilings</code>	One or more empty-space frontier techniques, applied identically to both sides. "ols" is not accepted here because it estimates central tendency rather than an empty space; it can be supplied as reference instead.
<code>reference</code>	Central-tendency lines drawn beside the two frontiers for comparison, or NULL for none. "ols" is the ordinary least-squares regression of the outcome on the condition, fitted to all the data. It is an average-effect summary; it does not enter any joint index, threshold or support decision, and is reported by <code>nsca_reference()</code> .
<code>test.rep</code>	Number of permutation resamples for the engines' own component tests. Zero skips testing, in which case no joint-support decision can be reached. The p-value cannot fall below $1 / \text{test.rep}$, nor below $1 / n!$ for a sample of n observations, so a count or a sample too small to reach <code>test.p_threshold</code> is reported with a warning.
<code>test.seed</code>	Optional seed for the two component permutation tests, so that <code>p_nec</code> , <code>p_suf</code> and therefore <code>p_nsca_iut</code> and <code>joint_support</code> are reproducible. The caller's random stream is restored afterward. A single seed yields two sequences. The two components are tested on two independent shuffles, and seeding both engines with the same number would give

them the same sequence, that is, the shared-sequence design obtained unintentionally and without the bookkeeping that makes it interpretable. The seed given here is therefore used once, to draw two component seeds; `shared.test.rep` and `shared.seed` remain the way to request a single sequence explicitly.

Details

Because the tests run by default, we suggest supplying `test.seed` so that the p-values are reproducible. An unseeded permutation test draws its seed from the clock and reports a different p-value on each run, which makes the value unsuitable for reporting; the `Settings` block states whether a seed was fixed and, if it was not, adds a note that the p-values may differ on the next run.

Value

An object of class `nsca_result`.

See Also

[nsca_analysis\(\)](#)

Examples

```
set.seed(1)
x <- sort(runif(60))
dat <- data.frame(X = x, Y = pmin(pmax(x + rnorm(60, 0, 0.1), 0), 1))

# The default 1000 permutations take longer than an example should;
# nsca_analysis() runs none unless asked.
nsca(dat, "X", "Y")
```

nsca_analysis

Necessary and sufficient condition analysis

Description

Estimates the two empty-space frontiers implied by a necessary-and-sufficient statement and assesses them jointly. The sufficiency side is estimated by [SCAtools::sca_analysis\(\)](#) and the necessity side by [NCA::nca_analysis\(\)](#), on diagonally opposite corners of the same scatter plot.

Usage

```
nsca_analysis(
  data,
  x,
  y,
  direction = "HH",
  ceilings = c("ce_fdh", "cr_fdh"),
```

```

reference = NULL,
scope = NULL,
threshold.x = "percentage.range",
threshold.y = "percentage.range",
convention = c("absolute", "directional"),
steps = 10,
step.size = NULL,
cutoff = 0,
qr.tau = 0.95,
reference.tau = 0.5,
test.rep = 0,
test.p_confidence = 0.95,
test.p_threshold = 0.05,
relevance = NULL,
shared.test.rep = 0,
test.seed = NULL,
shared.seed = NULL,
correspondence.tolerance = 0,
geometry.max_overlap = 0.01,
geometry.max_reconstruction = 0.02,
na.action = c("pairwise", "listwise"),
purity = FALSE
)

```

Arguments

<code>data</code>	A data frame or object coercible to a data frame.
<code>x</code>	Columns containing one or more conditions.
<code>y</code>	A single outcome column.
<code>direction</code>	Necessary-and-sufficient direction(s): "HH", "LH", "HL", or "LL". The first letter is the condition level and the second the outcome level.
<code>ceilings</code>	One or more empty-space frontier techniques, applied identically to both sides. "ols" is not accepted here because it estimates central tendency rather than an empty space; it can be supplied as reference instead.
<code>reference</code>	Central-tendency lines drawn beside the two frontiers for comparison, or NULL for none. "ols" is the ordinary least-squares regression of the outcome on the condition, fitted to all the data. It is an average-effect summary; it does not enter any joint index, threshold or support decision, and is reported by nsca_reference() .
<code>scope</code>	Optional theoretical scope <code>c(xmin, xmax, ymin, ymax)</code> , shared by both sides.
<code>threshold.x</code> , <code>threshold.y</code>	Reporting scales for nsca_thresholds() , as in SCAtools::sca_analysis() .
<code>convention</code>	Reporting convention, as in SCAtools::sca_analysis() .
<code>steps</code>	Number of outcome levels, or an explicit vector of levels on the <code>threshold.y</code> scale.
<code>step.size</code>	Optional spacing between outcome levels.

cutoff	How out-of-range threshold values are represented.
qr.tau	Quantile used by the quantile-regression frontier, stated as the share of observations the line is intended to leave on the non-empty side. It is read that way in both directions: the engine fits τ when the empty space is an upper corner and $1 - \tau$ when it is a lower one (corners 3 and 4), so the default 0.95 is the 95th-percentile line where the empty space lies above and the 5th-percentile line where it lies below. Sufficiency in the "HH" direction has its empty space at high X with low Y, which is corner 4, and therefore fits the 5th percentile: a floor, not a ceiling. No manual inversion of <code>qr.tau</code> is needed to compensate. Two reported quantities behave differently for <code>qr</code> than for the other frontiers, and both follow from the line being placed by a quantile rather than by the observations at the edge of the cloud. <code>frontier_accuracy</code> counts the share of observations on the non-empty side, which quantile regression sets to <code>qr.tau</code> by construction, so a value of 0.95 reflects the input rather than evidence of a good fit. <code>fit</code> compares this frontier's empty area with CE-FDH's, and the engine returns NA whenever this area is the larger of the two. This often happens for a quantile line, because it may pass through observations where an envelope cannot. An NA fit for <code>qr</code> is therefore an ordinary result rather than a failure.
reference.tau	Quantile for a "qr" reference line, defaulting to 0.5 – the median line. This is separate from <code>qr.tau</code> : a <code>qr</code> frontier lies at the edge of the cloud and uses <code>qr.tau</code> , whereas a <code>qr</code> reference line lies in its middle and uses <code>reference.tau</code> . Both can be requested at once, which the engine cannot express in a single call, so reference lines are computed by SCAtools rather than by the engine.
test.rep	Number of permutation resamples for the engines' own component tests. Zero skips testing, in which case no joint-support decision can be reached. The p-value cannot fall below $1 / \text{test.rep}$, nor below $1 / n!$ for a sample of n observations, so a count or a sample too small to reach <code>test.p_threshold</code> is reported with a warning.
test.p_confidence	Confidence level for permutation p-value accuracy.
test.p_threshold	Significance level used by the intersection-union combination of the two directional component tests.
relevance	Optional pre-specified practical-relevance thresholds for the component effect sizes, as one number applied to both or <code>c(necessity, sufficiency)</code> . Support requires an effect at least this large as well as a significant test. NULL means no relevance criterion was pre-specified, and support then rests on significance alone.
shared.test.rep	Number of replications of the shared permutation sequence. Zero, the default, skips it; <code>p_weakest_perm</code> and <code>p_sum_perm</code> are then NA and the component p-values come from the engines. Its p-values cannot fall below $1 / (\text{shared.test.rep} + 1)$, and a count too small to reach <code>test.p_threshold</code> is reported with a warning.
test.seed	Optional seed for the two component permutation tests, so that <code>p_nec</code> , <code>p_suf</code> and therefore <code>p_nsca_iut</code> and <code>joint_support</code> are reproducible. The caller's random stream is restored afterward.

A single seed yields two sequences. The two components are tested on two independent shuffles, and seeding both engines with the same number would give them the same sequence, that is, the shared-sequence design obtained unintentionally and without the bookkeeping that makes it interpretable. The seed given here is therefore used once, to draw two component seeds; `shared.test.rep` and `shared.seed` remain the way to request a single sequence explicitly.

<code>shared.seed</code>	Optional seed for the shared sequence. The caller's random stream is restored afterward.
<code>correspondence.tolerance</code>	How wide a necessity-sufficiency interval still counts as threshold correspondence, in the units of the condition. In a perfectly necessary-and-sufficient relation the two thresholds coincide at every outcome level, so the interval between them is zero; this argument sets how far from zero still counts as close enough. The default of 0 is not a recommendation. The width at which a relation ceases to be necessary-and-sufficient is a substantive claim about the subject matter, and a package-supplied default would be the kind of unjustified tolerance the framework cautions against. We suggest setting it only when a particular width can be justified, and otherwise leaving it at zero and reading the interval itself. It is distinct from the floating-point tolerance that <code>nsca_thresholds()</code> uses to decide whether two computed numbers are equal. That tolerance is a numerical matter; this one is substantive.
<code>geometry.max_overlap</code>	Largest excess frontier overlap, beyond what a step frontier produces by construction, that still counts as acceptable geometry.
<code>geometry.max_reconstruction</code>	Largest residual of the area identity that still counts as acceptable geometry.
<code>na.action</code>	How missing values are dropped, as in <code>SCAtools::sca_analysis()</code> . Both sides use the same rule, because a necessity effect size and a sufficiency effect size are comparable only when both are areas over the same box.
<code>purity</code>	Whether to compute the engine's additional purity metrics where it offers them. Off by default: the engine computes them only for a corner with neither axis flipped, so at most one side of an NSCA model can have them, and which side, if any, depends on the direction. No NSCA statistic uses them, and they are computationally expensive on data with many frontier points.

Value

An object of class `nsca_result`.

Matched estimation

Both sides are estimated with the same frontier technique, the same theoretical scope, the same observations and the same outcome levels. Each joint index compares the two empty zones, and such a comparison is meaningful only when both sides are measured in the same way: envelopment frontiers follow the data closely and yield the largest empty areas, whereas regression frontiers cut into them, so mixing techniques across sides would let the choice of technique, rather than the evidence, determine which side appears weaker. Supplying several techniques produces one

internally matched row per technique, which offers a way to check whether a conclusion is robust to that choice.

Degenerate scopes

A constant outcome is refused when no theoretical scope is given, because it has no scope of its own and every empty area would be measured against a scope of zero height. With a scope imposed, the analysis runs but issues a warning and marks the affected rows degenerate in `nsca_table()`. An axis that does not vary inside the declared scope makes every reported area a property of the scope rather than of the data, and the resulting areas overstate the evidence: a constant outcome at the center of the scope leaves the upper and lower halves both empty, so both components reach 0.5, the admissible region closes, and every joint index reaches its maximum for data that carry no information. A milder warning is issued when the observations span less than five percent of the declared scope on either axis.

The shared permutation sequence

`shared.test.rep` replaces the engines' two separate permutation runs with a single sequence used for both. Each replication shuffles the outcome once, refits both engines on that same shuffled frame, and records `d_nec`, `d_suf` and their minimum together. This is what makes `p_weakest_perm` interpretable: the null distribution of a statistic involving both components depends on how they co-vary under random pairing, and independent permutations would remove that dependence. When it is used, `p_nec` and `p_suf` are taken from the same sequence, the `p_source` column reports "shared", and all five p-values – `p_nec`, `p_suf`, `p_nsca_iut`, `p_weakest_perm` and `p_sum_perm` – are mutually consistent. It requires two engine fits per replication and is therefore off by default.

The orderings of the outcome are drawn following the rule NCA uses for its own permutation test: when `shared.test.rep` is 720 or fewer, all orderings are distinct, and a count larger than the $n!$ orderings a sample of n has is lowered to $n!$, so that each ordering is used once and the p-values are exact.

References

- Chan, Y. (in press). *From Necessity to Sufficiency and Necessity-and-Sufficiency: Extending Condition Analysis in Degree*. University of Leeds.
- Dul, J. (2016). Necessary Condition Analysis (NCA): Logic and methodology of "necessary but not sufficient" causality. *Organizational Research Methods*, 19(1), 10-52. doi:[10.1177/1094428115584005](https://doi.org/10.1177/1094428115584005)

See Also

`nsca_table()`, `nsca_results()`, `nsca_joint()`, `nsca_thresholds()`, `nsca_corners()`

Examples

```
set.seed(1)
x <- sort(runif(60))
dat <- data.frame(X = x, Y = pmin(pmax(x + rnorm(60, 0, 0.1), 0), 1))
fit <- nsca_analysis(dat, "X", "Y", direction = "HH", ceilings = "ce_fdh")
fit
```

nsca_compare

*Average effect beside both condition claims***Description**

Reports the ordinary least-squares coefficient beside the necessity and the sufficiency effect size for the same relationship, and classifies each pair into the four configurations of the condition analysis in degree framework.

Usage

```
nsca_compare(
  model,
  x = NULL,
  ceiling = NULL,
  beta_cut = .nsca_beta_cut_default,
  d_cut = NULL
)
```

Arguments

model	An object returned by <code>nsca_analysis()</code> .
x	Optional condition names or positions. All conditions by default.
ceiling	Frontier technique. Defaults to the first technique estimated.
beta_cut	Absolute standardized coefficient at or above which the average effect counts as present.
d_cut	Effect size at or above which the empty corner counts as present: one number, or <code>c(necessity = , sufficiency =)</code> . Defaults to the model's relevance, then to 0.10.

Value

A data frame with one row per condition and component.

Why two rows per condition

There is one average effect and there are two boundaries. A flat slope with an empty necessity corner and a flat slope with an empty sufficiency corner are different findings, so a single row per condition would have to select one of them without indicating which. Each condition therefore produces a necessity row and a sufficiency row; beta and the slope columns repeat down the pair, because they describe the same line.

What is being compared

beta is the standardized regression coefficient, which describes how far the expected outcome moves with the condition. effect_size is the share of the scope left empty in the corner that the component's claim excludes. Neither implies the other. A relationship can have a strong average effect and no empty corner, or an empty corner and almost no slope.

The four configurations

- I** Neither the average effect nor the empty corner reaches its cut.
- II** A flat slope with a largely empty corner, often reflecting a level shift or a threshold. This configuration is not detectable by a design that reports only a regression coefficient, because configurations I and II have the same flat slope.
- III** A slope with counterexamples still present. The condition shifts the expected outcome without bounding it.
- IV** Both an average effect and a largely empty corner.

The numbering follows the framework, where it is stated for sufficiency. The necessity rows apply the same two questions to the other corner. This reflects the argument on which NCA was proposed, namely that a condition can be necessary without raising the expected outcome, rather than a new typology; the numerals are nevertheless borrowed, as acknowledged here.

The cuts

`beta_cut` defaults to 0.30, the lower edge of the medium band in Cohen (1988). `d_cut` defaults to the relevance given to `nsca_analysis()`, per component, when one was pre-specified, because the researcher has then already stated a criterion that this function does not override; otherwise it defaults to 0.10, the lower edge of the medium band in Dul (2016). One number may be supplied for both components, or `c(necessity = , sufficiency = )` for separate values. The cuts used are returned as columns, so that each table records the values that produced its classification.

These are conventions, not benchmarks calibrated for this design.

`effect_size_null`, `effect_size_excess` and `effect_size_critical` are reported beside the classification for that reason. A condition can reach `d_cut` with an effect size that largely reflects the envelope fitting sample extremes, and the label would read "dominant" in either case. The cut is kept at its conventional value; the table adds the values that qualify it. See `nsca_table()` for how the floor is measured.

References

Chan, Y. (in press). *Mean Effects Are Not Sufficiency Effects: How Sufficient Condition Analysis Complements Regression Analysis*. University of Leeds.

Chan, Y. (in press). *From Necessity to Sufficiency and Necessity-and-Sufficiency: Extending Condition Analysis in Degree*. University of Leeds.

See Also

`nsca_locationshift()`, `nsca_reference()`, `nsca_table()`, `SCAtools::sca_compare()`

Examples

```
set.seed(1)
x <- sort(runif(60))
dat <- data.frame(X = x, Y = pmin(pmax(x + rnorm(60, 0, 0.1), 0), 1))
fit <- nsca_analysis(dat, "X", "Y", ceilings = "ce_fdh")
nsca_compare(fit)
```

nsca_corners	<i>Empty corners of a necessary-and-sufficient statement</i>
--------------	--

Description

Returns the two scatter-plot corners that a necessary-and-sufficient statement implies are empty. For each direction, the necessity corner is diagonally opposite the sufficiency corner.

Usage

```
nsca_corners(direction = "HH")
```

Arguments

direction	One or more of "HH", "LH", "HL", "LL". The first letter refers to the level of the condition and the second to the level of the outcome.
-----------	--

Value

A data frame with the necessity and sufficiency corners and the side of the point cloud each frontier bounds.

Examples

```
nsca_corners("HH")
nsca_corners(c("HH", "LL"))
```

nsca_direction_map	<i>Complete mapping between directions, corners and component statements</i>
--------------------	--

Description

Reproduces Table 1 of the condition analysis in degree framework, which sets out the main directional types of the X-Y relationship together with the expected empty space and boundary each implies for necessity and for sufficiency, and adds the corner numbering the frontier engines use.

Usage

```
nsca_direction_map()
```

Details

The direction is a claim about the theorized relationship, not about the shape of the estimated frontier, which may be a step function or a straight line under any of the four types. Theory may instead posit a curvilinear or U-shaped relationship, for which the expected empty space is no longer a single corner; such relationships are outside the scope of this package.

Value

A data frame describing all four necessary-and-sufficient directions. The biconditional column retains this name because it contains the joint statement written out in full, and the shorter logical term is more convenient as a column name.

References

Chan, Y. (in press). *From Necessity to Sufficiency and Necessity-and-Sufficiency: Extending Condition Analysis in Degree*. University of Leeds.

See Also

[nsca_corners\(\)](#), [nsca_terms\(\)](#)

Examples

```
nsca_direction_map()
```

nsca_extract	<i>Extract one value from an NSCA result</i>
--------------	--

Description

Extract one value from an NSCA result

Usage

```
nsca_extract(model, x = NULL, ceiling = NULL, param = "weakest_effect")
```

Arguments

model	An object returned by nsca_analysis() .
x	Condition name. Defaults to the first condition.
ceiling	Frontier technique. Defaults to the first requested.
param	A column of nsca_table() , or a parameter name understood by the underlying engines, prefixed with "nec:" or "suf:". Retired column names, listed by nsca_legacy_names() , are rejected with an error that names their replacement.

Value

The selected value.

See Also

[nsca_table\(\)](#), [nsca_legacy_names\(\)](#)

Examples

```
set.seed(1)
x <- sort(runif(60))
dat <- data.frame(X = x, Y = pmin(pmax(x + rnorm(60, 0, 0.1), 0), 1))
fit <- nsca_analysis(dat, "X", "Y", ceilings = "ce_fdh")
nsca_extract(fit, param = "weakest_effect")
nsca_extract(fit, param = "nec:Ceiling accuracy")
nsca_extract(fit, param = "suf:Effect size")
```

nsca_joint	<i>Joint strength of the two components</i>
------------	---

Description

Combines a necessity effect size and a sufficiency effect size into one joint index, with an explicit choice of how much the stronger component may compensate for the weaker one.

Usage

```
nsca_joint(d_nec, d_suf, p = 0, normalize = TRUE)
```

Arguments

d_nec, d_suf	Necessity and sufficiency effect sizes. Vectors are recycled to a common length.
p	Degree of the power mean, which sets the degree of compensation. Defaults to 0, the geometric mean.
normalize	Whether to multiply by two so that the index spans $[0, 1]$. This bound relies on $M_p \leq (d_{nec} + d_{suf})/2 \leq 0.5$ and therefore holds only for $p \leq 1$.

Details

The index is the power mean $M_p = ((d_{nec}^p + d_{suf}^p)/2)^{1/p}$, optionally doubled so that it spans $[0, 1]$. The parameter p sets the degree of compensation:

- $p = -\text{Inf}$ $\min(d_{nec}, d_{suf})$. Fully non-compensatory: the index follows the weaker component only, so additional strength on the other side does not raise it. This is the `weakest_effect` column of `nsca_table()`, reported there unnormalized so that it stays on the components' own scale.
- $p = -1$ The harmonic mean. Less compensatory than the geometric mean, and zero whenever either component is zero.
- $p = 0$ The geometric mean. Partially compensatory: a larger component raises the index, but at a diminishing rate, and the index is zero whenever either component is zero. This is the `balanced_joint_effect` column of `nsca_table()`.
- $p = 1$ The arithmetic mean, that is, half the sum. Fully compensatory: one strong component alone can produce a high index, so a necessary-only relation can score as highly as a necessary-and-sufficient one. For this reason the sum is less suited to summarizing a conjunction, although it belongs to the same family rather than being a different kind of quantity.

The geometric mean is an intermediate choice, and describing it as a non-compensatory "AND" would overstate its properties. It penalizes asymmetry: at an equal sum of 0.80, (0.40, 0.40) gives 0.80 and (0.70, 0.10) gives 0.529.

No magnitude benchmarks are supplied. Conventions for a single NCA effect size do not carry over, because this index has a different scale and a different null: under independence both components carry a positive finite-sample bias that a product of the two amplifies rather than cancels, and its size depends on n , the frontier technique and the scope. We suggest calibrating by simulation on the design at hand before attaching verbal labels to values.

Value

A numeric vector, NA where either component is missing or negative.

See Also

[nsca_table\(\)](#), [nsca_analysis\(\)](#)

Examples

```
nsca_joint(0.40, 0.40)           # balanced
nsca_joint(0.70, 0.10)           # same sum, penalized for asymmetry
nsca_joint(0.40, 0.40, p = -Inf) # the minimum, normalized
nsca_joint(0.40, 0.40, p = -Inf, normalize = FALSE) # weakest_effect
```

nsca_legacy_names	<i>Names retired in NSCA 0.3.0, 0.4.0 and 0.5.x</i>
-------------------	---

Description

The 0.3.0 vocabulary followed the geometry rather than the interpretation: the region between the two frontiers was named for what it is, and the two joint indices for how they treat a weak component rather than for a verdict.

Usage

```
nsca_legacy_names()
```

Details

0.4.0 adopted the vocabulary of *condition analysis in degree*. The region compatible with both components is the **admissible region**; the distance on the X scale between the two thresholds at one outcome target is the **necessity-sufficiency interval**; evidence for both components is **joint support**; and *boundary* is reserved for the theoretical line that separates an expected empty space, so the argument controlling the strictness of a reported inequality became *inequality*.

0.5.x replaced the shorthand "empty zone" with the framework's "empty space", renamed `joint_support_status` to `significance_status` because it considers only the two p-values, and named the technique column of the joint tables `frontier`, as in the component tables, so that "ceiling" refers only to the boundary type.

Retired column names remain available. `nsca_table()` and `nsca_thresholds()` append them as duplicate columns when called with `legacy = TRUE`, each an exact copy of the current column. `nsca_extract()` no longer accepts them: it stops with an error that names the replacement. The retired argument boundary was removed in 0.5.0; its replacement is inequality.

Value

A named character vector: names are the retired names, values the current ones.

See Also

`nsca_terms()`

Examples

```
nsca_legacy_names()
```

nsca_locationshift	<i>Location-shift diagnostic for both frontiers</i>
--------------------	---

Description

Places the ordinary least-squares slope, the median (quantile regression at $\tau = 0.5$) slope and the fitted frontier slope side by side, once for each component.

Usage

```
nsca_locationshift(model, x = NULL, ceiling = NULL, tolerance = 0.1)
```

Arguments

model	An object returned by <code>nsca_analysis()</code> .
x	Optional condition names or positions. All conditions by default.
ceiling	Frontier technique. Defaults to the first technique estimated.
tolerance	Relative spread among the three slopes at or below which they are reported as agreeing.

Details

When all three agree, the frontier is a *location shift*: the central tendency has shifted as a whole to one edge of the data, so the component adds a level to what the regression already describes but does not add a slope. When the frontier slope departs from the other two, the boundary has a shape of its own and the relationship is not a shift.

The two components can give different answers, and may frequently do so. A band whose upper edge tracks the regression while its lower edge does not describes a relationship with a floor of its own shape and a ceiling that is only the mean shifted upward, and this asymmetry is not visible in a table with one row per condition.

This is a natural follow-up to configuration II of `nsca_compare()`, in which the average slope is flat but an excluded corner is empty.

Only frontier techniques that fit a line report a slope. A step frontier (`ce_fdh`, `ce_vrs`) has none by construction, so `frontier_slope` is NA for those techniques and the comparison requires a regression-type frontier such as `cr_fdh`.

Value

A data frame with one row per condition and component.

See Also

`nsca_compare()`, `SCAtools::sca_locationshift()`

Examples

```
set.seed(1)
x <- sort(runif(60))
dat <- data.frame(X = x, Y = pmin(pmax(x + rnorm(60, 0, 0.1), 0), 1))
fit <- nsca_analysis(dat, "X", "Y", ceilings = "cr_fdh")
nsca_locationshift(fit)
```

nsca_magnitude	<i>Conventional magnitude bands for a component effect size</i>
----------------	---

Description

This function calls `SCAtools::sca_magnitude()`. It is provided here so that NSCA tables can be interpreted without attaching the other package.

Usage

```
nsca_magnitude(value = NULL, scale = c("effect_size", "beta"))
```

Arguments

- value Optional numeric vector. If omitted, the band table is returned.
- scale The convention to use: "effect_size" for Dul's bands on a component effect size, "beta" for Cohen's bands on the standardized coefficient that `nsca_compare()` reports. Both use .10, .30 and .50 as edges, with the labels shifted by one band, which is why `beta_cut` and `d_cut` default to different numbers.

Value

With `value`, a character vector of labels. Without it, a data frame of bands.

Which numbers this applies to

`d_nec` and `d_suf`, and `effect_size` in the component tables, under the default `scale = "effect_size"`. These are ordinary NCA effect sizes, and Dul's bands are a published convention that can be cited for them. `scale = "beta"` labels the standardized coefficient that `nsca_compare()` reports beside them, following Cohen's convention. The two conventions share their edges but shift their labels by one band, so a value of 0.35 is labeled "medium" as a coefficient and "large" as an effect size.

The bands do not apply to `weakest_effect`, `balanced_joint_effect` or `joint_empty_space_coverage`. Those are this package's own constructions; no convention has been published for them, and bands supplied by the same package that defines the indices would lack an independent basis. The `print()` method states the same, and `nsca_power()` allows a design to be calibrated by simulation instead.

Read the band beside the floor

A band label indicates where a value lies on a conventional scale. It does not indicate how much of that value the geometry would produce under random pairing alone. `d_nec_null` and `d_suf_null` from `nsca_table()` address that question, and we suggest reporting the two together: a `d_nec` of 0.10 labeled "medium" at a floor of 0.055 is a different finding from the same 0.10 at a floor of 0.007.

References

- Cohen, J. (1988). *Statistical Power Analysis for the Behavioral Sciences* (2nd ed.). Lawrence Erlbaum.
- Dul, J. (2016). Necessary Condition Analysis (NCA): Logic and methodology of "necessary but not sufficient" causality. *Organizational Research Methods*, 19(1), 10-52. doi:10.1177/1094428115584005

See Also

`nsca_table()`, `nsca_compare()`, `SCAtools::sca_magnitude()`

Examples

```
nsca_magnitude()
nsca_magnitude(c(0.04, 0.18, 0.35, 0.62))
nsca_magnitude(c(0.04, 0.18, 0.35, 0.62), scale = "beta")
```

<code>nsca_normalize</code>	<i>Normalize numeric variables for NSCA</i>
-----------------------------	---

Description

Delegated to `SCAtools::sca_normalize()`, which wraps the engine's own utility. It is exported from this package so that an analysis does not need to attach the other package to rescale its inputs.

Usage

```
nsca_normalize(data, scale = NULL, min_max = NULL)
```

Arguments

<code>data</code>	Input data frame or matrix.
<code>scale</code>	Output minimum-maximum pair, or one pair per column.
<code>min_max</code>	Optional empirical or theoretical minimum-maximum pair, or one pair per column.

Details

Both components of an NSCA model are fitted to the same two columns, so a transformation applied here applies to both. The effect sizes are not invariant to a monotone rescaling of either variable: normalizing changes the values, and it is therefore advisable to report which normalization was used.

Value

Data on the requested scales.

See Also

[SCAtools::sca_normalize\(\)](#), [nsca_rescale\(\)](#)

Examples

```
dat <- data.frame(X = c(0, 2, 4, 10), Y = c(10, 20, 30, 50))
nsca_normalize(dat)
```

nsca_outliers

Influential observations on both sides

Description

Identifies the individual observations that support the necessity frontier and those that support the sufficiency frontier. This is the counterpart of [SCAtools::sca_outliers\(\)](#), which addresses the question for one corner.

Usage

```
nsca_outliers(
  data,
  x = NULL,
  y = NULL,
  direction = "HH",
  ceiling = NULL,
  scope = NULL,
  k = 1,
  min.dif = 0.01,
  max.results = 25,
  condensed = FALSE
)
```

Arguments

<code>data</code>	A fitted <code>nsca_analysis()</code> model, or a data frame.
<code>x</code>	With a model, optional condition names or positions; all conditions by default. With a data frame, one condition column name.
<code>y</code>	Outcome column name. Taken from the model when one is supplied; supplying it as well is an error.
<code>direction</code>	One of "HH", "LH", "HL", "LL". Taken from the model when one is supplied; supplying it as well is an error.
<code>ceiling</code>	Frontier technique, passed to <code>NCA::nca_outliers()</code> . With a model, defaults to the first technique that the model estimated.
<code>scope</code>	Optional theoretical scope <code>c(xmin, xmax, ymin, ymax)</code> . Taken from the model when one is supplied; supplying it as well is an error.
<code>k</code>	Number of observations that may be removed at once.
<code>min.dif</code>	Smallest change in the effect size to report.
<code>max.results</code>	Maximum number of rows to keep per side.
<code>condensed</code>	Passed to <code>NCA::nca_outliers()</code> .

Details

A necessary-and-sufficient claim rests on two frontiers simultaneously, and they are supported by different observations: the necessity frontier by the cases at the top-left of the cloud, the sufficiency frontier by the cases at the bottom-right. A case that is influential for one side is often not influential for the other, so a single list would obscure which claim is fragile. The component column identifies the side.

Value

A data frame whose first columns name the condition and the component, with one block per side, or NULL when neither side has anything to report. When only one side does, only that side is returned.

Two ways to call it

When a fitted model is passed, the conditions, outcome, direction and scope are taken from it, so that the diagnostic cannot describe a different analysis from the one being reported:

```
fit <- nsca_analysis(data, x = c("A", "B"), y = "Y", direction = "HH")
nsca_outliers(fit)
```

Every condition in the model is examined unless `x` selects a subset. Passing `y`, `direction` or `scope` alongside a model is an error rather than an override: the model already fixes all three, and a second specification could place the two components on different corners, which would invalidate a two-sided table.

A data frame may be passed instead, as in earlier versions, when no fitted model is available.

Reading it

The output is best interpreted as a fragility check rather than as a data-cleaning step. An observation that moves a frontier behaves as the method expects, since frontiers are determined by extreme cases rather than by central tendency, so identifying one is not in itself grounds for removing it. The result indicates how much of the claim rests on how few cases, which is worth reporting in either case.

See Also

[SCAtools::sca_outliers\(\)](#), [nsca_corners\(\)](#)

Examples

```
set.seed(1)
x <- sort(runif(60))
dat <- data.frame(X = x, Y = pmin(pmax(x + rnorm(60, 0, 0.1), 0), 1))
fit <- nsca_analysis(dat, "X", "Y", ceilings = "ce_fdh")
nsca_outliers(fit)
```

nsca_output

Display selected NSCA output

Description

Prints the tables and draws the plots that are typically needed to report a necessary-and-sufficient analysis, in a single call. This is the counterpart of [SCAtools::sca_output\(\)](#).

Usage

```
nsca_output(
  model,
  summaries = TRUE,
  thresholds = FALSE,
  plots = TRUE,
  tests = FALSE,
  selection = NULL,
  ceiling = NULL,
  scale = NULL,
  outcome_scale = NULL,
  convention = NULL,
  inequality = c("strict", "inclusive")
)
```

Arguments

model	An object returned by nsca_analysis() .
summaries	Whether to print the three result layers.
thresholds	Whether to print the dual-threshold table.

plots	Whether to draw the frontier plots.
tests	Whether to draw the permutation-test plots. Both components are drawn in one figure by <code>nsca_test_plot()</code> , <code>NCA::nca_output()</code> adds the engine's text on the p-accuracy, and the shared sequence, when one was run, is drawn by <code>nsca_shared_plot()</code> .
selection	Optional condition names or positions.
ceiling	Frontier technique for the threshold and test output. Defaults to the first technique estimated.
scale, outcome_scale, convention	Optional reporting overrides passed to <code>nsca_thresholds()</code> .
inequality	"strict" or "inclusive".

Value

Invisibly, a list of the tables and plots produced.

See Also

`nsca_results()`, `nsca_plot()`, `nsca_test_plot()`, `nsca_shared_plot()`

Examples

```
set.seed(1)
x <- sort(runif(60))
dat <- data.frame(X = x, Y = pmin(pmax(x + rnorm(60, 0, 0.1), 0), 1))
fit <- nsca_analysis(dat, "X", "Y", ceilings = "ce_fdh")
nsca_output(fit, thresholds = TRUE)
```

<code>nsca_plot</code>	<i>Plot both frontiers</i>
------------------------	----------------------------

Description

Draws the scatter plot with the necessity and sufficiency frontiers and shades the admissible region between them. Several frontier techniques may be drawn at once. Unlike the computation, a chart involves no numerical comparison, so combining techniques here is appropriate and may be informative.

Usage

```
nsca_plot(model, x = NULL, ceilings = NULL, shade = TRUE, annotate = TRUE)

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

Arguments

model	An object returned by <code>nsca_analysis()</code> .
x	Optional condition names or positions.
ceilings	Frontier techniques to draw. Defaults to all that were estimated.
shade	Whether to shade the admissible region under the first technique.
annotate	Whether to label the two corners that the claim requires to be empty. FALSE removes both labels, which may be preferable for small multiples or for a figure with its own caption.
...	Passed on to <code>nsca_plot()</code> by the <code>plot()</code> method.

Value

A ggplot for one condition, or a named list of plots.

See Also

`nsca_analysis()`, `nsca_table()`

Examples

```
set.seed(1)
x <- sort(runif(60))
dat <- data.frame(X = x, Y = pmin(pmax(x + rnorm(60, 0, 0.1), 0), 1))
fit <- nsca_analysis(dat, "X", "Y", ceilings = c("ce_fdh", "cr_fdh"),
                    reference = "ols")
nsca_plot(fit)
plot(fit)
```

nsca_power

Power for a necessary-and-sufficient claim

Description

Simulates from `nsca_random()` and reports how often each component rejects, how often both reject together, and how far the realized effect sizes differ from the population values.

Usage

```
nsca_power(
  n = c(50, 100, 200),
  direction = "HH",
  d_nec = 0.1,
  d_suf = 0.1,
  slope_nec = 1,
  slope_suf = 1,
  ceiling = "ce_fdh",
```

```

alpha = 0.05,
rep = 50,
test.rep = 100,
seed = NULL,
shared = FALSE
)

```

Arguments

<code>n</code>	Sample sizes to evaluate.
<code>direction</code>	One of "HH", "LH", "HL", "LL".
<code>d_nec, d_suf</code>	Population effect sizes of the necessity and the sufficiency corner, each in $[0, 1]$, passed to <code>nsca_random()</code> . The default of 0.10 on each side matches the default effect size of <code>SCAtools::sca_power()</code> .
<code>slope_nec, slope_suf</code>	Slopes of the two boundary lines on the unit square, as for "HH"; non-negative.
<code>ceiling</code>	Frontier technique.
<code>alpha</code>	Significance level for both component tests.
<code>rep</code>	Replications per sample size.
<code>test.rep</code>	Permutation replications within each fit.
<code>seed</code>	Optional seed. The caller's random stream is restored.
<code>shared</code>	Whether to use one shared permutation sequence of <code>test.rep</code> shuffles per replication, and also to report the power of <code>p_weakest_perm</code> and <code>p_sum_perm</code> .

Value

A data frame with one row per sample size. `power_weakest_perm` and `power_sum_perm` are NA unless `shared = TRUE`.

Describing the design

`d_nec` and `d_suf` are the population effect sizes of the two corners, and `slope_nec` and `slope_suf` the slopes of the two boundary lines on the unit square; see `nsca_random()`. The effect sizes may come from theory, from a pilot study or from comparable studies. A slope estimated in the units of a study corresponds to $\text{slope} * (\text{range of } X) / (\text{range of } Y)$ on the unit square. A power analysis is a planning tool: values estimated from the data of the study itself provide little information about that study's power.

Three power estimates

`power_necessity` and `power_sufficiency` are the rejection rates of the two component tests; `power_joint` is the rate at which the intersection-union test rejects, which requires both. The joint rate is the one for which a necessary-and-sufficient study is powered, and it cannot exceed either of the two component rates.

`power_if_independent` is the product of the two component rates: the joint power the tests would have if they rejected independently. The tests are not independent, since the two frontiers bound

the same cloud and both tests use the same observations, so the product serves only as a reference point whose independence assumption does not hold. `rejection_phi` measures the departure from it directly. It is the correlation of the two rejection indicators across replications: zero under independence, positive when the components tend to reject together, negative when they trade off, and NA when one of them rejected in every replication or in none.

The two supplementary tests

With `shared = TRUE` every replication draws one shared permutation sequence (see `nsca_analysis()`) instead of the engines' two separate runs, all five p-values come from it, and two more rates are reported: `power_weakest_perm` and `power_sum_perm`, how often `p_weakest_perm` and `p_sum_perm` reject. With `shared = FALSE` both columns are NA.

They address the questions that those p-values address (see `nsca_table()`), not the joint claim, and their rates are best interpreted with that in mind. In our simulations with both corners equally empty, both supplementary tests rejected more often than the joint test. With one corner substantially empty and the other with an effect size close to zero, `power_sum_perm` remained high, and `power_weakest_perm` exceeded the rate at which the weaker corner's own test rejected. One largely empty corner can be enough to make the sum large, and the smaller of the two areas is compared with a null distribution that lies below the null of either area alone, since random pairing rarely leaves both corners empty at the same time. These are among the reasons why `joint_support` uses `p_nsca_iut`.

Checking a design in which one side is absent

Every p-value in this package compares the data with random pairing of the same values (see `nsca_table()`). When one corner is substantially empty, random pairing can fill the opposite corner more than the data do, so the test of that corner may reject more often than its nominal level even where the corner is not empty. In our simulations this tendency was noticeable, and its size depended on the strength of the other side, the sample size and the distributions involved. The scripts behind these statements, with their results, are installed in `system.file("simulations", package = "NSCA")`.

Researchers who wish to examine this in their own design can simulate it with one side removed, setting the side under examination to zero and the other to the expected value, for example `nsca_power(n = 60, d_nec = 0.25, d_suf = 0, rep = 500, test.rep = 999)`. In such a design the claim that both sides hold is false, so `power_sufficiency` and `power_joint` (and, with `shared = TRUE`, `power_weakest_perm` and `power_sum_perm`) estimate false-positive rates rather than power, while `power_necessity` remains the power of the side that holds. `d_nec = 0` examines the other side in the same way.

Rates noticeably above alpha suggest that the corresponding p-value is best interpreted together with the effect sizes, the scatter plot and theory rather than in isolation. A possible further step is to find the smaller alpha at which the rate in the side-removed design falls to the intended level; how well that calibration transfers depends on how closely the simulated design resembles the data. The simulated design is a simplification (a uniform condition and an outcome uniform between straight boundaries), so its rates are best read as indications for similar designs.

Precision of the simulated rates

Each rate is a proportion over the replications, so it carries Monte Carlo error of about $\sqrt{\text{rate} * (1 - \text{rate}) / \text{rep}}$: near 0.05 that is about 0.01 with `rep = 500` and 0.007 with `rep = 1000`. `test.rep` sets how finely each p-value is resolved; 999 or more is advisable for a table that will be reported.

The realized effect size exceeds the population one

The two boundary lines are the population frontiers, so `d_nec_declared` and `d_suf_declared` are the true effect sizes. A finite sample does not reproduce them, and the deviation has a systematic direction: the CE-FDH ceiling is the running maximum of the sample, which lies below the population boundary, so the empty region above it is larger than the true one. The same argument applies, in reverse, to the floor. Both empty areas are therefore overstated, and the overstatement decreases as `n` grows: in one simulation with both effect sizes at 0.32, the realized necessity effect size averaged about 0.43 at `n = 40` and 0.37 at `n = 200`.

`d_nec_realized` and `d_suf_realized` report this quantity, so that the size of the bias appears in the same table as the corresponding power.

This is the same overstatement that `nsca_table()` reports as `d_nec_null` and `d_suf_null`, measured differently. There it is obtained from the permutation null on one observed sample, that is, from what random pairing alone produces at that `n`, scope and technique. Here it is the gap between a population of known effect size and what fitting recovers from a finite sample drawn from it. A simulation can compare against a known true value; an analysis cannot, which is why the analysis uses the null instead. In our simulations the two agreed in direction and in order of magnitude, and both decreased as `n` grew.

No claim is made about what this implies for power. The observed statistic is inflated, and so is the permutation null, which the same estimator computes on shuffled data; the extent to which the inflation cancels is an empirical question that this function does not address. The gap is best read as an indication of the estimator's small-sample behavior under this design, and a power computed where the gap is large rests on a statistic that may not yet measure what it measures in larger samples.

Cost

Every replication fits two engines and runs two permutation tests, so the cost is `rep * length(n) * test.rep * 2`. We suggest starting with `rep = 50` and `test.rep = 100` to see the general shape, and increasing both only for the values that will be reported. The shared sequence refits both engines on every shuffle, which takes two to three times as long as the engines' own tests.

The permutation test cannot return a p-value below $1 / (\text{test.rep} + 1)$ with `shared = TRUE`, so with fewer than $1 / \alpha$ permutations (20 at $\alpha = 0.05$) no rejection is possible, and the function warns that the estimated power would then be 0 regardless of the design.

See Also

`nsca_random()`, `SCAtools::sca_power()`

Examples

```
# Kept small for speed: a run intended for reporting uses the defaults and
# takes minutes rather than seconds.
power <- nsca_power(n = c(30, 60), d_nec = 0.25, d_suf = 0.25, rep = 2,
                   test.rep = 39, seed = 1)

power
```

nsca_powerplot

Plot a necessary-and-sufficient power analysis

Description

Draws the three rejection rates that `nsca_power()` reports against sample size. This is the counterpart of `SCAtools::sca_powerplot()`, which draws one curve because a sufficiency analysis has one test.

Usage

```
nsca_powerplot(
  df_power,
  show = c("necessity", "sufficiency", "joint"),
  reference = TRUE,
  target = 0.8
)
```

Arguments

<code>df_power</code>	A data frame from <code>nsca_power()</code> .
<code>show</code>	Which rates to draw: any of "necessity", "sufficiency", "joint", "weakest" and "sum".
<code>reference</code>	Whether to draw <code>power_if_independent</code> as a dashed reference line.
<code>target</code>	Optional horizontal line at a conventional target, 0.80 by default. NULL draws none.

Details

Of the three curves, the joint curve, which is the lowest, is the relevant one for planning. `joint` is the rate at which the intersection-union test rejects, which requires both components to reject, so a study is adequately powered when this curve reaches the target, not when either component curve does. It is drawn with the heaviest line for that reason.

`power_if_independent` is drawn as a dashed reference when `reference` is TRUE. It is the joint power the two tests would have if they rejected independently, which they do not, so the vertical distance between it and the joint curve shows their dependence. When the joint curve lies above it, the components tend to reject together and a study sized on the product would be larger than necessary; when it lies below, they tend to trade off and the product is optimistic.

The curves are subject to the caveats of `nsca_power()`: the simulated design is a simplification, and the realized effect sizes tend to overshoot the population values at small n . The curves are therefore best read alongside `d_nec_realized`, `d_suf_realized` and the declared effect sizes rather than in isolation.

"weakest" and "sum" draw the power of the two supplementary tests, which `nsca_power()` reports with `shared = TRUE`. They address questions other than the joint claim and are drawn only when requested.

Value

A ggplot object.

See Also

`nsca_power()`, `SCAtools::sca_powerplot()`

Examples

```
power <- nsca_power(n = c(30, 60), rep = 2, test.rep = 39, seed = 1)
nsca_powerplot(power)
```

nsca_random

Simulate a necessary-and-sufficient pattern

Description

Draws observations inside a band, so that both corners that a necessary-and-sufficient claim requires to be empty are empty. This is the counterpart of `SCAtools::sca_random()`, which produces a single empty corner because a sufficiency claim requires one.

Usage

```
nsca_random(
  n = 100,
  direction = "HH",
  d_nec = 0.1,
  d_suf = 0.1,
  slope_nec = 1,
  slope_suf = 1,
  names = c("X", "Y")
)
```

Arguments

<code>n</code>	Number of observations.
<code>direction</code>	One of "HH", "LH", "HL", "LL".
<code>d_nec, d_suf</code>	Population effect sizes of the necessity and the sufficiency corner, each in $[0, 1]$.
<code>slope_nec, slope_suf</code>	Slopes of the two boundary lines on the unit square, as for "HH"; non-negative.
<code>names</code>	Column names for the condition and the outcome.

Details

The band is described by what it leaves empty. `d_nec` and `d_suf` are the two population effect sizes: the shares of the unit square left empty in the necessity corner and in the sufficiency corner. Each boundary is a straight line, and since the same area can be left empty by lines of many slopes, `slope_nec` and `slope_suf` specify which line is meant; the intercepts then follow. A slope of 1, the default, is parallel to the diagonal. `d_suf = 0` gives a pattern with no empty sufficiency corner, and `d_nec = 0` one with no empty necessity corner.

The condition is drawn uniformly on $[0, 1]$, and at each value of the condition the outcome is drawn uniformly between the two boundaries. Where a boundary leaves the square, the band is truncated at its edge, so the points are uniform across the band at each value of the condition rather than over its whole area.

All data are generated for "HH" and then reflected. Every direction is "HH" with one or both axes mirrored, so a single generator combined with a reflection ensures that the four directions are exact mirror images of each other. Slopes are therefore given as for "HH", as non-negative numbers; for the other directions the reflection supplies the sign. A slope estimated in the units of a study corresponds to $\text{slope} * (\text{range of } X) / (\text{range of } Y)$ on the unit square.

Value

A data frame with two columns, both on $[0, 1]$.

See Also

`nsca_power()`, `SCAtools::sca_random()`

Examples

```
dat <- nsca_random(n = 60, d_nec = 0.3, d_suf = 0.3)
nsca_analysis(dat, "X", "Y", ceilings = "cefdh")

# A pattern in which only necessity holds
only_nec <- nsca_random(n = 60, d_nec = 0.3, d_suf = 0)
```

nsca_reference	<i>Central-tendency reference lines</i>
----------------	---

Description

Fits the central-tendency lines for each condition, on the same observations both frontiers were fitted to. "ols" is the ordinary least-squares regression; "qr" is a quantile regression at `reference.tau`, which defaults to 0.5 – the median line, a central-tendency summary that is less sensitive to individual observations than the least-squares line.

Usage

```
nsca_reference(model, x = NULL, line = NULL)
```

Arguments

<code>model</code>	An object returned by <code>nsca_analysis()</code> .
<code>x</code>	Optional condition names or positions. All conditions by default.
<code>line</code>	Which reference lines to report. Defaults to those requested when the model was fitted, or to "ols" if none were requested.

Details

Neither is a component of a condition claim. Both describe how the expected outcome changes with the condition; the two frontiers describe which condition-outcome combinations are absent. A reference line therefore has a slope and an intercept but no empty space, no effect size, no permutation p-value and no threshold.

The computation is delegated to `SCAtools::sca_reference()` rather than reimplemented. The sufficiency layer of an NSCA result is an `sca_result` fitted to the same observations, so the line computed there is the line this package would compute, and a single implementation is maintained.

Value

A data frame with one row per condition and reference line.

See Also

`nsca_analysis()`, `SCAtools::sca_reference()`

Examples

```
set.seed(1)
x <- sort(runif(60))
dat <- data.frame(X = x, Y = pmin(pmax(x + rnorm(60, 0, 0.1), 0), 1))
fit <- nsca_analysis(dat, "X", "Y", ceilings = "ce_fdh",
                    reference = c("ols", "qr"))
nsca_reference(fit)
```

nsca_resample	<i>Variability of the joint result across subsamples</i>
---------------	--

Description

Refits one condition on random subsamples of several sizes and reports the distribution of both component effect sizes and both joint indices at each size.

Usage

```
nsca_resample(
  model,
  x = NULL,
  ceiling = NULL,
  sizes = c(0.4, 0.6, 0.8),
  reps = 100,
  seed = NULL
)
```

Arguments

model	An object returned by <code>nsca_analysis()</code> .
x	A condition name or position. Defaults to the first condition.
ceiling	Frontier technique. Defaults to the first technique estimated.
sizes	Subsample sizes. A value at or below 1 is interpreted as a fraction of n, and a value above 1 as a count.
reps	Subsamples drawn per size.
seed	Optional seed. The caller's random stream is restored afterward.

Value

A data frame with one row per subsample size and quantity.

Why four quantities

Both frontiers are fitted to the sample's own extremes, so both components tend to be overstated when the sample is small. The joint indices are built from the two components, so they inherit both overstatements rather than canceling them: `weakest_effect` takes the smaller component and rises with it, and `balanced_joint_effect` is a geometric mean of two upwardly biased values. Resampling only the components would leave this effect unreported.

Why there is no confidence interval

The reasons are those set out in `SCAtools::sca_resample()`. In brief, the ordinary n-out-of-n bootstrap is inconsistent for boundary estimators, because a duplicated extreme cannot extend an envelope further outward; subsampling restores consistency but is biased upward, since a smaller sample tends to leave more of the corner empty by chance; and correcting for that bias requires a convergence rate that depends on the technique and the dimension, which this package does not assume.

Reading it

observed is the full-sample value of the quantity, repeated on every row as a reference point. The spread at a given size indicates how much the value depends on the particular cases that were sampled. The trend across the `m` column indicates how much of the value may be attributable to the small size of the sample.

These results may be compared with `d_nec_null` and `d_suf_null` from `nsca_table()`, which measure the same overstatement differently: by random pairing at the full sample size rather than by refitting at a smaller one.

Cost

Each replication at each size requires a full two-engine refit, including the band reconstruction. The function is therefore noticeably slower than its `SCAtools` counterpart, which is why the default for `reps` is lower.

See Also

`nsca_table()`, `nsca_outliers()`, `SCAtools::sca_resample()`

Examples

```
set.seed(1)
x <- sort(runif(60))
dat <- data.frame(X = x, Y = pmin(pmax(x + rnorm(60, 0, 0.1), 0), 1))
fit <- nsca_analysis(dat, "X", "Y", ceilings = "ce_fdh")
# A report would use the default reps = 100
nsca_resample(fit, reps = 10, seed = 1)
```

nsca_rescale

Convert values to an NSCA reporting scale

Description

The conversion that `nsca_thresholds()` uses, exported so that a quantity obtained elsewhere can be placed on the same scale before it is compared with a reported threshold.

Usage

```
nsca_rescale(
  values,
  reference,
  bounds = range(reference, na.rm = TRUE),
  scale = "percentage.range",
  letter = c("H", "L"),
  convention = c("absolute", "directional"),
  inequality = c("strict", "inclusive")
)
```

Arguments

values	Numeric vector in actual units.
reference	Observed values of the same variable, used by the percentile and sd scales.
bounds	Length-two axis bounds, including any theoretical scope. Defaults to the range of reference.
scale	One of the names returned by nsca_scales() .
letter	"H" or "L", the direction letter for this axis.
convention	"absolute" or "directional".
inequality	"strict" or "inclusive"; affects the percentile scale only.

Details

A necessity threshold on "percentage.range" and a percentage computed elsewhere are comparable only if the latter was converted by the same rule, including the direction-aware mirroring and the percentile strictness that follows the inequality. Both components of an NSCA model share one axis and one direction letter per axis, so one conversion serves both sides.

Delegated to [SCAtools::sca_rescale\(\)](#), which is the function that produced the thresholds being compared against.

Value

A numeric vector on the requested scale. Non-finite inputs return NA.

See Also

[nsca_thresholds\(\)](#), [nsca_scales\(\)](#), [SCAtools::sca_rescale\(\)](#)

Examples

```
x <- c(1, 3, 4, 7, 9)
nsca_rescale(4, x, scale = "percentile", letter = "H")
nsca_rescale(4, x, scale = "percentile", letter = "L")
```

nsca_results	<i>All three NSCA result layers</i>
--------------	-------------------------------------

Description

Returns the necessity analysis, the sufficiency analysis, and their joint necessary-and-sufficient analysis as three explicit tables. The third is not an independent analysis: its `p_nsca_iut` is the conjunction $\max(p_nec, p_suf)$ and can pass only when both component tests pass.

Usage

```
nsca_results(model)

## S3 method for class 'nsca_result'
print(x, ...)

## S3 method for class 'nsca_result'
summary(object, ...)

## S3 method for class 'summary_nsca_result'
print(x, ...)
```

Arguments

<code>model</code>	An object returned by <code>nsca_analysis()</code> .
<code>x</code>	An object returned by <code>nsca_analysis()</code> , for the print method.
<code>...</code>	Additional arguments reserved for methods.
<code>object</code>	An object returned by <code>nsca_analysis()</code> .

Value

A named list containing necessity, sufficiency, and necessary_and_sufficient data frames. The methods `print`, or return a summary object.

See Also

`nsca_table()`

Examples

```
set.seed(1)
x <- sort(runif(60))
dat <- data.frame(X = x, Y = pmin(pmax(x + rnorm(60, 0, 0.1), 0), 1))
# With the two component tests, so that the joint decision can be reached.
# A report would use 1000 or more permutations.
fit <- nsca_analysis(dat, "X", "Y", ceilings = "ce_fdh",
                    test.rep = 99, test.seed = 1)
```

```
layers <- nsca_results(fit)
names(layers)
layers$necessity
fit
summary(fit)
```

nsca_scales*Reporting scales available for NSCA threshold tables*

Description

Lists the scales and conventions understood by [nsca_analysis\(\)](#) and [nsca_thresholds\(\)](#), with a note on how each behaves when the direction refers to a low level.

Usage

```
nsca_scales()
```

Details

Delegated to [SCAtools::sca_scales\(\)](#). Both thresholds in an NSCA table are converted by that package's rule, so the list of scales is that package's list; restating it here would create a second copy that would need to be kept consistent.

Value

A data frame describing each scale.

See Also

[nsca_thresholds\(\)](#), [nsca_rescale\(\)](#), [SCAtools::sca_scales\(\)](#)

Examples

```
nsca_scales()
```

nsca_scope

The scope by which each component effect size is divided

Description

Both components of an NSCA result are empty areas divided by the area of the X-Y scope, so the scope is the denominator of every quantity the analysis reports, including the joint indices. It is not necessarily the scope that was requested: the NCA engine widens a declared scope until it contains every observation, which changes the denominator without changing the scope the researcher declared.

Usage

```
nsca_scope(model)
```

Arguments

`model` An object returned by [nsca_analysis\(\)](#).

Details

This function reports both. `declared_*` is NA when no scope was declared and the observed range was used. `used_*` is the scope by which the engines divided, and `adjusted` is TRUE for a condition on which the two differ. This is the counterpart of [SCAtools::sca_scope\(\)](#).

Value

A data frame with one row per condition.

See Also

[nsca_table\(\)](#), [nsca_analysis\(\)](#), [SCAtools::sca_scope\(\)](#)

Examples

```
set.seed(1)
x <- sort(runif(60))
dat <- data.frame(X = x, Y = pmin(pmax(x + rnorm(60, 0, 0.1), 0), 1))
fit <- nsca_analysis(dat, "X", "Y", ceilings = "ce_fdh",
                    scope = c(0, 1, 0, 1))
nsca_scope(fit)
```

nsca_scope_sensitivity

Sensitivity of the joint result to the scope

Description

Refits one condition on a series of scopes and reports both components and both joint indices for each. This is the counterpart of `SCAtools::sca_scope_sensitivity()`; the scope has a greater influence in a two-sided analysis, as described below.

Usage

```
nsca_scope_sensitivity(
  model,
  x = NULL,
  ceiling = NULL,
  widen = c(0, 0.25, 0.5, 1),
  scopes = NULL
)
```

Arguments

model	An object returned by <code>nsca_analysis()</code> .
x	A condition name or position. Defaults to the first condition.
ceiling	Frontier technique. Defaults to the first technique estimated.
widen	Fractions of the observed span to add to each side of each axis. 0 is the observed range.
scopes	Optional list of explicit <code>c(xmin, xmax, ymin, ymax)</code> boxes, used instead of widen when specific theoretical scopes are of interest.

Value

A data frame with one row per scope.

Why the scope matters more in a two-sided analysis

In a one-sided analysis the scope is the denominator of one effect size. In a two-sided analysis it is the denominator of both, and therefore of every index built from them.

The direction of the change may be contrary to expectation. Widening the box does not dilute the empty areas: the frontiers remain where the data are, so every square unit added beyond the cloud falls on the empty side of one frontier or the other. The empty areas grow faster than the box, `d_nec` and `d_suf` both rise, `weakest_effect` and `balanced_joint_effect` rise with them, and `admissible_region_share` falls toward 0. In the limit the data occupy a very small part of a large box, and almost all of that box is excluded by one claim or the other.

The same data and the same two frontiers can therefore be described as a loosely bounded relationship or as an almost entirely constrained one, depending on a box that was chosen rather than

estimated. This table allows researchers to assess which of these descriptions their own result is closer to.

Interpreting geometry_acceptable and the criterion that failed

The flag can become FALSE at modest amounts of widening, and the reason may differ from the one first assumed. On one 80-case sample:

widen	d_nec	d_suf	overlap_share	reconstruction_error	geometry
0.00	.353	.471	.00008	.00005	TRUE
0.25	.462	.515	.00003	.05555	FALSE
0.50	.526	.555	.00002	.12502	FALSE
1.00	.595	.608	.00001	.22221	FALSE

The two frontiers do not cross: overlap_share decreases, to one part in a hundred thousand. The criterion that fails is reconstruction_error, against its 0.02 tolerance. Outside the observed range of X there are no data to determine a step frontier, so the engines' extension of it and this package's own reconstruction diverge, and the identity $1 - d_{nec} - d_{suf} + \text{overlap} = \text{admissible}$ no longer holds. Detecting this is the purpose of reconstruction_error.

A flagged row should therefore be regarded as unverified rather than as incorrect: the package can no longer confirm that the areas it reports describe the frontiers the engines fitted. Both diagnostic columns are included in the table so that the reader can see which criterion was not met, rather than interpreting a geometry failure as whichever failure was expected.

What the boxes are

By default, each box is the observed range widened by a fraction of the observed span on each side, because such a quantity is easy for a reader to judge: "a box half as wide again as the data" is more interpretable than a pair of numbers. With the default widen, the first row is the observed range itself, which is the box the engines use when no scope is declared and the smallest box that contains the data, since any smaller box is widened back to it.

When the model was fitted on a declared scope, that box is added as a final row labeled "as fitted (declared)", so that the user can see where the chosen scope lies among the alternatives without having to judge it by eye.

Why no derived share is reported

No column reports a change from a baseline, because the choice of baseline depends on the user's question. Taking the observed range as the baseline addresses how much of the result is attributable to the box, counting from the smallest box that contains the data; taking the scope on which the model was fitted addresses how sensitive the result is to the chosen box. These are different questions, and the table does not choose between them. The data do not change between rows, so every difference within a column reflects the box.

See Also

`nsca_scope()`, `nsca_table()`, `SCAtools::sca_scope_sensitivity()`

Examples

```
set.seed(1)
x <- sort(runif(60))
dat <- data.frame(X = x, Y = pmin(pmax(x + rnorm(60, 0, 0.1), 0), 1))
fit <- nsca_analysis(dat, "X", "Y", ceilings = "ce_fdh")
nsca_scope_sensitivity(fit)
```

nsca_shared_plot	<i>Plot the shared permutation null</i>
------------------	---

Description

Draws the null distribution of each component effect size, of their minimum and of their sum, from the single shuffle sequence `shared.test.rep` produces. Each panel marks the observed value (solid line) and the critical value (dashed line), that is, the $1 - \alpha$ quantile that the observed value needs to exceed for the test to reject.

Usage

```
nsca_shared_plot(
  model,
  x = NULL,
  ceiling = NULL,
  components = c("necessity", "sufficiency", "weakest", "sum"),
  bins = 30
)
```

Arguments

<code>model</code>	An object returned by <code>nsca_analysis()</code> with <code>shared.test.rep > 0</code> .
<code>x</code>	A condition name or position. Defaults to the first condition.
<code>ceiling</code>	Frontier technique. Defaults to the first technique estimated.
<code>components</code>	Which panels to draw: any of "necessity", "sufficiency", "weakest" and "sum".
<code>bins</code>	Number of histogram bins.

Value

A ggplot object.

Why this plot needs the shared sequence

The engines' own permutation runs shuffle the outcome twice, independently, so the pairs they produce are drawn from a product measure rather than from the joint law. This does not affect `p_nec` and `p_suf`, whose marginals are unchanged, but it invalidates any statistic that is a function of both. The two empty spaces are disjoint subsets of one scope box, so `d_nec + d_suf` is bounded and the two cannot both be large at the same time. Pairing a large draw of one with a large draw of the other, taken from different shuffles, yields combinations that no dataset could generate.

`weakest_effect` and `component_sum` are functions of both components. Their panels are therefore drawn only from `attr(model, "nsca")$shared`, and this function stops with an error when that sequence is absent rather than assembling a substitute from the engines' separate draws. Such a substitute may appear plausible, but it would place its upper tail too far to the right.

See Also

```
nsca_analysis(), nsca_table(), SCAtools::sca_test_plot()
```

Examples

```
set.seed(1)
x <- sort(runif(60))
dat <- data.frame(X = x, Y = pmin(pmax(x + rnorm(60, 0, 0.1), 0), 1))
# A report would use 1000 or more replications.
fit <- nsca_analysis(dat, "X", "Y", ceilings = "ce_fdh",
                    shared.test.rep = 49, shared.seed = 1)
nsca_shared_plot(fit)
```

nsca_table	<i>Joint necessity and sufficiency results</i>
------------	--

Description

One row per condition and frontier technique. Both sides of a row use the same technique, the same observations and the same scope.

Usage

```
nsca_table(model, legacy = FALSE, p.adjust = "none")
```

Arguments

- `model` An object returned by `nsca_analysis()`.
- `legacy` Whether to append the column names retired in 0.3.0, 0.4.0 and 0.5.x as duplicates. See `nsca_legacy_names()`.

p.adjust Multiple-testing correction applied across the conditions within each frontier technique, as a method name understood by `stats::p.adjust()`. "none", the default, leaves the table unchanged.

Any other method adjusts each component family separately and rebuilds the joint p-value from the adjusted pair, so `p_nsca_iut_adjusted` is `max(p_nec_adjusted, p_suf_adjusted)` rather than `p.adjust(p_nsca_iut)`. These are different numbers, and only the first agrees with the component columns printed beside it. Every significance and support column is then re-derived; the relevance half of `necessity_supported` and `sufficiency_supported` is unchanged, since an effect size does not change when more tests are run. The `*_critical` columns are not adjusted; see the section on multiple testing.

Value

A data frame of joint results.

What is primary

The component effect sizes `d_nec` and `d_suf` are primary. The remaining indices combine them, and three such combinations are reported rather than one, because the geometry of two empty zones does not single out any one of them.

weakest_effect `min(d_nec, d_suf)`, the fully non-compensatory summary. It is zero whenever either component is zero and remains on the components' own scale, so its maximum is about 0.5 rather than 1.

balanced_joint_effect `2 * sqrt(d_nec * d_suf)`, normalized onto `[0, 1]`. Partially compensatory: a larger component raises it at a diminishing rate, and it is zero whenever either component is zero. At an equal component sum of 0.80, `(0.40, 0.40)` gives 0.80 and `(0.70, 0.10)` gives 0.529. It is not a measure of balance: `(0.50, 0.125)` and `(0.25, 0.25)` both score 0.50. The balance column measures balance separately.

joint_empty_space_coverage The share of the scope that at least one claim rules out, `1 - admissible_region_share`. Curvature-neutral: it reaches 1 whenever the admissible region closes, regardless of the shape of the relation.

`min` and the geometric mean are the power means at $p = -\text{Inf}$ and $p = 0$, so choosing between them amounts to choosing how much compensation to allow; the component sum corresponds to $p = 1$, the fully compensatory end, and is less suited to summarizing a conjunction because one strong component can dominate it. `nsca_joint()` exposes the family directly.

The area identity

The two empty spaces and the admissible region tile the scope, so by inclusion and exclusion

$$\text{admissible_region_share} = 1 - d_{\text{nec}} - d_{\text{suf}} + O$$

where O is `overlap_share`. The remaining properties follow from this identity rather than being asserted separately. Because `admissible_region_share` is a share of the scope, it lies in `[0, 1]` unconditionally, and therefore so does `joint_empty_space_coverage`. The other two indices are bounded only up to the overlap:

$$d_{nec} + d_{suf} \leq 1 + O, \quad \text{weakest_effect} \leq 0.5 + O/2, \quad \text{balanced_joint_effect} \leq 1 + O$$

Both bounds are tight and are attained exactly by a perfect diagonal, where O is $1 / (n - 1)$: at $n = 40$ the observed values are 0.5128 and 1.0256 against bounds of 0.5128 and 1.0256. The excess reflects discretization rather than evidence, and the indices are left unclamped so that it remains visible.

Vocabulary

The column names follow the condition analysis in degree framework. `relationship` states the direction as a claim about the theorized X-Y relationship, as in Table 1 of that framework, not about the shape of the estimated frontier. `necessity_empty_space` and `sufficiency_empty_space` name what each expected empty space would contain, and `necessity_boundary` and `sufficiency_boundary` state whether the boundary separating it is a ceiling, which limits how high the outcome can be for a given condition value, or a floor, which limits how low. One of the two is a ceiling and the other a floor, because the two corners are diagonally opposite.

The disjointness check

Whenever the two frontiers do not cross, the floor lies at or below the ceiling throughout the scope, so no point belongs to both expected empty spaces. Because each component effect size is that space's share of the same scope, the two shares cannot sum to more than one. Component effect sizes that sum to more than one therefore indicate that the frontiers have crossed and that the joint result requires diagnosis before interpretation, not that it is unusually strong. `component_sum` reports $d_{nec} + d_{suf}$ so that this rule can be applied directly.

`frontiers_crossed` addresses the same question from the measurement rather than from the sum. The sum serves as a proxy when the crossing itself is not available; here it is available, because `overlap_share` is the area on which the two empty spaces overlap, integrated from the reconstructed frontiers. Using the measurement avoids a limitation of the proxy: the sum and the allowance below are computed by different routes and can disagree in the last few digits on precisely the configuration the check is intended to accept.

That allowance exists because a step frontier overlaps itself by construction. Between two consecutive observations the upper staircase still holds the earlier value while the lower one has already moved, so the two spaces overlap on every tread; summed over the $n - 1$ treads this is exactly $1 / (n - 1)$ of the scope, attained by a perfect diagonal, which is an ideal relation rather than a failure. For every smooth frontier the allowance is zero, and any overlap indicates a crossing.

`frontiers_crossed` asks whether the frontiers crossed at all; `geometry_acceptable` asks whether they crossed by more than `geometry.max_overlap`. A row can be flagged as crossed and still have acceptable geometry.

Geometry diagnostics

`reconstruction_error` is the residual of the identity above. The two sides come from different sources: `admissible_region_share` and `overlap_share` are integrated from the frontiers this package reconstructs, while d_{nec} and d_{suf} are reported by the engines from their own internal

frontiers. A residual near zero indicates that the two agree; a large one indicates that the reconstruction differs from what the engine fitted. A plot may not reveal this, because an incorrectly reconstructed frontier still looks like a frontier.

degenerate marks a condition whose axis does not vary inside the declared scope. geometry_acceptable combines both diagnostics with the excess frontier overlap, allowing a step frontier the $1 / (n - 1)$ it produces by construction.

Decisions

Each p-value addresses a different question, and joint_support is based on the first of them.

- `p_nsca_iut` is $\max(p_{nec}, p_{suf})$: whether each of the two corners is emptier than random pairing produces. It passes only when both components reject at `test.p_threshold`. Against random pairing it is level-alpha without any assumption that the two component tests are independent (the intersection-union rule), so no correction is needed merely for combining two pre-specified tests. It is typically conservative, so an NSCA design tends to require more observations than either component alone. Its reliability depends on that of the component tests it combines; ?NSCA describes a situation in which one of them may reject too readily.
- `p_weakest_perm` asks whether the smaller of the two empty areas, `weakest_effect`, is larger than random pairing produces. Its null is random pairing of both sides at once, so it is calibrated against that, and not against a state in which one component holds and the other does not. It concerns the smaller area, whereas `p_nsca_iut` concerns the weaker evidence, so the two can disagree when one side's random-pairing floor is high. On one shared sequence it is at most the p-value of the component whose observed area is the smaller, so it can reject where that component's own test does not; it is therefore not suited to serve as evidence that both corners are empty.
- `p_sum_perm` asks whether the two corners together, `component_sum = d_nec + d_suf`, are emptier than random pairing produces. A single strongly empty corner can be enough to make it small, so it describes the two corners jointly and complements `p_nsca_iut`, which asks the same of each corner separately. `joint_empty_space_coverage` differs from `component_sum` only by `overlap_share`, which is zero unless the two frontiers cross.

`p_weakest_perm` and `p_sum_perm` require `shared.test.rep`, because the null distribution of a statistic involving both components depends on how the two co-vary under one shuffle. We suggest choosing the p-value that corresponds to the research question before examining the results; reporting several p-values as findings raises a multiple-testing question (see `p.adjust`). Multiplicity across several conditions or frontiers is a separate matter.

`p_source` reports whether the p-values came from the engines' own separate runs or from the shared sequence.

Null distributions

Each null summary in a row is taken from the distribution that produced its p-value: the shared sequence when there is one, otherwise each engine's own test. `d_nec_null` and `d_suf_null` are the means: what random pairing alone produces at this sample size, scope and technique. `d_nec_critical`, `d_suf_critical`, `component_sum_critical` and `weakest_effect_critical` are the $1 - \alpha$ quantiles: the value that random pairing exceeds in only a proportion α of permutations, which each statistic needs to exceed for a single test at α to reject. Each statistic

has its own critical value; that of the sum is not the sum of the two. The last two require a shared sequence. Adjustment for multiple testing does not change them; see the section on multiple testing.

`p_nec_accuracy`, `p_suf_accuracy`, `p_weakest_perm_accuracy` and `p_sum_perm_accuracy` give the Monte Carlo margin of each p-value, as NCA reports its p-accuracy: the half-width of an interval at `test.p_confidence`. They are zero when every ordering of the outcome was evaluated, since the p-value is then exact.

`necessity_supported` and `sufficiency_supported` require significance and, when a relevance threshold was pre-specified, an effect at least that large. `joint_support` additionally requires `geometry_acceptable`, so a degenerate or poorly reconstructed row cannot be reported as supported.

Blind spot

All three joint summaries are areas over the declared scope and share one blind spot. A constant outcome at the center of an imposed scope leaves the upper and lower halves both empty, so both components reach 0.5, the admissible region closes, and every index reaches its maximum for data that carry no information. `balanced_joint_effect` is the most affected, since `weakest_effect` at least follows the weaker side once the constant lies off center. The degenerate flag and the permutation tests address this case: permuting a constant outcome reproduces the same geometry each time, so every p-value is 1.

Dul's conventional bands apply to `d_nec` and `d_suf`, which are ordinary NCA effect sizes; `nsca_magnitude()` returns them as a convention to cite rather than as values calibrated for a particular design. We suggest reading a band together with `d_nec_null`, which indicates how much of the value the geometry produces at that sample size, scope and technique.

No benchmarks are supplied for the joint indices. Conventions for a single NCA effect size do not carry over to them: the scales differ, and under independence both components carry a positive finite-sample floor that `balanced_joint_effect` amplifies rather than cancels, by an amount depending on the sample size, the frontier technique and the scope. The joint indices are this package's own constructions, no convention has been published for them, and one supplied here would come from the same package that defines the indices. Calibration by simulation on the design at hand is suggested instead; `nsca_power()` and `nsca_resample()` support this.

Multiple testing

Each condition involves two component tests, combined into one decision by the intersection-union test, so a model with k conditions produces $2k$ component p-values and k joint ones.

Within a condition, the multiplicity is already accounted for. The IUT rejects only when both components reject, so `p_nsca_iut` is the larger of the two p-values and the level of the test is at most α ; the test is conservative, and its rejection rate is usually below the nominal rate.

Across conditions, no adjustment is made by default. If the k joint tests were independent, the probability of at least one false positive at α would be at most $1 - (1 - \alpha)^k$; it is an upper bound because each joint test is itself conservative. The Settings block printed by `print()` reports both points together, since the conservativeness of each joint test alone gives an incomplete picture.

Adjustment is available as an argument rather than applied by default, for the reason given in `SCAtools::sca_table()`: the appropriate correction depends on the design. When the family is known, `p.adjust` applies the correction; it adjusts the two components and rebuilds the joint p-value from them, as described under that argument.

Adjustment changes the p-values and the decisions, not the *_critical columns. These remain the critical values of single tests at alpha: reference lines rather than the threshold an adjusted test needs to exceed, so an effect size above its critical value may still fall short after adjustment. An adjusted counterpart exists only for Bonferroni, as the 1 - alpha / k quantile of each null, which requires many more permutations to estimate than the p-value does; the step-wise methods have none, because the level at which each condition is tested depends on the p-values of the others. The table prints a message to this effect whenever it adjusts.

Adjustment also raises the smallest p-value a condition can reach. A permutation p-value cannot fall below 1 / test.rep ('1 / (shared.test.rep

- 1)from the shared sequence), and for a condition that is the only strong one amongkit is multiplied b
pha' the condition cannot be significant, regardless of the data. The table issues a warning in
that case and states the number of permutations that would be required.

See Also

[nsca_results\(\)](#), [nsca_joint\(\)](#), [nsca_extract\(\)](#), [nsca_legacy_names\(\)](#)

Examples

```
set.seed(1)
x <- sort(runif(60))
dat <- data.frame(X = x, Y = pmin(pmax(x + rnorm(60, 0, 0.1), 0), 1))
fit <- nsca_analysis(dat, "X", "Y", ceilings = "ce_fdh")
nsca_table(fit)

# Column names retired in earlier releases, appended as duplicates
names(nsca_table(fit, legacy = TRUE))
```

nsca_terms	<i>Terminology of condition analysis in degree</i>
------------	--

Description

A machine-readable glossary linking the vocabulary of the condition analysis in degree framework to the names this package uses. Names retired along the way are listed by [nsca_legacy_names\(\)](#).

Usage

```
nsca_terms()
```

Value

A data frame with one row per term.

References

Chan, Y. (in press). *From Necessity to Sufficiency and Necessity-and-Sufficiency: Extending Condition Analysis in Degree*. University of Leeds.

See Also

[nsca_legacy_names\(\)](#), [nsca_direction_map\(\)](#), [nsca_table\(\)](#)

Examples

```
nsca_terms()
```

nsca_test_plot	<i>Plot the two component permutation tests</i>
----------------	---

Description

Draws the null distribution of each component effect size against the observed value: necessity in the upper panel, sufficiency in the lower. This is the counterpart of [SCAtools::sca_test_plot\(\)](#), which draws one panel because a sufficiency analysis has one component.

Usage

```
nsca_test_plot(model, x = NULL, ceiling = NULL, bins = 30)
```

Arguments

model	An object returned by nsca_analysis() with <code>test.rep > 0</code> .
x	A condition name or position. Defaults to the first condition.
ceiling	Frontier technique. Defaults to the first technique estimated.
bins	Number of histogram bins.

Details

Each panel marks the observed effect size (solid line) and the critical effect size (dashed line), the $1 - \alpha$ quantile of that component's null distribution.

These are the engines' own tests, requested through `test.rep`. Each provides a valid null distribution for its own component, and the intersection-union p-value $\max(p_{\text{nec}}, p_{\text{suf}})$ reported in the subtitle is valid whether or not the two were drawn together, because it rejects only when both components reject and each component's test controls its own error rate. These two panels cannot, however, provide a null distribution for a statistic that depends on both components at once, such as $\min(d_{\text{nec}}, d_{\text{suf}})$: the two histograms come from two independent shuffles, so the pairs behind them were not drawn from one joint law. [nsca_shared_plot\(\)](#) serves that purpose and requires `shared.test.rep > 0`.

Value

A ggplot object.

See Also

[nsca_shared_plot\(\)](#), [nsca_plot\(\)](#), [SCAtools::sca_test_plot\(\)](#)

Examples

```

set.seed(1)
x <- sort(runif(60))
dat <- data.frame(X = x, Y = pmin(pmax(x + rnorm(60, 0, 0.1), 0), 1))
# A report would use 1000 or more permutations.
fit <- nsca_analysis(dat, "X", "Y", ceilings = "ce_fdh",
                    test.rep = 99, test.seed = 1)
nsca_test_plot(fit)

```

nsca_thresholds

*Dual thresholds and the necessity-sufficiency interval***Description**

For each outcome level, reports the necessity and sufficiency thresholds and the direction-oriented distance between them.

Usage

```

nsca_thresholds(
  model,
  ceiling = NULL,
  x = NULL,
  scale = NULL,
  outcome_scale = NULL,
  convention = NULL,
  correspondence.tolerance = NULL,
  inequality = c("strict", "inclusive"),
  digits = 6L,
  legacy = FALSE
)

```

Arguments

model	An object returned by nsca_analysis() .
ceiling	A single frontier technique, applied to both sides.
x	Optional condition names or positions.
scale, outcome_scale	Reporting scales, defaulting to those requested in nsca_analysis() . See SCAtools::sca_scales() .
convention	"absolute" or "directional".
correspondence.tolerance	The largest necessity-sufficiency interval, in the units of the condition, that still counts as threshold correspondence. NULL uses the tolerance with which the model was fitted, which is 0 unless one was declared. See nsca_analysis() for the reasons why no substantive default is provided.

	A non-zero value adds two columns, <code>corresponds</code> and <code>correspondence_tolerance</code> , and enables the "near correspondence" category of <code>threshold_status</code> . At zero these columns are omitted, because they would carry no judgment, and a binary verdict placed beside a continuous interval tends to attract more attention than the interval itself. The default output therefore reports the interval; a verdict is added only when a width has been chosen and can be justified.
<code>inequality</code>	Strict ("strict", the default) or inclusive ("inclusive") inequalities in the reported rules.
<code>digits</code>	Significant digits in the formatted text.
<code>legacy</code>	Whether to append the column names retired in 0.3.0, 0.4.0 and 0.5.x as duplicates. See nsca_legacy_names() .

Details

For a fixed outcome target, the *necessity-sufficiency interval* is the distance on the condition axis between the two thresholds: if a target requires at least 40 units of the condition and 70 units are sufficient for it, the interval extends from 40 to 70. It is not a confidence interval and does not express statistical uncertainty.

`necessity_sufficiency_interval` and `overlap_width` are reported on scale, the same scale as the two threshold columns whose distance they measure, so that a row can be checked by subtraction. `necessity_sufficiency_interval_actual` and `overlap_width_actual` report the same two quantities in the units of the data, whatever scale is requested, and `threshold_gap_actual` is the signed gap in actual units from which both are derived.

Under `scale = "percentile"` the reported column is a difference of ranks rather than a distance, so it is interpreted as the share of cases lying between the two thresholds. This is a useful quantity, but it is not a width: equal percentile gaps in a dense and in a sparse part of the distribution correspond to very different distances on the condition axis. For a normalized width that is comparable across studies, `"percentage.range"` or `"sd"` may be more suitable.

The interval is also distinct from the component effect sizes. Effect sizes aggregate exclusion over the whole scope, whereas the interval is evaluated at one outcome target. Large component effects therefore do not guarantee a narrow interval at any particular level, and a narrow interval at one target does not imply anything about the others. Reporting both quantities, at several targets, is not redundant.

Value

A data frame with one row per condition and outcome level.

The three regions

Necessity and sufficiency address different questions about the same outcome level, and together they divide the condition axis into three parts. For a high-X direction:

below `necessity_threshold` The outcome level is out of reach.

between the two thresholds It is admitted but not guaranteed. This is the admissible interval, and `necessity_sufficiency_interval` is its width. It is the horizontal cross-section, at one outcome level, of the region whose area `admissible_region_share` reports.

above sufficiency_threshold The fitted frontier guarantees it.

For a low-X direction these inequalities reverse: values above the necessity threshold are out of reach and values below the sufficiency threshold are guaranteed. `threshold_gap_actual` is direction-oriented, so a positive value indicates an admitted interval in all four directions. A negative value is reported in `threshold_status`, with its magnitude in `overlap_width` and `overlap_width_actual`, rather than being truncated to zero.

The interpretation of a crossing depends on the technique. An exact envelope cannot cross itself: CE-FDH's two frontiers are a running minimum and a running maximum over the same observations, so the floor cannot rise above the ceiling, and a crossing there reflects either the one-tread artifact of a step function or a genuine specification problem. Fitted lines, by contrast, can cross. The line techniques (CR-FDH, CR-VRS, qr, cols) regress a straight line through the frontier peers, and the narrower the band between the two frontiers, the more easily two separately fitted lines cross inside it. A small crossing under a line technique may therefore indicate a band narrower than the fitting error, which is the pattern an almost-perfect necessary-and-sufficient relation would produce. Such a crossing is consistent with strong correspondence rather than with a misspecified model.

The two interpretations are distinguished by magnitude rather than by sign. Within `correspondence.tolerance` the two thresholds are treated as lying at the same X, whichever side the estimate fell on, and corresponds is TRUE. Beyond the tolerance in the negative direction, the floor lies above the ceiling over an interval, which is incompatible with the relation; corresponds is then NA, because the correspondence question cannot be answered there.

The interval collapses to zero exactly where the relation is deterministic at that level, reported as "exact correspondence": one X value is then both the minimum required and the sufficient level. An estimated interval of zero is consistent with exact correspondence but does not by itself establish it, and a small non-zero interval is best described as near correspondence only when a substantively justified tolerance has been defined. A positive interval does not negate joint support; it indicates imperfect threshold correspondence. Neither NCA nor SCA alone can produce this table, since each supplies one edge of the interval.

The two out-of-range markers are interpreted differently on the two sides, because their meaning inverts under contraposition. On the necessity side they retain their necessity interpretation, giving "no_minimum", "no_maximum" and "unattainable"; on the sufficiency side SCAtools supplies the sufficiency interpretation.

See Also

[nsca_table\(\)](#), [nsca_analysis\(\)](#), [nsca_legacy_names\(\)](#)

Examples

```
set.seed(1)
x <- sort(runif(60))
dat <- data.frame(X = x, Y = pmin(pmax(x + rnorm(60, 0, 0.1), 0), 1))
fit <- nsca_analysis(dat, "X", "Y", ceilings = "ce_fdh")
nsca_thresholds(fit)
nsca_thresholds(fit, scale = "actual")
```

Index

NCA::nca_analysis(), 7
NCA::nca_outliers(), 22
NCA::nca_output(), 24
NSCA (NSCA-package), 3
nsca, 6
nsca(), 4
NSCA-package, 3
nsca_analysis, 7
nsca_analysis(), 4–7, 12, 13, 15, 17, 18, 22, 23, 25, 27, 32, 33, 36–39, 41, 42, 48, 49, 51
nsca_compare, 12
nsca_compare(), 3, 4, 19, 20
nsca_corners, 14
nsca_corners(), 11, 15, 23
nsca_direction_map, 14
nsca_direction_map(), 48
nsca_extract, 15
nsca_extract(), 18, 47
nsca_joint, 16
nsca_joint(), 11, 43, 47
nsca_legacy_names, 17
nsca_legacy_names(), 15, 42, 47, 48, 50, 51
nsca_locationshift, 18
nsca_locationshift(), 4, 13
nsca_magnitude, 19
nsca_magnitude(), 46
nsca_normalize, 20
nsca_outliers, 21
nsca_outliers(), 4, 34
nsca_output, 23
nsca_output(), 4
nsca_plot, 24
nsca_plot(), 4, 5, 24, 48
nsca_power, 25
nsca_power(), 3–5, 29–31, 46
nsca_powerplot, 29
nsca_random, 30
nsca_random(), 25, 26, 28
nsca_reference, 32
nsca_reference(), 6, 8, 13
nsca_resample, 33
nsca_resample(), 4, 46
nsca_rescale, 34
nsca_rescale(), 21, 37
nsca_results, 36
nsca_results(), 4, 11, 24, 47
nsca_scales, 37
nsca_scales(), 35
nsca_scope, 38
nsca_scope(), 4, 40
nsca_scope_sensitivity, 39
nsca_scope_sensitivity(), 4
nsca_shared_plot, 41
nsca_shared_plot(), 24, 48
nsca_table, 42
nsca_table(), 3–5, 11, 13, 15–18, 20, 25, 27, 28, 34, 36, 38, 40, 42, 48, 51
nsca_terms, 47
nsca_terms(), 5, 15, 18
nsca_test_plot, 48
nsca_test_plot(), 4, 24
nsca_thresholds, 49
nsca_thresholds(), 3–5, 8, 10, 11, 18, 24, 34, 35, 37
plot.nsca_result (nsca_plot), 24
print(), 46
print.nsca_result (nsca_results), 36
print.summary_nsca_result (nsca_results), 36
SCAtools::sca_analysis(), 7, 8, 10
SCAtools::sca_compare(), 13
SCAtools::sca_locationshift(), 19
SCAtools::sca_magnitude(), 19, 20
SCAtools::sca_normalize(), 20, 21
SCAtools::sca_outliers(), 21, 23
SCAtools::sca_output(), 23

SCAtools::sca_power(), [26](#), [28](#)
SCAtools::sca_powerplot(), [29](#), [30](#)
SCAtools::sca_random(), [30](#), [31](#)
SCAtools::sca_reference(), [32](#)
SCAtools::sca_resample(), [34](#)
SCAtools::sca_rescale(), [35](#)
SCAtools::sca_scales(), [37](#), [49](#)
SCAtools::sca_scope(), [38](#)
SCAtools::sca_scope_sensitivity(), [39](#),
[40](#)
SCAtools::sca_table(), [46](#)
SCAtools::sca_test_plot(), [42](#), [48](#)
stats::p.adjust(), [43](#)
summary.nsca_result(nsca_results), [36](#)