Package: inferguidR (via r-universe)

November 3, 2026

Type Package

Title Guided Inference, Power Analysis and Diagnostic Accuracy

Version 0.0.0.9000

Description Plans and explains statistical analyses. Provides a registry of power-analysis procedures modelled on the test inventory of the 'G*Power' 3.1 program, immutable analysis specifications that are validated against each procedure's modes, inputs and options, and results that always carry a typed status (ok, infeasible, unsupported, invalid input, numerical failure or reference pending) together with provenance and a reproducible R call. Specifications, results, analysis envelopes, advisor recommendations and report snapshots are serialised as canonical JSON with SHA-256 hashes and versioned JSON Schemas, and are imported again without evaluating code. Power calculations, diagnostic-accuracy and classifier-cutoff tools, a guided method advisor and an interactive 'shiny' application are added in stages; a procedure without an implemented engine returns the status 'unsupported' instead of a number. Numerical agreement with 'G*Power' is not claimed until it has been shown against recorded reference cases.

License MIT + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 8.0.0

Depends R (>= 4.1.0)

Imports digest, grDevices, graphics, grid, jsonlite, stats, utils

Suggests bslib, cluster, Exact, exact2x2, geepack, ggplot2, jsonvalidate, knitr, lme4, MASS, MESS, nlme, nnet, pROC, pwr, pwrss, quarto, rmarkdown, shiny, shinytest2, testthat (>= 3.0.0), WebPower, withr, yardstick

Config/testthat/edition 3

Config/testthat/parallel true

URL https://github.com/CTTIR/inferguidR

BugReports https://github.com/CTTIR/inferguidR/issues

VignetteBuilder knitr

Config/Needs/website CTTIR/themakR

Repository https://cttir.r-universe.dev

Date/Publication 2026-10-29 11:21:30 UTC

RemoteUrl https://github.com/CTTIR/inferguidR

RemoteRef HEAD

RemoteSha e91938f77eaa991b69989b121bfd32c124c95db0

Contents

advisor_answer . . . 4
advisor_override . . . 5
advisor_power_spec . . . 6
advisor_question . . . 7
advisor_r_code . . . 8
advisor_recommendation . . . 12
advisor_replay . . . 13
advisor_rules . . . 14
advisor_save . . . 16
advisor_session . . . 17
advisor_set_language . . . 18
advisor_validate_rules . . . 18
analysis_envelope . . . 20
analysis_spec . . . 21
attach_classification_data . . . 22
binormal_model . . . 23
binormal_rates . . . 24
boschloo_compare_fisher . . . 24
boschloo_control . . . 26
boschloo_plotdata_allocation . . . 27
boschloo_plotdata_lattice . . . 28
boschloo_plotdata_nuisance . . . 30
boschloo_plotdata_power_effect . . . 31
boschloo_plotdata_power_n . . . 32
boschloo_power . . . 33
boschloo_probabilities . . . 34
boschloo_region . . . 35
boschloo_sample_size . . . 36
boschloo_test . . . 38
boschloo_variants . . . 39
calculate_power . . . 40
calculator_functions . . . 41
calibration_summary . . . 42

classification_auc . . . 43
classification_evaluated_spec . . . 44
classification_examples . . . 45
classification_metrics . . . 46
classification_plot_data . . . 47
classification_spec . . . 49
classification_table . . . 51
classification_uncertainty . . . 52
classifier_model . . . 54
compare_classifiers . . . 55
compare_designs . . . 56
create_report_snapshot . . . 56
cutoff_rule . . . 58
cutoff_sweep . . . 60
distribution_calculator . . . 61
effect_size . . . 62
evaluate_cutoff . . . 64
export_report . . . 66
format_metric_values . . . 67
ig_config . . . 68
ig_domain . . . 68
ig_from_json . . . 70
ig_hash . . . 71
ig_method . . . 71
ig_option . . . 74
ig_options . . . 75
ig_param . . . 75
ig_params . . . 77
ig_standard_params . . . 77
ig_status . . . 78
ig_status_codes . . . 79
ig_to_json . . . 80
is_ig_status . . . 80
launch_app . . . 81
learning_lesson . . . 82
list_methods . . . 82
metric_dictionary . . . 83
net_benefit . . . 84
plot_classification . . . 85
plot_design_sweep . . . 87
plot_power . . . 87
power_plot_data . . . 88
ppv_gap_max_prevalence . . . 89
pr_summary . . . 90
predictive_values_at_prevalence . . . 91
prevalence_for_ppv_target . . . 92
prevalence_scenario . . . 93
protocol_record . . . 94

protocol_text 95
report_pdf_capabilities 96
select_cutoff 96
solve_design 98
sweep_design 98
update.classification_spec 99
update.power_spec 100
validation_status 101
workbench_app 102
xy_plot_data 102

Index **105**

`advisor_answer` *Answer, change or retract advisor questions*

Description

`advisor_answer()` answers the current question, or changes the answer to a question answered earlier. Changing an earlier answer is backtracking: every answer recorded after it, and any override, are invalidated (listed in `session$invalidated`) and the walkthrough continues from the changed question. Giving the same answer again changes nothing. A question that is neither current nor answered is refused, so only applicable follow-ups can be answered.

Usage

```
advisor_answer(session, answer, question_id = NULL, rules = NULL)

advisor_back(session, to = NULL, rules = NULL)
```

Arguments

`session`	An `advisor_session`.
`answer`	An option id of the question (see `advisor_question()`), for example `"dont_know"`.
`question_id`	The question to answer. `NULL` (default) answers the current question.
`rules`	An `advisor_rules` object, or `NULL` for the bundled rules. It must be the rule set the session was created with.
`to`	Question id to go back to (it must have been answered). `NULL` retracts only the last answer.

Details

`advisor_back()` retracts the last answer, or every answer from question `to` onwards, and makes that question current again.

Value

A new `advisor_session`. Invalid answers raise a condition of class `inferguidR_invalid_input`.

See Also

Other advisor: advisor_override(), advisor_power_spec(), advisor_question(), advisor_r_code(), advisor_recommendation(), advisor_replay(), advisor_rules(), advisor_save(), advisor_session(), advisor_set_language(), advisor_validate_rules()

Examples

```
s <- advisor_session()
s <- advisor_answer(s, "compare")
s <- advisor_answer(s, "analysis")
s <- advisor_answer(s, "describe", question_id = "q.aim") # backtracking
s$invalidated$question_ids
```

`advisor_override` *Replace the recommended variant by a listed alternative*

Description

A user may replace the variant of the outcome only by a variant listed in its alternatives with `override_allowed = TRUE`. The override is recorded with the alternative's why-not text and the status becomes at most `conditional`. Variants listed as rejected routes are refused with the stated reason; any other variant is refused as not compatible with the recorded design (`refusal = "not_listed"`). Listed alternatives must also satisfy their design guards: the F variance-ratio test requires two groups, and the ordered-proportion trend test requires a recorded binary outcome.

Usage

```
advisor_override(session, analysis_variant_id, rules = NULL)

advisor_clear_override(session, rules = NULL)
```

Arguments

`session`	An `advisor_session` that has reached a method outcome.
`analysis_variant_id`	The variant to use instead.
`rules`	An `advisor_rules` object, or `NULL` for the bundled rules. It must be the rule set the session was created with.

Value

A new `advisor_session`. A refusal is a condition of classes `inferguidR_override_refused` and `inferguidR_invalid_input`; its `$status$details$refusal` is one of `"rejected_route"`, `"not_listed"`, `"override_not_allowed"`, `"no_method"`, `"unknown_variant"`, `"incompatible_group_count"` or `"incompatible_outcome_type"`.

See Also

Other advisor: advisor_answer(), advisor_power_spec(), advisor_question(), advisor_r_code(), advisor_recommendation(), advisor_replay(), advisor_rules(), advisor_save(), advisor_session(), advisor_set_language(), advisor_validate_rules()

Examples

```
s <- advisor_replay(c(q.aim = "compare", q.stage = "analysis",
  q.outcome_type = "continuous", q.unit = "individuals",
  q.structure = "independent", q.independence_check = "no",
  q.n_factors = "one", q.n_groups = "two", q.covariates = "no",
  q.estimand_continuous = "mean", q.variance = "not_justified",
  q.direction = "two_sided", q.missingness = "none"))
s2 <- advisor_override(s, "t_means_two_independent_student")
advisor_recommendation(s2)$status
```

---

advisor_power_spec *Create a power specification for an explicitly mapped advisor variant*

Description

Converts a completed advisor session to a power specification without running a calculation. Currently the three Boschloo analysis variants are mapped to the same standalone Boschloo engine: central, minlike and an explicitly prespecified less/greater direction. Other methods and unrelated overrides return a typed unsupported condition. Missing answers must be resolved first. Existing saved sessions require the usual explicit rules-version migration/replay; this function does not relabel them.

Usage

```
advisor_power_spec(
  session,
  inputs,
  mode = "a_priori",
  alternative = NULL,
  allocation = NULL,
  control = boschloo_control(),
  rules = NULL
)
```

Arguments

session	A completed advisor_session.
inputs	Named list with explicit p1, p2, alpha, and either power, allocation_ratio (a priori) or n1, n2 (post hoc). Optional n_max bounds an a priori search. Numerical controls are supplied through control, not duplicated in inputs.
mode	Either "a_priori" or "post_hoc".

alternative	Required "less" or "greater" for a one-sided recommendation; NULL or "two.sided" for a two-sided variant.
allocation	Explicit a priori policy: "n2_ceiling" or "total_nearest". Must be NULL for a fixed post-hoc design.
control	Numerical settings from boschloo_control(). The generic planner supports its default polish_tol, max_levels and ambiguity_band; changed values are refused rather than discarded. Caching does not affect the numerical specification.
rules	Optional matching advisor_rules object.

Details

Group 1 is p1 and group 2 is p2, as in the exported analysis example. alternative = "greater" means p1 > p2; "less" means p1 < p2. The chosen direction is never inferred from the supplied probabilities. The mapping establishes the identity of analysis and planning procedures, not exhaustive numerical certification or agreement with G*Power.

Value

A power_spec for calculate_power() or solve_design(). Retain the session/recommendation alongside the specification to preserve the question trace and scientific assumptions.

See Also

Other advisor: advisor_answer(), advisor_override(), advisor_question(), advisor_r_code(), advisor_recommendation(), advisor_replay(), advisor_rules(), advisor_save(), advisor_session(), advisor_set_language(), advisor_validate_rules()

---

advisor_question *Show an advisor question*

Description

Returns the text of a question in the session language: the question, why it matters, the options (with the worked explanation of 'I do not know' and the assumption it records, if any), the confirmation rules that apply when data are supplied, and the recorded answer.

Usage

```
advisor_question(session, question_id = NULL, language = NULL, rules = NULL)
```

Arguments

session	An advisor_session.
question_id	A question id; NULL (default) gives the current question, or NULL when an outcome has been reached.
language	"en" or "de"; NULL uses the session language.
rules	An advisor_rules object, or NULL for the bundled rules. It must be the rule set the session was created with.

Value

An object of class advisor_question, or NULL.

See Also

Other advisor: advisor_answer(), advisor_override(), advisor_power_spec(), advisor_r_code(), advisor_recommendation(), advisor_replay(), advisor_rules(), advisor_save(), advisor_session(), advisor_set_language(), advisor_validate_rules()

Examples

```
q <- advisor_question(advisor_session(language = "de"))
q$text
q$options[[length(q$options)]]$explanation
```

---

advisor_r_code *Export the selected advisor method as an executable R teaching example*

Description

Returns a bundled example for the exact selected method. The script uses synthetic data and records the recommendation, assumptions and rule version as comments. It does not analyse user data or evaluate code during export. A missing template raises an unsupported condition; no other test is used.

Usage

```
advisor_r_code(session, alternative = NULL, conf.level = 0.95, rules = NULL)
```

Arguments

session	An advisor_session.
alternative	NULL for a recorded two-sided design, or an explicit "greater" or "less" for a recorded pre-specified one-sided design. The meaning of "greater" is stated in the method-specific script (for example x minus y, positive-sign probability, or an odds ratio above one). It must be chosen before inspecting results.

`conf.level` Confidence level strictly between zero and one. This is an explicit example setting, not a value inferred from the advisor answers.

`rules` An `advisor_rules` object, or `NULL` for the bundled rules. It must be the rule set the session was created with.

Details

The bundled examples cover one-sample, Welch, pooled two-sample and paired t-tests; exact binomial and uncorrected score proportion tests; Pearson, Spearman and Kendall correlation; one-sample and paired sign tests; exact conditional and mid-p McNemar tests; fixed-margin Fisher tests; and central, minlike and directional Boschloo tests using the package engine; one-sample/paired signed-rank and independent rank-sum tests; Pearson goodness-of-fit and independence tests, independent-proportion score tests, and corrected/uncorrected asymptotic McNemar tests. Omnibus examples do not implement directional hypotheses. Sparse-cell diagnostics never switch tests. Independent-proportion intervals use unpooled normal variance, while their score test uses pooled variance; decisions need not coincide. Other new chi-squared examples explicitly do not compute confidence intervals. General-table Fisher examples condition on both margins and order tables by probability. They reject directional hypotheses, supply no sampling interval, and never substitute simulation or an approximation after resource failure. Their stated teaching limits are 100 units and five categories per dimension. Descriptive examples require explicit continuous, binary, ordinal or nominal measurement meaning, report mean or marginal category intervals and no p-values. Binary inputs must declare both factor levels, or use logical values and an explicit event. Ordinal quantiles are observed ordered categories (type 1). Descriptive export requires alternative = NULL because it has no hypothesis. Independent-group omnibus examples cover pooled and Welch one-way ANOVA and Kruskal-Wallis, with explicit variance assumptions or tied-rank correction. They do not supply sampling intervals or unplanned pairwise comparisons. Repeated-measures examples cover tied-rank Friedman and binary Cochran Q omnibus tests for three or more occasions, with confirmed unique participant IDs. Missing occasions require an explicit complete-case choice and exclusions are counted; validity requires justified missing-completely-at-random assumptions. Zero conditional variance is unavailable. No intervals or pairwise tests are invented. Paired nominal examples distinguish Stuart-Maxwell marginal homogeneity from Bowker symmetry, require identical named categories on both axes and retain the supplied count of excluded pairs. Singular marginal covariance or zero symmetry pair totals are explicitly unavailable, without category pooling. Linear regression examples use prespecified numeric predictors, an intercept and classical homoscedastic t/F inference. Complete-case exclusion is explicit and counted. Coefficients describe associations. Multiple-regression examples separate the omnibus test from an optional prespecified nested predictor-set test on exactly the same rows. No stepwise selection or robust-error switch is performed. Rank-deficient and numerically exact fits cannot supply inference. Model routes export two-sided inference. The standalone simple-regression example also supports prespecified directional slope tests, separately from the two-sided coefficient table. Correlation-to-regression overrides remain refused. PCA examples form exploratory weighted sums of centered numeric variables, standardised by default. The analyst chooses the retained component count; weights, scores, variance fractions and reconstruction error are reported. Missing-case exclusion is explicit. Repeated eigenvalues identify subspaces rather than unique individual axes. Components are not latent factors. PCA export requires alternative = NULL; conf.level is not used by its script. Fixed-cutoff diagnostic counts examples require declared sampling, rule origin and evaluation split. They retain typed undefined/infinite metrics and marginal Clopper-Pearson intervals, with observed and prevalence-scenario values separate. A tuned cutoff cannot claim validation on its tuning data. Four

aggregate cells do not identify ROC/PR curves, AUC or a cutoff sweep. Independent units and a fixed rule must be justified; no clustering or selection uncertainty is inferred. These examples require alternative = NULL and use conf.level for marginal intervals. Brunner-Munzel examples estimate P(X > Y) plus half the tie probability, using an explicit heteroscedastic placement-variance t approximation. They require at least ten observations per group in this teaching implementation. Zero placement variance is unavailable. Approximate intervals are intersected with `[0,1]` and unconstrained limits retained. This is not a mean/median-shift test. Monte Carlo goodness-of-fit examples use prespecified multinomial probabilities, a recorded seed and replicate count, and restore the calling RNG state. Their plus-one p-value and computational precision interval are distinct from a sampling interval for an effect. Probabilities are normalised only for roundoff within 64 machine epsilons; no fitted probabilities or automatic asymptotic fallback are used. Simulation allocation is limited to ten million cells. Cochran-Armitage trend examples require a recorded binary outcome, named independent-binomial group counts and explicitly confirmed, increasing scores chosen before inspecting outcomes. Score spacing is part of the hypothesis. The public omnibus route exports two-sided inference; the standalone function also supports a prespecified signed direction. No event variation is unavailable, and sparse counts do not trigger an exact or continuity-corrected substitute. No risk-difference or logistic-slope confidence interval is manufactured. Jonckheere-Terpstra examples require a prespecified group order and independent unit IDs, with numeric or explicitly ordered outcomes. Inclusive permutation tails use a recorded seed and plus-one correction; the two-sided p-value is the capped doubled smaller tail. The public route exports two-sided inference; the standalone function accepts a prespecified increasing or decreasing trend. Caller RNG state is restored. Simulation intervals describe computational precision, not population effects. All-equal outcomes have p = 1 without simulation. Limits are 200 rows, 10 groups and ten million pair comparisons. Binormal classifier examples are hypothetical fixed-cutoff scenarios with explicit assumed AUC, standard-deviation ratio, prevalence and count scale. Higher scores are positive, on the scale standardised to non-events. Analytic class-conditional rates are separate from expected-count metrics, including zero-prevalence denominators. Expected N does not validate sample-size planning or supply empirical confidence intervals. Finite cutoffs are required by the manual-rule API. These scripts do not use conf.level and require alternative = NULL. Scored classifier examples require explicit independent-unit identifiers, score direction and comparator. Exploration enumerates decision sets descriptively without intervals; a predeclared cutoff requires an explicit declaration. The held-out example selects Youden J on tuning data with a fewest-alerts tie break, then transfers the fixed numeric rule. Known overlapping IDs are refused, but disjoint IDs do not prove independence. Symbolic alert-none rules without a numeric cutoff cannot transfer. No score-model training or nested resampling is implied. Missing data are rejected. All three require alternative = NULL; only predeclared and held-out scripts use conf.level, for marginal fixed-rule Clopper-Pearson intervals. ANCOVA examples fit a prespecified additive common-slope model to independent units, with one categorical group and numeric covariates. The overall group F test compares nested models fitted to exactly the same retained rows. Adjusted means use pooled covariate means; all pairwise differences are explicitly unadjusted marginal contrasts. Common slopes must be confirmed before fitting; no interaction pretest or causal interpretation is implied. Rank deficiency, constant responses and numerically exact fits are refused. Missing data require an explicit complete-case choice. Public examples match two versus multiple groups; the omnibus test remains nondirectional. Mean confidence intervals are not prediction intervals for individuals. Spread examples cover the normal-sample two-group F variance-ratio test, median-centred Brown-Forsythe (Levene) and Fligner-Killeen tests. The recorded spread route is nondirectional. The F interval concerns the raw population variance ratio; robust omnibus examples do not supply sampling intervals. These examples answer a spread question, never select a mean test by pretesting. Their contracts

reject missing or infinite measurements. Rank correlations explicitly use asymptotic tests. Paired examples require one wide row per confirmed, unique participant identifier. Mean-test intervals concern the raw mean difference (x minus y or a stated reference). Proportion intervals concern the event probability. Pearson intervals use the Fisher transform; rank-correlation intervals are explicitly unavailable in these base R examples. Sign and McNemar intervals concern positive-sign or discordance probabilities, not median or marginal differences. Mid-p examples explicitly use an exact conditional interval, not an inversion of the mid-p test. Fisher intervals concern the conditional odds ratio and inherit base R's numerical root tolerance. Boschloo examples preserve the selected ordering and direction; sampling intervals are explicitly unavailable, and numerical optimisation bounds are labelled separately. Signed-rank examples require interval scale and symmetry. Their intervals concern the pseudomedian. Rank-sum examples report descriptive superiority; a shift interval is available only under an explicitly requested location-shift model, never for ordinal codes. Exact rank examples reject zeros/ties where relevant and samples above their stated teaching limit; approximation requires an explicit choice. Rank calculations retain the supplied numeric precision; signed-rank approximations omit null differences explicitly across R versions. The pooled-Z unconditional example uses its own bounded small-sample implementation, with a zero difference null, fixed group sizes and a confirmed independent binomial design. Total sample size is at most 30. Two-sided inference orders absolute pooled Z. Endpoints are included in the nuisance search; lower and upper numerical bounds are separate from sampling uncertainty. The example refuses non-convergence and withholds a decision when its numerical interval contains the significance threshold. Factorial examples use two or three fixed crossed factors, full interactions, sum contrasts and prespecified type-III coefficient-block F tests. Each declared cell needs two observations; empty cells, repeated units and unjustified equal variance are refused. Raw term p-values and Holm adjustment across all terms are separate. Complete-case exclusion is explicit. Partial eta squared is not an additive fraction of total explained variation in unbalanced designs. Repeated one-way examples apply the Greenhouse-Geisser correction routinely to both F degrees of freedom, without a Mauchly pretest. Three to thirty occasions and at least three complete independent subjects are required. Subject exclusion requires an explicit complete-case choice. Degenerate contrast variance is refused; no post-hoc tests or effect-size interval are computed. The correction is approximate, not an exact covariance-free test. Regression examples use prespecified numeric predictors and explicit independent units. Logistic slope odds ratios receive profile-likelihood intervals; separate Wald coefficient summaries are labelled. Poisson and NB2 count models require positive exposure offsets; NB2 dispersion is re-estimated in both nested models. Ordered outcomes require a justified proportional-odds model, and multinomial outcomes require an explicit reference category. MASS and nnet are required for the corresponding examples. Singular, near-boundary and unresolved fits are refused. Complete-case exclusion is explicit; no model is selected by a pretest. Clustered Gaussian examples use nlme REML random-intercept models with a prespecified compound-symmetric covariance and containment degrees of freedom. The repeated example supports an occasion factor; the cluster example uses a cluster-level factor. Binary examples distinguish conditional lme4 mixed-model odds ratios from marginal geepack GEE odds ratios. GEE intervals use the robust sandwich without a small-cluster correction. MAR available-case likelihood and MCAR unweighted GEE assumptions are explicit. Boundary and unresolved fits are refused, and the implementation cluster minimum is not an adequacy guarantee. Prediction examples refit preprocessing and the prespecified model inside every individual bootstrap resample. They report apparent and optimism-corrected performance, Monte Carlo standard errors and all failed-resample counts. Any failed resample prevents selective averaging. Logistic risk examples include fixed-orientation AUC, Brier/log loss and calibration intercept/slope. Internal validation is not external validation; Monte Carlo precision is not a sampling interval. No cutoff, hypothesis-test

power or prediction-model sample-size calculation is substituted. RNG state is restored after use. Common-factor examples fit a prespecified factor count by maximum likelihood with oblique promax rotation and explicit Heywood-boundary refusal. They do not substitute PCA or select factor counts automatically. Clustering examples state k, scaling, distance and algorithm, and assess80-percent subsampling stability with adjusted Rand indices. Gower/PAM requires declared variable types and weights. Cluster is required for Gower/PAM. Silhouette and internal stability do not establish real populations; failed subsamples are not selectively averaged. Exploratory procedures have no directional alternative. New-case linear predictions offer pointwise conditional-mean or new-outcome Student-t intervals under independent homoscedastic normal errors. Logistic risk intervals use ordinary-bootstrap percentiles (quantile type 6), requiring at least five expected draws per tail; this resolution floor does not guarantee coverage. Risk estimation intervals do not quantify binary-outcome uncertainty or validation-performance uncertainty. All resamples must fit successfully. The common-factor template also offers independent-column permutation parallel analysis with SMC communalities and explicit quantile thresholds. This is exploratory factor-count guidance with Monte Carlo variation, not automatic model selection. Prediction-performance exports require alternative = NULL; their confidence setting is used for the separate new-case prediction intervals. Diagnostic plots do not trigger a switch of test. Planning availability remains the availability recorded by the advisor; the script does not invent a sample size or substitute a pooled test for Welch planning.

Value

A single character string containing an R script. Use `writeLines()` to save it. Running the script prints a result; some examples also draw plots.

See Also

Other advisor: `advisor_answer()`, `advisor_override()`, `advisor_power_spec()`, `advisor_question()`, `advisor_recommendation()`, `advisor_replay()`, `advisor_rules()`, `advisor_save()`, `advisor_session()`, `advisor_set_language()`, `advisor_validate_rules()`

`advisor_recommendation` *Recommendation for an advisor session*

Description

Builds the immutable, versioned recommendation for the current state of a session: answered questions (raw and effective), the rule trace, outcome, status (`recommended`, `conditional`, `needs_information` or `outside_supported_scope`), estimand, rationale, assumptions (including those recorded by 'I do not know' answers and status modifiers, and with an override the overriding variant's own assumptions or an explicit record that the rules state none), alternatives with why-not texts and rejected routes, missing information, the R template id with an explicit availability status, planning compatibility and availability, learning resources with explicit gaps, source-tree nodes of both trees and the rules, variants and resources versions.

Usage

```
advisor_recommendation(session, rules = NULL)
```

Arguments

session	An advisor_session.
rules	An advisor_rules object, or NULL for the bundled rules. It must be the rule set the session was created with.

Details

Statuses follow the rule set: a recommended outcome is downgraded to conditional by an 'I do not know' answer that routes with an assumption, by an applicable status modifier or by an override; statuses are never upgraded. An unfinished walkthrough gives needs_information with the pending question. Planning compatibility is never exact_gpower_match without matching authentic 'G*Power' evidence and a qualified mapping; no advisor route currently claims this status.

Value

A recommendation object (see ig_to_json() for export).

See Also

Other advisor: advisor_answer(), advisor_override(), advisor_power_spec(), advisor_question(), advisor_r_code(), advisor_replay(), advisor_rules(), advisor_save(), advisor_session(), advisor_set_language(), advisor_validate_rules()

Examples

```
s <- advisor_replay(c(q.aim = "compare", q.stage = "planning",
  q.outcome_type = "binary", q.unit = "individuals",
  q.structure = "independent", q.independence_check = "no",
  q.n_factors = "one", q.n_groups = "two", q.covariates = "no",
  q.binary_margins = "group_sizes_fixed", q.direction = "two_sided",
  q.missingness = "none"))
rec <- advisor_recommendation(s)
rec$preferred$analysis_variant_id
rec$planning$planning_compatibility
```

---

advisor_replay *Replay an answer map (deep link)*

Description

Rebuilds a session from a map of question ids to option ids, for example from a deep link. The map is replayed through the graph in transition order: an unknown question or option id makes the whole map invalid (`invalid_input`), answers to questions that the path does not reach are reported in `session$ignored_answers` and do not influence the result, and a map that stops at an applicable but unanswered question leaves that question current (the recommendation is then `needs_information` with that pending question).

Usage

```
advisor_replay(answers, language = "en", rules = NULL)
```

Arguments

`answers`	A named character vector or named list: question id -> option id.
`language`	Language of question texts: `"en"` or `"de"`. Decisions do not depend on the language.
`rules`	An `advisor_rules` object, or `NULL` for the bundled rules.

Value

An `advisor_session`.

See Also

Other advisor: `advisor_answer()`, `advisor_override()`, `advisor_power_spec()`, `advisor_question()`, `advisor_r_code()`, `advisor_recommendation()`, `advisor_rules()`, `advisor_save()`, `advisor_session()`, `advisor_set_language()`, `advisor_validate_rules()`

Examples

```
s <- advisor_replay(c(q.aim = "compare", q.stage = "analysis",
  q.outcome_type = "continuous", q.variance = "justified"))
s$ignored_answers
advisor_question(s)$question_id
```

`advisor_rules` *Load an advisor rule set*

Description

Reads the versioned advisor data (rule graph, analysis variants, both source trees, the source crosswalk and the learning-resource registry), checks it and compiles every condition with an allowlisted interpreter. Conditions are JSON data (`q`/`in`, `all`, `any`, `not`, `always`); expression strings are never parsed or evaluated.

Usage

```
advisor_rules(
  rules = NULL,
  variants = NULL,
  uzh_tree = NULL,
  descrtab2_tree = NULL,
  crosswalk = NULL,
  resources = NULL
)
```

Arguments

`rules`, `variants`, `uzh_tree`, `descrtab2_tree`, `crosswalk`, `resources`
: Paths of local files, or `NULL` for the bundled copy: the rule graph (`rules.json`), analysis variants, UZH tree, DescrTab2 tree, source crosswalk (CSV) and learning-resource registry.

Details

Without arguments the rule set bundled with the package is returned (it is read once and cached). Its files are byte-identical copies of the audited specification and pass `advisor_validate_rules()` in the package tests, so at load time only their structure is checked. Paths can be given to load a different copy, for example a revised draft; such a copy is always fully validated first (including the exhaustive walk, about two seconds) and refused if any check fails. Use `advisor_validate_rules()` to inspect the problems of a draft without loading it.

Value

An object of class `advisor_rules`. Its `versions` element holds the rules id and version, the source-tree, variant and resource versions and SHA-256 content hashes of every input. A rule set that fails a check raises a condition of class `inferguidR_invalid_input` whose `$status$details$problems` lists every problem as `"[check] message"`.

See Also

`advisor_validate_rules()`, `advisor_session()`

Other advisor: `advisor_answer()`, `advisor_override()`, `advisor_power_spec()`, `advisor_question()`, `advisor_r_code()`, `advisor_recommendation()`, `advisor_replay()`, `advisor_save()`, `advisor_session()`, `advisor_set_language()`, `advisor_validate_rules()`

Examples

```
rs <- advisor_rules()
rs$versions$rules_version
```

advisor_save *Save and restore an advisor session*

Description

advisor_save() writes the session state as canonical JSON: session schema version, rules id, version and content hash, language, raw answer trace, override, ignored deep-link answers and the outcome and status the state produced, sealed by a SHA-256 state hash.

Usage

```
advisor_save(session, file = NULL, rules = NULL)

advisor_restore(json = NULL, file = NULL, rules = NULL)
```

Arguments

session	An advisor_session.
file	Path of a local file to write (advisor_save()) or read (advisor_restore()).
rules	An advisor_rules object, or NULL for the bundled rules. It must be the rule set the session was created with.
json	A JSON string produced by advisor_save().

Details

advisor_restore() reads such a document as data (nothing is evaluated). With the same rules version the answers are replayed and must reproduce the saved outcome and status. A state saved under a different rules version is not re-interpreted: a condition of class inferguidR_migration_required (status invalid_input) is raised, with the rule change log in $status$details$change_log, so that an explicit migration can map the answers and report changes.

Value

advisor_save() returns the JSON text (class json), invisibly when file is given. advisor_restore() returns an advisor_session.

See Also

Other advisor: advisor_answer(), advisor_override(), advisor_power_spec(), advisor_question(), advisor_r_code(), advisor_recommendation(), advisor_replay(), advisor_rules(), advisor_session(), advisor_set_language(), advisor_validate_rules()

Examples

```
s <- advisor_answer(advisor_session(), "compare")
js <- advisor_save(s)
identical(advisor_restore(js), s)
```

advisor_session *Start a guided advisor walkthrough*

Description

Creates an empty advisor session at the first question of the rule graph. Sessions are immutable: advisor_answer(), advisor_back(), advisor_override() and advisor_set_language() return new sessions. Only questions that apply to the answers given so far are ever asked; 'I do not know' is an answer on every question and never invents a default (its policy either stops with needs_information, routes without an extra assumption, or routes with a recorded assumption and a downgrade to conditional).

Usage

```
advisor_session(language = "en", rules = NULL)
```

Arguments

language	Language of question texts: "en" or "de". Decisions do not depend on the language.
rules	An advisor_rules object, or NULL for the bundled rules.

Value

An object of class advisor_session.

See Also

advisor_answer(), advisor_recommendation(), advisor_save()

Other advisor: advisor_answer(), advisor_override(), advisor_power_spec(), advisor_question(), advisor_r_code(), advisor_recommendation(), advisor_replay(), advisor_rules(), advisor_save(), advisor_set_language(), advisor_validate_rules()

Examples

```
s <- advisor_session()
s <- advisor_answer(s, "compare")
advisor_question(s)$question_id
```

advisor_set_language *Change the language of an advisor session*

Description

Switches question texts between English and German. Answers, outcome and status are unchanged.

Usage

```
advisor_set_language(session, language, rules = NULL)
```

Arguments

session	An advisor_session.
language	"en" or "de".
rules	An advisor_rules object, or NULL for the bundled rules. It must be the rule set the session was created with.

Value

A new advisor_session.

See Also

Other advisor: advisor_answer(), advisor_override(), advisor_power_spec(), advisor_question(), advisor_r_code(), advisor_recommendation(), advisor_replay(), advisor_rules(), advisor_save(), advisor_session(), advisor_validate_rules()

Examples

```
advisor_question(advisor_set_language(advisor_session(), "de"))$text
```

advisor_validate_rules *Validate an advisor rule set*

Description

Checks a rule graph and its companion files before they are used:

Usage

```
advisor_validate_rules(
  rules = NULL,
  variants = NULL,
  uzh_tree = NULL,
  descrtab2_tree = NULL,
  crosswalk = NULL,
  resources = NULL,
  walk = TRUE
)
```

Arguments

rules, variants, uzh_tree, descrtab2_tree, crosswalk, resources	Paths of local files, or NULL for the bundled copy: the rule graph (rules.json), analysis variants, UZH tree, DescrTab2 tree, source crosswalk (CSV) and learning-resource registry.
walk	If TRUE (default), enumerate all answer paths. The walk is skipped when structural problems make it meaningless.

Details

- structure: unique ids, allowlisted conditions that reference existing questions and options, contiguous transition orders ending in an unconditional guard, existing targets;
- content: English and German texts for every question, option and outcome; an 'I do not know' option with explanation and policy on every question; routing only on design and aim (no normality, variance or sphericity pre-test and no p-value wording); no planning claim of exact_gpower_match or independently_validated_extension; template ids that match the primary variant; resolvable variant, reference, source-node, crosswalk and learning ids; documented summary counts;
- walk: an exhaustive enumeration of every answer path, which must never reach a guard, must respect applies_when, must not cycle, must reach every question, outcome and non-guard transition, and must match the documented path count; design-consistency invariants are evaluated on every path (for example, paired data never reach an independent-groups test, and two independent binomial samples are always analysed with Boschloo's test).

Value

An object of class advisor_rules_validation: a list with valid (logical), problems (data frame with columns check and message), walk (enumeration summary, or NULL) and versions. Files that cannot be read or parsed (not JSON, duplicate keys, remote paths, unexpected shapes) give valid = FALSE with a schema problem; the function does not raise errors for defective rule sets.

See Also

advisor_rules()

Other advisor: advisor_answer(), advisor_override(), advisor_power_spec(), advisor_question(), advisor_r_code(), advisor_recommendation(), advisor_replay(), advisor_rules(), advisor_save(), advisor_session(), advisor_set_language()

Examples

```
v <- advisor_validate_rules()
v$valid
v$walk$answer_paths
```

---

analysis_envelope *Wrap a specification and its result in an analysis envelope*

Description

The envelope is the unit that reports, comparisons and the application pass around. It pairs one immutable specification with its result (or no result yet) and records shared provenance.

Usage

```
analysis_envelope(spec, result = NULL, kind = NULL)
```

Arguments

spec	A power_spec (or, from later phases, a classification_spec or observed_test_spec).
result	The matching result, or NULL when not computed yet.
kind	Optional analysis kind ("power", "classification", "observed_test"); inferred from spec when omitted.

Value

An immutable object of class analysis_envelope.

Examples

```
spec <- analysis_spec("t_means_one_sample", "post_hoc",
  inputs = list(alpha = 0.05, n_total = 20, d = 0.5),
  options = list(tails = "two"))
analysis_envelope(spec, calculate_power(spec))
```

`analysis_spec` *Create an analysis specification*

Description

Builds the immutable specification of one power analysis. The method (see `list_methods()`), the analysis `mode` and the inputs are validated: unknown or inapplicable inputs and options, domain violations, a supplied solve target, missing required values and inconsistent duplicate inputs are rejected with a typed `invalid_input` error. Every default that is applied is written into the specification.

Usage

```
analysis_spec(
  method,
  mode,
  inputs = list(),
  options = list(),
  target = NULL,
  ...
)
```

Arguments

`method`	Method id.
`mode`	Analysis mode.
`inputs`	Named list of inputs (numbers, integers, flags or codes).
`options`	Named list (or named character vector) of option codes.
`target`	Optional solve target. Defaults to the mode's target; needed only where a method offers alternative targets.
`...`	Further named inputs.

Details

Modes and their solve targets:

- `"a_priori"`: total sample size (`n_total`) from effect, `alpha` and `power`;
- `"post_hoc"`: `power` from effect, `alpha` and the sample size;
- `"sensitivity"`: the effect size from `alpha`, `power` and the sample size;
- `"criterion"`: `alpha` (and critical values) from effect, `power` and the sample size;
- `"compromise"`: `alpha` and `power` jointly from effect, sample size and the ratio `error_ratio` = beta/alpha.

For methods without a registered engine the specification is validated against the registry metadata only (standard inputs, options and modes); other inputs are recorded as `unvalidated_inputs`. Such specifications can be stored and exported, and `calculate_power()` returns status `"unsupported"` for them.

Inputs may be given in `inputs`, in `...`, or both. The same name given twice with identical values is accepted once; with different values it is rejected.

Value

An immutable object of class `power_spec`.

See Also

`update.power_spec()`, `calculate_power()`, `ig_to_json()`

Examples

```
spec <- analysis_spec("t_means_two_independent", "a_priori",
  inputs = list(alpha = 0.05, power = 0.8, d = 0.5),
  options = list(tails = "two"))
spec
# This method has an engine (phase 05 vertical slice), so a number is returned:
calculate_power(spec)
```

`attach_classification_data` *Reattach scored data to a classification specification*

Description

Specifications serialised to JSON carry a SHA-256 fingerprint of the analysed data, never the rows. To evaluate such a specification again, reattach the original data: the columns named in the model records are read and the fingerprint must match exactly.

Usage

```
attach_classification_data(spec, data)
```

Arguments

`spec`	A `classification_spec` with input kind `scored_data`.
`data`	The original data frame.

Value

The specification with the data attached.

See Also

- classification_spec()
- evaluate_cutoff()
- plot_classification()

Examples

```
d <- data.frame(y = c(1, 1, 0, 0, 0), s = c(3, 2, 2, 1, 0))
spec <- classification_spec(d, truth = "y", positive_label = 1, scores = "s", cutoff = 2)
back <- ig_from_json(ig_to_json(analysis_envelope(spec)))$spec
spec2 <- attach_classification_data(back, d)
identical(spec2$spec_hash, spec$spec_hash)
```

---

binormal_model *Binormal score model*

Description

Parameters of the binormal model used for theoretical classification scenarios: non-events $S_0 \sim N(0,1)$, events $S_1 \sim N(d, \sigma^2)$ with $d = \Phi^{-1}(AUC)\sqrt{1+\sigma^2}$.

Usage

```
binormal_model(auc, sigma = 1)
```

Arguments

auc	Area under the ROC curve in (0, 1).
sigma	Ratio of the event to the non-event standard deviation (> 0).

Value

A list with auc, sigma, d, the class distributions and auc_check (the AUC recomputed as $\Phi(d/\sqrt{1+\sigma^2})$).

See Also

- classification_spec()
- evaluate_cutoff()
- plot_classification()

Examples

```
binormal_model(0.8, 1.5)
```

binormal_rates *Analytic rates of the binormal model at given cutoffs*

Description

Analytic rates of the binormal model at given cutoffs

Usage

```
binormal_rates(cutoff, auc, sigma = 1)
```

Arguments

cutoff	Numeric vector of real-valued cutoffs (alert if score >= cutoff) on the score scale standardised to the non-event distribution.
auc	Area under the ROC curve in (0, 1).
sigma	Ratio of the event to the non-event standard deviation (> 0).

Value

A data frame with `cutoff`, `tpr` and `fpr`, computed from the normal upper tails (no discretisation).

See Also

- `classification_spec()`
- `evaluate_cutoff()`
- `plot_classification()`

Examples

```
binormal_rates(c(0, 1, 2), auc = 0.8, sigma = 1)
```

boschloo_compare_fisher *Compare Boschloo's and Fisher's rejection regions*

Description

Classifies every possible table of a design as rejected by both tests, by Boschloo's test only, by Fisher's conditional test only, or by neither, and reports the attained size of both regions and (optionally) their power at `(p1, p2)`. Both tests are evaluated on the same design, level and sidedness convention unless `fisher_convention = "minlike"` requests the two-sided Fisher p-value of `stats::fisher.test()` (which is a different two-sided convention from the central variant and is labelled as such).

Usage

```
boschloo_compare_fisher(
  n1,
  n2,
  alpha = 0.05,
  variant,
  p1 = NULL,
  p2 = NULL,
  fisher_convention = c("same", "minlike"),
  control = boschloo_control()
)
```

Arguments

n1, n2	Planned group sizes.
alpha	Nominal significance level.
variant	"minlike", "central", "less" or "greater" (see boschloo_test()).
p1, p2	Optional alternative event probabilities for a power comparison.
fisher_convention	"same" (default) or "minlike".
control	Numerical settings from boschloo_control().

Details

With the same convention, Boschloo's p-value never exceeds Fisher's. The result checks this on the p-values of every table (dominance_holds, computed for designs with at most 5000 tables). Fisher p-values within the relative tie tolerance of alpha count as equal to alpha.

Value

An object of class boschloo_fisher_comparison with a table of all (x1, x2) with both p-values (the Boschloo p-value is NA above 5000 tables), both decisions and the category, counts per category, the dominance check, attained sizes and power of both tests.

See Also

Other Boschloo: boschloo_control(), boschloo_plotdata_allocation(), boschloo_plotdata_lattice(), boschloo_plotdata_nuisance(), boschloo_plotdata_power_effect(), boschloo_plotdata_power_n(), boschloo_power(), boschloo_probabilities(), boschloo_region(), boschloo_sample_size(), boschloo_test(), boschloo_variants()

Examples

```
cmp <- boschloo_compare_fisher(12, 12, alpha = 0.05, variant = "central",
                               p1 = 0.2, p2 = 0.7)
cmp$counts
```

`boschloo_control` *Numerical settings for Boschloo's test*

Description

Collects the settings that affect Boschloo p-values, rejection regions, power and sample sizes. They are recorded in every result and are part of every cache key, so results computed with different settings are never mixed.

Usage

```
boschloo_control(
  tie_tolerance = 1e-07,
  grid_points = 1001L,
  nuisance_tol = 1e-09,
  polish_tol = 1e-12,
  max_levels = 60L,
  ambiguity_band = c(1e-12, 1e-06),
  cache = TRUE
)
```

Arguments

`tie_tolerance`	Relative tie tolerance for ordering statistics (default `1e-7`, the tolerance used by `stats::fisher.test()`). It must lie in `[ambiguity_band[1], ambiguity_band[2]]`.
`grid_points`	Number of points of the global nuisance grid on `[0, 1]` (default `1001`).
`nuisance_tol`	Absolute tolerance of the rigorous supremum bound (default `1e-9`).
`polish_tol`	Tolerance of the final Brent refinement of the maximiser (default `1e-12`).
`max_levels`	Maximum number of bisection levels (default `60`).
`ambiguity_band`	Relative gaps between distinct ordering values that are flagged as ambiguous (default `c(1e-12, 1e-6)`, a half-open interval `(lower, upper]` with `lower <= 1e-12` and `1e-10 <= upper <= 1e-3`).
`cache`	Cache orderings and rejection regions in memory (keyed by design, variant, alpha and all settings above).

Details

The nuisance supremum `sup_pi P_pi(A)` of a tail set `A` is written in Bernstein form `sum_k c_k dbinom(k, N, pi)`, where `c_k` is the conditional null probability of `A` given `k` events in total. It is maximised by a global grid of `grid_points` equally spaced values of `pi` (both endpoints included), followed by branch-and-bound bisection: every grid interval is bounded with a rigorous local bound of the second derivative of the Bernstein polynomial, and intervals whose bound exceeds the best value found by more than `nuisance_tol` are bisected. A final Brent step (`stats::optimize()`, tolerance `polish_tol`) polishes the maximiser. Each supremum is reported as a pair: the refined value (`p_value`, a lower bound attained at a computed point) and a rigorous upper bound (`p_upper`), with

p_upper - p_value <= nuisance_tol plus a floating-point allowance. Tables are rejected only when p_upper <= alpha.

Ordering statistics are compared with the relative tolerance tie_tolerance: values whose relative difference is at most this tolerance are treated as ties (both tables enter the tail set together). Values above 0.5 are compared on the complementary tail, which is computed directly so that values near 1 keep their relative accuracy. A pair of distinct ordering values whose relative gap falls in ambiguity_band is flagged as an ambiguous tie. For n1 + n2 <= 56 the ordering is then recomputed in exact integer arithmetic; above that size a result whose tail set depends on the ambiguous pair is refused (status "numerical_failure").

Gaps at or below the lower edge of ambiguity_band are taken as exact ties, and gaps above the upper edge as exact distinctions, without a flag. The tie tolerance therefore has to lie inside the band: a tolerance below the lower edge would split exactly tied tables whose computed values differ by rounding (for 1/5 versus 6/7 with variant = "less" it would give p = 0.0174 instead of 0.0232), and a tolerance above the upper edge would silently merge distinct values. For the same reason the lower edge is at most 1e-12 (distinct values come as close as about 4e-10 at (500, 1500)), and the upper edge is at least 1e-10 (exactly tied values are computed at most about 4.6e-13 apart up to n1 + n2 = 2000).

Value

An object of class boschloo_control.

See Also

Other Boschloo: boschloo_compare_fisher(), boschloo_plotdata_allocation(), boschloo_plotdata_lattice(), boschloo_plotdata_nuisance(), boschloo_plotdata_power_effect(), boschloo_plotdata_power_n(), boschloo_power(), boschloo_probabilities(), boschloo_region(), boschloo_sample_size(), boschloo_test(), boschloo_variants()

Examples

```
boschloo_control()
boschloo_control(grid_points = 2001L)
```

---

boschloo_plotdata_allocation

Plot data: Boschloo power versus the allocation at a fixed total

---

Description

Exact power and attained size of every split n1 + n2 = n_total (from n1 = 1 to n1 = n_total - 1, or the splits in n1), each with its own rejection region. Shows how unequal allocation changes power and how discreteness makes neighbouring splits differ.

Usage

```
boschloo_plotdata_allocation(
  p1,
  p2,
  n_total,
  alpha = 0.05,
  variant,
  n1 = NULL,
  control = boschloo_control()
)
```

Arguments

p1, p2	Event probabilities of group 1 and group 2 under the alternative (in [0, 1]). With p1 = p2 the result is a null rejection probability, at most the attained size.
n_total	Fixed total sample size (at least 2).
alpha	Nominal significance level.
variant	"minlike", "central", "less" or "greater" (see boschloo_test()).
n1	Optional subset of group-1 sizes to evaluate.
control	Numerical settings from boschloo_control().

Value

An object of class boschloo_plotdata (plot id "boschloo_power_allocation").

See Also

Other Boschloo: boschloo_compare_fisher(), boschloo_control(), boschloo_plotdata_lattice(), boschloo_plotdata_nuisance(), boschloo_plotdata_power_effect(), boschloo_plotdata_power_n(), boschloo_power(), boschloo_probabilities(), boschloo_region(), boschloo_sample_size(), boschloo_test(), boschloo_variants()

Examples

```
pd <- boschloo_plotdata_allocation(0.2, 0.6, n_total = 30, variant = "central",
                                   n1 = 8:22)
pd$data[which.max(pd$data$power), ]
```

---

boschloo_plotdata_lattice

Plot data: lattice of possible tables with the Boschloo rejection region

---

Description

One row per possible table (x1, x2) of a planned design: the sample proportions, the Fisher ordering statistic, the Boschloo p-value with its upper bound, rejection membership at alpha, and the probability of the table under the null at the size-maximising common probability and under the alternative (p1, p2) when given. An observed table can be marked; marking it does not change the planning inputs.

Usage

```
boschloo_plotdata_lattice(
  n1,
  n2,
  alpha = 0.05,
  variant,
  p1 = NULL,
  p2 = NULL,
  observed = NULL,
  pvalues = TRUE,
  control = boschloo_control()
)
```

Arguments

n1, n2	Planned group sizes.
alpha	Nominal significance level.
variant	"minlike", "central", "less" or "greater" (see boschloo_test()).
p1, p2	Optional alternative event probabilities.
observed	Optional c(x1, x2) to mark.
pvalues	Compute the Boschloo p-value of every table (one supremum per ordering level; skipped above 5000 tables).
control	Numerical settings from boschloo_control().

Value

An object of class boschloo_plotdata (plot id "boschloo_lattice").

See Also

Other Boschloo: boschloo_compare_fisher(), boschloo_control(), boschloo_plotdata_allocation(), boschloo_plotdata_nuisance(), boschloo_plotdata_power_effect(), boschloo_plotdata_power_n(), boschloo_power(), boschloo_probabilities(), boschloo_region(), boschloo_sample_size(), boschloo_test(), boschloo_variants()

Examples

```
pd <- boschloo_plotdata_lattice(8, 10, alpha = 0.05, variant = "minlike",
                                p1 = 0.2, p2 = 0.7)
head(pd$data)
```

`boschloo_plotdata_nuisance` *Plot data: nuisance envelope*

Description

The null probability of a tail set as a function of the common event probability `pi`. For an observed test (a `boschloo_test()` result) the curve is `P_pi(tables at least as extreme as the observed one)`, whose maximum is the p-value (two curves for the central variant). For a rejection region (a `boschloo_region()` result) it is the null rejection probability, whose maximum is the attained size.

Usage

```
boschloo_plotdata_nuisance(x, grid_points = 401L, control = NULL)
```

Arguments

`x`	A `boschloo_test` or `boschloo_region` object.
`grid_points`	Number of equally spaced values of `pi` (the maximiser is added).
`control`	Numerical settings. By default they are taken from `x`; a supplied value must match the settings recorded in `x`.

Value

An object of class `boschloo_plotdata` (plot id `"boschloo_nuisance"`).

See Also

Other Boschloo: `boschloo_compare_fisher()`, `boschloo_control()`, `boschloo_plotdata_allocation()`, `boschloo_plotdata_lattice()`, `boschloo_plotdata_power_effect()`, `boschloo_plotdata_power_n()`, `boschloo_power()`, `boschloo_probabilities()`, `boschloo_region()`, `boschloo_sample_size()`, `boschloo_test()`, `boschloo_variants()`

Examples

```
res <- boschloo_test(3, 15, 9, 15, variant = "minlike")
env <- boschloo_plotdata_nuisance(res)
max(env$data$probability)
```

`boschloo_plotdata_power_effect` *Plot data: Boschloo power versus the alternative*

Description

Exact power of one fixed design (one rejection region) over a set of alternatives (`p1, p2`). `p1` and `p2` are recycled to a common length, so a scalar `p1` with a vector `p2` gives power versus the group-2 probability, and a vector `p1` with `p2 = p1 + d` gives power versus the baseline at a fixed difference `d`.

Usage

```
boschloo_plotdata_power_effect(
  n1,
  n2,
  p1,
  p2,
  alpha = 0.05,
  variant,
  control = boschloo_control()
)
```

Arguments

`n1, n2`	Planned group sizes.
`p1, p2`	Event probabilities of group 1 and group 2 (vectors, recycled to a common length).
`alpha`	Nominal significance level.
`variant`	`"minlike"`, `"central"`, `"less"` or `"greater"` (see `boschloo_test()`).
`control`	Numerical settings from `boschloo_control()`.

Value

An object of class `boschloo_plotdata` (plot id `"boschloo_power_effect"`).

See Also

Other Boschloo: `boschloo_compare_fisher()`, `boschloo_control()`, `boschloo_plotdata_allocation()`, `boschloo_plotdata_lattice()`, `boschloo_plotdata_nuisance()`, `boschloo_plotdata_power_n()`, `boschloo_power()`, `boschloo_probabilities()`, `boschloo_region()`, `boschloo_sample_size()`, `boschloo_test()`, `boschloo_variants()`

Examples

```
pd <- boschloo_plotdata_power_effect(20, 20, p1 = 0.2, p2 = seq(0, 1, 0.1),
                                     variant = "central")
pd$data
base <- seq(0.05, 0.6, by = 0.05)
boschloo_plotdata_power_effect(20, 20, p1 = base, p2 = base + 0.3,
                               variant = "central")$data
```

`boschloo_plotdata_power_n` *Plot data: Boschloo power versus sample size*

Description

Exact power and attained size of every design of an allocation sequence between two total sample sizes, with the target line. The staircase is not smoothed; designs are the integer designs of the declared allocation.

Usage

```
boschloo_plotdata_power_n(
  p1,
  p2,
  alpha = 0.05,
  variant,
  n_range,
  power = NULL,
  allocation_ratio = 1,
  allocation = c("n2_ceiling", "total_nearest"),
  fisher = FALSE,
  control = boschloo_control()
)
```

Arguments

p1, p2	Event probabilities of group 1 (reference) and group 2 under the alternative; see `boschloo_probabilities()` for baseline-plus-effect input.
alpha	Nominal significance level.
variant	`"minlike"`, `"central"`, `"less"` or `"greater"` (see `boschloo_test()`).
n_range	Range of total sample sizes `c(from, to)`.
power	Target power.
allocation_ratio	The allocation ratio r = n2 / n1 (default 1).
allocation	`"n2_ceiling"` (default) or `"total_nearest"`.
fisher	Also compute the power of Fisher's conditional test with the same convention (comparison only).
control	Numerical settings from `boschloo_control()`.

Value

An object of class boschloo_plotdata (plot id "boschloo_power_n").

See Also

Other Boschloo: boschloo_compare_fisher(), boschloo_control(), boschloo_plotdata_allocation(), boschloo_plotdata_lattice(), boschloo_plotdata_nuisance(), boschloo_plotdata_power_effect(), boschloo_power(), boschloo_probabilities(), boschloo_region(), boschloo_sample_size(), boschloo_test(), boschloo_variants()

Examples

```
pd <- boschloo_plotdata_power_n(0.2, 0.7, alpha = 0.05, variant = "central",
                                n_range = c(10, 40))
head(pd$data)
```

---

boschloo_power *Power of Boschloo's test for a fixed design*

Description

Computes the exact power at the alternative (p1, p2) as the double binomial sum over the rejection region of boschloo_region(): the probability that a future study with group sizes n1 and n2 produces a table that Boschloo's test rejects. No asymptotic approximation is used.

Usage

```
boschloo_power(
  n1,
  n2,
  p1,
  p2,
  alpha = 0.05,
  variant,
  control = boschloo_control()
)
```

Arguments

n1, n2	Planned group sizes.
p1, p2	Event probabilities of group 1 and group 2 under the alternative (in [0, 1]). With p1 = p2 the result is a null rejection probability, at most the attained size.
alpha	Nominal significance level.
variant	"minlike", "central", "less" or "greater" (see boschloo_test()).
control	Numerical settings from boschloo_control().

Value

An immutable object of class `boschloo_power` with the power, the attained size, the number of rejected tables, the region hash and the inputs.

See Also

Other Boschloo: `boschloo_compare_fisher()`, `boschloo_control()`, `boschloo_plotdata_allocation()`, `boschloo_plotdata_lattice()`, `boschloo_plotdata_nuisance()`, `boschloo_plotdata_power_effect()`, `boschloo_plotdata_power_n()`, `boschloo_probabilities()`, `boschloo_region()`, `boschloo_sample_size()`, `boschloo_test()`, `boschloo_variants()`

Examples

```
boschloo_power(20, 20, p1 = 0.2, p2 = 0.6, alpha = 0.05, variant = "central")
```

`boschloo_probabilities` *Event probabilities from a baseline and an effect*

Description

Converts a baseline event probability of group 1 (the reference group) and one effect measure into the probability of group 2, checks that it is a valid probability and returns all effect scales. Effects are oriented as group 2 relative to group 1: difference `p2 - p1`, ratio `p2 / p1`, odds ratio `odds(p2) / odds(p1)`.

Usage

```
boschloo_probabilities(
  p1,
  p2 = NULL,
  difference = NULL,
  ratio = NULL,
  odds_ratio = NULL
)
```

Arguments

`p1`	Event probability of group 1, in `[0, 1]`.
`p2`, `difference`, `ratio`, `odds_ratio`	Exactly one of: the probability of group 2, the risk difference, the risk ratio or the odds ratio.

Value

A list with `p1`, `p2`, `difference`, `ratio` and `odds_ratio` (`NA` with a reason in `undefined` when a scale is not defined, for example a ratio with `p1 = 0`).

See Also

Other Boschloo: boschloo_compare_fisher(), boschloo_control(), boschloo_plotdata_allocation(), boschloo_plotdata_lattice(), boschloo_plotdata_nuisance(), boschloo_plotdata_power_effect(), boschloo_plotdata_power_n(), boschloo_power(), boschloo_region(), boschloo_sample_size(), boschloo_test(), boschloo_variants()

Examples

```
boschloo_probabilities(0.2, difference = 0.15)
boschloo_probabilities(0.2, odds_ratio = 3)
```

---

boschloo_region *Rejection region of Boschloo's test for a fixed design*

Description

Enumerates all `(n1 + 1) * (n2 + 1)` possible tables of a planned study and returns the tables that Boschloo's test rejects at level `alpha`, together with the attained size (the largest null rejection probability over the common event probability). The region is exactly the set of tables for which boschloo_test() decides `"reject"`.

Usage

```
boschloo_region(n1, n2, alpha = 0.05, variant, control = boschloo_control())
```

Arguments

`n1`, `n2`	Planned group sizes.
`alpha`	Nominal significance level.
`variant`	`"minlike"`, `"central"`, `"less"` or `"greater"` (see boschloo_test()).
`control`	Numerical settings from boschloo_control().

Value

An immutable object of class `boschloo_region` with the rejected tables (`rejected`, columns `x1`, `x2`), the attained size (`size`, with value, rigorous upper bound and maximising nuisance probability), the boundary of each ordering side, the status, the settings and a region hash.

See Also

Other Boschloo: boschloo_compare_fisher(), boschloo_control(), boschloo_plotdata_allocation(), boschloo_plotdata_lattice(), boschloo_plotdata_nuisance(), boschloo_plotdata_power_effect(), boschloo_plotdata_power_n(), boschloo_power(), boschloo_probabilities(), boschloo_sample_size(), boschloo_test(), boschloo_variants()

Examples

```
reg <- boschloo_region(10, 12, alpha = 0.05, variant = "central")
reg
reg$size$value
```

`boschloo_sample_size` *Sample size for Boschloo's test*

Description

Finds the smallest design of a declared allocation sequence whose exact Boschloo power at `(p1, p2)` reaches `power`, recomputing the achieved power and attained size of every reported design from its rejection region (the same region as `boschloo_region()` and `boschloo_test()`).

Usage

```
boschloo_sample_size(
  p1,
  p2,
  power = 0.8,
  alpha = 0.05,
  variant,
  allocation_ratio = 1,
  allocation = c("n2_ceiling", "total_nearest"),
  stability_window = 10L,
  n_max = 500L,
  control = boschloo_control()
)
```

Arguments

`p1`, `p2`	Event probabilities of group 1 (reference) and group 2 under the alternative; see `boschloo_probabilities()` for baseline-plus-effect input.
`power`	Target power.
`alpha`	Nominal significance level.
`variant`	`"minlike"`, `"central"`, `"less"` or `"greater"` (see `boschloo_test()`).
`allocation_ratio`	The allocation ratio r = n2 / n1 (default 1).
`allocation`	`"n2_ceiling"` (default) or `"total_nearest"`.
`stability_window`	`K`, the number of following designs that must also reach the target for the `stable` design (default 10).
`n_max`	Largest total sample size searched (default 500).
`control`	Numerical settings from `boschloo_control()`.

Details

Because power is not monotone in the sample size, two designs are reported:

- `first`: the first design in the sequence whose power reaches the target. Every earlier design up to it was either evaluated and failed, or is excluded by a Neyman-Pearson bound (the power of the most powerful level-`alpha` test of a simple null `(pi0, pi0)` against `(p1, p2)`, which no level-`alpha` test can exceed). It is therefore the minimum over the admissible designs with `n1 + n2 <= n_max`.
- `stable`: the first design from which the next `stability_window` designs of the sequence also reach the target (the window `[N, N + K]` in design steps). Nothing is claimed for designs beyond the window.

Allocation policies (allocation ratio r = `allocation_ratio` = n2 / n1):

- `"n2_ceiling"`: `n1 = 1, 2, ...` and `n2 = max(1, ceiling(r * n1))`;
- `"total_nearest"`: `N = 2, 3, ...`, `n1 = round(N / (1 + r))` (halves rounded up, kept between 1 and `N - 1`) and `n2 = N - n1`.

Value

An immutable object of class `boschloo_sample_size` with status (`"ok"`; `"infeasible"` when no design up to `n_max` reaches the target or when `p1 = p2`; `"invalid_input"` for a one-sided variant whose direction contradicts the alternative or a target power not above `alpha`; `"numerical_failure"` when a region cannot be certified), the `first` and `stable` designs (group sizes, total, achieved power, attained size), the evaluated staircase, the Neyman-Pearson exclusion, the minimality evidence and the search policy. Malformed arguments (for example a probability outside `[0, 1]`) signal an `inferguidR_invalid_input` condition instead.

See Also

Other Boschloo: `boschloo_compare_fisher()`, `boschloo_control()`, `boschloo_plotdata_allocation()`, `boschloo_plotdata_lattice()`, `boschloo_plotdata_nuisance()`, `boschloo_plotdata_power_effect()`, `boschloo_plotdata_power_n()`, `boschloo_power()`, `boschloo_probabilities()`, `boschloo_region()`, `boschloo_test()`, `boschloo_variants()`

Examples

```
ss <- boschloo_sample_size(p1 = 0.2, p2 = 0.7, power = 0.8, alpha = 0.05,
                           variant = "central")
ss
ss$first
```

boschloo_test *Boschloo's unconditional exact test for two independent binomial samples*

Description

Computes the Boschloo p-value of an observed table. Group 1 is `(x1, n1)`, group 2 is `(x2, n2)`; `x1` and `x2` count the events (the coding is recorded through `groups` and `event`). The null hypothesis is `p1 = p2`. Tables are ordered by a Fisher conditional p-value, and the p-value is the largest null probability, over the common event probability, of all tables that are at least as extreme as the observed one (ties included).

Usage

```
boschloo_test(
  x1,
  n1,
  x2,
  n2,
  variant,
  alpha = NULL,
  groups = c("group 1", "group 2"),
  event = "event",
  control = boschloo_control()
)
```

Arguments

`x1`, `x2`	Numbers of events in group 1 and group 2.
`n1`, `n2`	Group sizes (fixed by design).
`variant`	`"minlike"`, `"central"`, `"less"` or `"greater"`.
`alpha`	Optional significance level for a decision.
`groups`	Labels of group 1 and group 2 (recorded, not used in the computation).
`event`	Label of the counted outcome level.
`control`	Numerical settings from `boschloo_control()`.

Details

Variants (always stated explicitly):

- `"minlike"`: two-sided, ordered by the two-sided Fisher p-value of `stats::fisher.test()` (sum of conditional probabilities not larger than the observed one). Same definition as `Exact::exact.test(..., method = "boschloo", tsmethod = "square")`.
- `"central"`: two-sided, `min(1, 2 * min(p_less, p_greater))`, each one-sided p-value maximised separately.

- "less": one-sided, alternative p1 < p2.
- "greater": one-sided, alternative p1 > p2.

Mid-p values, randomisation, Berger-Boos confidence-set restrictions and non-zero null margins are not part of this test.

The p-value is reported with a rigorous upper bound (p_upper, see boschloo_control()). With alpha, the decision is "reject" when p_upper <= alpha, "do_not_reject" when p_value > alpha and "unresolved" otherwise (which requires the supremum to lie within the floating-point allowance of alpha). The same rule defines the rejection regions of boschloo_region(), so an observed table is rejected exactly when it lies in the region of its design.

Value

An immutable object of classes boschloo_test and observed_test_result with the p-value, its upper bound, the maximising nuisance probability, the ordering statistic, the Fisher p-value of the same sidedness convention, the observed specification ($spec), diagnostics and a reproducible call. Wrap it with analysis_envelope(res$spec, res).

See Also

Other Boschloo: boschloo_compare_fisher(), boschloo_control(), boschloo_plotdata_allocation(), boschloo_plotdata_lattice(), boschloo_plotdata_nuisance(), boschloo_plotdata_power_effect(), boschloo_plotdata_power_n(), boschloo_power(), boschloo_probabilities(), boschloo_region(), boschloo_sample_size(), boschloo_variants()

Examples

```
res <- boschloo_test(3, 15, 9, 15, variant = "minlike", alpha = 0.05)
res
res$p_value
boschloo_test(1, 20, 8, 20, variant = "less")$p_value
```

---

boschloo_variants *Boschloo test variants and their evidence status*

Description

Lists the four Boschloo variants with their identifiers, hypotheses, ordering statistics and two-sided constructions, the matching calls of the comparator packages, and the evidence status shared by all variants: the independent validation status, the planning-compatibility class and the provisional classification of the related G*Power procedures.

Usage

```
boschloo_variants()
```

Details

Group 1 is `(x1, n1)` and group 2 is `(x2, n2)` throughout; `x1` and `x2` count events. The null hypothesis is always `p1 = p2`. Mid-p values, randomisation, Berger-Boos confidence-set restrictions and non-zero null margins are not part of any variant.

Value

A data frame with one row per variant and these columns:

- `variant`, `variant_id`, `advisor_variant_id`: variant identifiers.
- `alternative`, `label`, `ordering_statistic`, `construction`: test definition.
- `exact_call`, `exact2x2_call`: comparator calls.
- `validation_status`, `planning_compatibility`: evidence status.
- `gpower_relation`, `gpower_classification_status`: reference relationship.

See Also

Other Boschloo: `boschloo_compare_fisher()`, `boschloo_control()`, `boschloo_plotdata_allocation()`, `boschloo_plotdata_lattice()`, `boschloo_plotdata_nuisance()`, `boschloo_plotdata_power_effect()`, `boschloo_plotdata_power_n()`, `boschloo_power()`, `boschloo_probabilities()`, `boschloo_region()`, `boschloo_sample_size()`, `boschloo_test()`

Examples

```
boschloo_variants()[, c("variant", "variant_id", "alternative")]
```

`calculate_power` *Calculate a power analysis*

Description

Solves the specification for its declared target: sample size (a priori), power (post hoc), effect size (sensitivity), alpha and critical values (criterion) or alpha and power jointly (compromise).

Usage

```
calculate_power(spec)
```

Arguments

`spec`	A `power_spec` created by `analysis_spec()`.

Details

The result always carries a typed status. Methods without an implemented engine return status `"unsupported"` with the reason `"engine for <method> not implemented yet"`; such results contain no numbers.

Value

A power_result.

See Also

solve_design(), analysis_spec()

Examples

```
spec <- analysis_spec("t_means_one_sample", "post_hoc",
  inputs = list(alpha = 0.05, n_total = 20, d = 0.5),
  options = list(tails = "two"))
res <- calculate_power(spec)
res$status
```

---

calculator_functions *Functions of the distribution calculator*

Description

Lists the functions that distribution_calculator() accepts, with their arguments in the order of the 'G*Power' 3.1.9.7 calculator, a description and notes on reference behaviour where 'G*Power' differs.

Usage

```
calculator_functions()
```

Value

A data frame with columns name, arguments, n_args, category (function or distribution), description and gpower_note (NA when there is nothing to note).

See Also

distribution_calculator()

Examples

```
fns <- calculator_functions()
fns[fns$category == "distribution", c("name", "arguments")][1:6, ]
```

`calibration_summary` *Calibration of predicted probabilities*

Description

Brier score, calibration in the large, calibration intercept and slope (logistic recalibration) and a grouped calibration table for scored models whose scores are declared as predicted probabilities of the positive class (`score_scale = "probability"`). Discrimination (AUC) does not establish calibration; this function assesses it separately. The groups of the table (quantiles of the predicted probability) are a display diagnostic only.

Usage

```
calibration_summary(spec, bins = 10)
```

Arguments

`spec`	A scored-data `classification_spec` with attached data.
`bins`	Number of quantile groups for the table (default 10); units with equal predicted probability stay in one group.

Value

A list with `summaries` (one row per model and quantity: `model_id`, `metric_id`, `value`, `status`, `reason_code`, `reason`, `setting`) and `table` (grouped observed versus predicted, one block per model).

See Also

- `classification_spec()`
- `evaluate_cutoff()`
- `plot_classification()`

Examples

```
ex <- classification_examples("scored_synthetic")
calibration_summary(ex$spec)$summaries
```

`classification_auc` *ROC AUC and partial AUC of scored models*

Description

Computes the empirical ROC AUC (ties count 1/2, declared direction, never auto-flipped), an optional partial AUC over a stated FPR or TPR range, and the DeLong confidence interval (own implementation; for a fixed set of scores, not for a selected cutoff). For a theoretical model the AUC is the model parameter and has no sampling interval.

Usage

```
classification_auc(spec, partial_auc = NULL, level = NULL)
```

Arguments

`spec`	A `classification_spec` (scored data with attached data, or theory).
`partial_auc`	Optional list with `focus` equal to `"fpr"` or `"tpr"` and `range` equal to `c(lo, hi)`. In pROC, `partial.auc = c(1, 0.8)` with `partial.auc.focus = "specificity"` corresponds to `focus = "fpr"` and `range = c(0, 0.2)` here.
`level`	Confidence level (default: the spec's uncertainty level).

Value

A data frame with one row per model and summary (`metric_id`, `variant`, `value`, `status`, `reason_code`, `reason`, `estimate_kind`, `setting`) and the DeLong interval columns `ci_lower`, `ci_upper`, `ci_status`, `ci_reason`.

See Also

- `classification_spec()`
- `evaluate_cutoff()`
- `plot_classification()`

Examples

```
d <- data.frame(y = rep(c(1, 0), c(6, 8)),
                s = c(9, 7, 7, 6, 5, 3, 7, 5, 4, 4, 3, 2, 2, 1))
spec <- classification_spec(d, truth = "y", positive_label = 1, scores = "s", cutoff = 5)
classification_auc(spec, partial_auc = list(focus = "fpr", range = c(0, 0.2)))
```

`classification_evaluated_spec` *Specification behind a classification result*

Description

`evaluate_cutoff()` with a cutoff override or a transferred rule, and `select_cutoff()`, evaluate an updated specification. This accessor returns that evaluated specification (with its data attached), which is the one an `analysis_envelope()` or a report must pair with the result.

Usage

```
classification_evaluated_spec(x)
```

Arguments

`x`	A `classification_result` computed in this session.

Details

The specification travels with the result in the R session only. A result read back from JSON has none; its envelope already holds the matching specification.

Value

The evaluated `classification_spec`.

See Also

- `classification_spec()`
- `evaluate_cutoff()`
- `plot_classification()`

Examples

```
ex <- classification_examples("scored_synthetic")
res <- select_cutoff(ex$spec, "max_youden")
spec <- classification_evaluated_spec(res)
spec$operating_rule$selection
analysis_envelope(spec, res)
```

`classification_examples` *Deterministic example classification analyses*

Description

Neutral synthetic examples (no real data, models or policy targets):

Usage

```
classification_examples(
 name = c("scored_synthetic", "binormal_equal_variance", "binormal_unequal_variance",
    "counts_fixed"),
  part = c("all", "tuning", "test"),
  models = c("model_a", "model_b")
)
```

Arguments

`name`	Example name.
`part`	For `"scored_synthetic"`: `"all"` (default), `"tuning"` or `"test"`.
`models`	For `"scored_synthetic"`: score columns to use.

Details

- `"scored_synthetic"`: 200 units (60 events) with two predicted probabilities `model_a` and `model_b` rounded to two decimals (ties occur); `part = "tuning"` and `part = "test"` give two disjoint halves for held-out evaluation.
- `"binormal_equal_variance"`: binormal model with AUC 0.8 and $\sigma = 1$, prevalence 0.1, N = 1000, cutoff at the Youden optimum $d/2$.
- `"binormal_unequal_variance"`: AUC 0.8, $\sigma = 1.5$, prevalence 0.1, N = 1000, cutoff 1.
- `"counts_fixed"`: one fixed 2x2 table (TP 40, FN 10, FP 20, TN 130).

Value

A list with `spec` (a `classification_spec`), `data` (the data frame for scored data, otherwise `NULL`) and `description`.

See Also

- `classification_spec()`
- `evaluate_cutoff()`
- `plot_classification()`

Examples

```
ex <- classification_examples("binormal_unequal_variance")
evaluate_cutoff(ex$spec)
```

`classification_metrics` *Classification metrics from 2x2 counts*

Description

Computes every operating-point metric of the `metric_dictionary()` from the four cells TP, FN, FP and TN. Undefined values are `NA` with a typed status and a reason; mathematical infinity has the status `"positive_infinity"` (value `NA`), never a silent number. PPV at zero alerts is undefined, never 1.

Usage

```
classification_metrics(
  x,
  count_kind = c("observed", "frequency_weighted", "expected"),
  lang = c("en", "de")
)
```

Arguments

`x`	A named numeric vector or list with `tp`, `fn`, `fp` and `tn`, or a `classification_result` (its observed operating-point metrics are returned, one block per model).
`count_kind`	For counts: `"observed"` (whole numbers, the default), `"frequency_weighted"` or `"expected"` (fractional expected counts, for example from `prevalence_scenario()`).
`lang`	Language of the reason texts and denominator labels.

Value

A data frame with the columns `metric_id`, `value`, `status`, `reason_code`, `reason`, `denominator`, `denominator_value`, `unit` and `estimate_kind` (plus `model_id` for results).

See Also

- `classification_spec()`
- `evaluate_cutoff()`
- `plot_classification()`

Examples

```
m <- classification_metrics(c(tp = 40, fn = 10, fp = 20, tn = 130))
m[m$metric_id %in% c("tpr", "ppv", "lr_pos", "mcc"), c("metric_id", "value", "status")]
# Zero alerts: PPV is undefined, not 1
z <- classification_metrics(c(tp = 0, fn = 5, fp = 0, tn = 95))
z[z$metric_id == "ppv", c("value", "status", "reason")]
```

---

classification_plot_data

Plot-ready data for the linked classification panels

Description

Returns the data of one classification panel for a `classification_result` as plain data frames with notes; no plot is drawn (static renderers and the application draw from these tables). All values come from the result, so panels, tables and reports of the same result agree.

Usage

```
classification_plot_data(
  x,
  panel = .ig_cls_panels,
  model_id = NULL,
  basis = c("observed", "scenario"),
  lang = c("en", "de"),
  ppv_levels = c(0.25, 0.5, 0.75),
  f1_levels = c(0.2, 0.4, 0.6, 0.8),
  ppv_target = NULL,
  metric_x = "fpr",
  metric_y = "tpr",
  realised_rules = NULL,
  bins = 40
)
```

Arguments

`x`	A `classification_result`.
`panel`	Panel name (see Details).
`model_id`	Model id (default: the first model).
`basis`	`"observed"` (default) or `"scenario"` (expected counts at the result's scenario prevalence) for prevalence-dependent panels.
`lang`	Label language, `"en"` or `"de"`.
`ppv_levels`	Iso-PPV reference levels of the ROC panel.
`f1_levels`	Iso-F1 reference levels of the PR panel.

`ppv_target` Optional PPV target for the prevalence panel (no default: targets are user decisions).

`metric_x`, `metric_y` Metric ids of the metric-pair panel.

`realised_rules` Optional data frame of realised varying-cutoff rules (`id`, `label`, `tp`, `fn`, `fp`, `tn`) for the ROC, PR and metric-pair panels; they are drawn as separate markers.

`bins` Number of display bins of the score distribution.

Details

Panels: `"confusion_table"` (cells, totals and headline rates with denominators), `"mosaic"` (rectangles for joint and within-class widths), `"score_distribution"` (exact counts per distinct score plus display-only bins; densities for a theoretical model), `"roc"` (attained operating points, current point, chance diagonal, iso-PPV references at a stated prevalence, log-axis boundary counts, realised rules), `"pr"` (attained points without interpolation, prevalence baseline, iso-F1 references), `"metric_vs_cutoff"` (five metric families along the decision sets), `"prevalence_curve"` (PPV and NPV versus prevalence at fixed TPR/FPR), `"metric_pair"` (any two operating-point metrics), `"metric_network"` (metric dependency graph from the dictionary), `"model_comparison"` (ROC curves and operating points of all models evaluated on the same units, with the paired DeLong comparisons; `model_id` is the focal model), `"sweep_table"` and `"current_table"`. Probability-score panels `"calibration"` (retained grouped observed rates and diagnostic summaries) and `"net_benefit"` (explicit decision probabilities with alert-all and alert-none references) use observed data, never prevalence scenarios.

Value

A list with `panel`, `model_id`, `basis`, `data` (named list of data frames) and `notes` (character).

See Also

- `classification_spec()`
- `evaluate_cutoff()`
- `plot_classification()`

Examples

```
ex <- classification_examples("scored_synthetic")
res <- evaluate_cutoff(ex$spec)
roc <- classification_plot_data(res, "roc")
head(roc$data$points)
roc$notes
```

`classification_spec` *Specify a binary classification or cutoff analysis*

Description

Builds an immutable, hashed `classification_spec`. Give exactly one kind of input:

Usage

```
classification_spec(
  data = NULL,
  truth = NULL,
  positive_label = NULL,
  scores = NULL,
  counts = NULL,
  theory = NULL,
  rule = NULL,
  cutoff = NULL,
  models = NULL,
  direction = NULL,
  comparator = NULL,
  score_scale = c("raw_score", "probability"),
  evaluation_split = NULL,
  negative_label = NULL,
  prevalence = NULL,
  prevalence_source = NULL,
  sampling = NULL,
  weights = NULL,
  weight_type = "frequency",
  cluster = NULL,
  id = NULL,
  missing = c("error", "exclude"),
  uncertainty = classification_uncertainty()
)
```

Arguments

`data`	A data frame for scored data.
`truth`	Name of the truth column in `data`.
`positive_label`	The value of `truth` that marks an event. Required: the positive class is never inferred.
`scores`	Character vector of score column names; names (if given) are used as model ids.
`counts`	Named counts `tp`, `fn`, `fp`, `tn` (whole numbers).
`theory`	List with `auc` in (0, 1), `sigma` > 0 and optionally `n_total` (expected-count scale, default 1000, at most 1e100).

`rule` An operating rule from `cutoff_rule()`.

`cutoff` Shorthand for `rule = cutoff_rule(cutoff = cutoff)`.

`models` Optional model metadata: a `classifier_model()` or a list of them, matched to the score columns by `model_id` or position.

`direction` `"higher_is_positive"` (default) or `"lower_is_positive"`; a single value or one per model (named by model id).

`comparator` `">="/">"` (higher is positive) or `"<="/"<"` (lower is positive); default `">="` or `"<="`. An explicit comparator implies the direction.

`score_scale` `"raw_score"` (default) or `"probability"`, shared by all models, or a character vector with one scale per model named by model id. Probability scores are predicted probabilities of the positive class; values must lie in [0, 1]. Brier score, calibration and net benefit need probabilities and the direction `"higher_is_positive"`; otherwise they are reported as not applicable.

`evaluation_split` How the evaluated data relate to model development: `"same_data"` (default, claims least), `"held_out_tuning"`, `"held_out_test"`, `"external"` or `"resampling"`.

`negative_label` Optional value of `truth` that marks a non-event; when given, every other value is an error.

`prevalence` Optional prevalence scenario in [0, 1] (required for theory; strictly between 0 and 1 when a cutoff is selected on a theoretical model). Observed results are kept separately.

`prevalence_source` Provenance of a scenario prevalence: `"assumed"` (default) or `"external"`.

`sampling` How the evaluation sample was drawn: `"cross_sectional"`, `"case_control"` (outcome-selected; the sample prevalence is not a population prevalence) or `"unknown"` (default).

`weights` Optional name of a frequency-weight column.

`weight_type` Only `"frequency"` is supported.

`cluster` Optional name of a cluster column (for example patient id when patients contribute several units).

`id` Optional name of a unit-id column; duplicates are an error.

`missing` `"error"` (default) or `"exclude"`: rows with a missing truth, score, weight or cluster value are dropped for all models alike and counted.

`uncertainty` Settings from `classification_uncertainty()`.

Details

- **Scored data:** `data` with a `truth` column, an explicit `positive_label` and one or more numeric `scores` columns (one model per column; all models are evaluated on the same units).
- **Aggregate counts:** `counts = c(tp =, fn =, fp =, tn =)` for one fixed operating point. No ROC curve or cutoff sweep can be computed from these.

- **Theory:** `theory = list(auc =, sigma =, n_total =)` for the binormal model $S_0 \sim N(0,1)$, $S_1 \sim N(d, \sigma^2)$ with $d = \Phi^{-1}(AUC)\sqrt{1+\sigma^2}$; the cutoff is on the score scale standardised to the non-event distribution and needs a prevalence.

The default direction is "higher score means positive" and is recorded as a default in the model record (`defaults_applied`) so that it can be confirmed; the direction is never chosen from the data. The comparator (`>=`, `>`, `<=`, `<`) fixes which side of the cutoff alerts and what happens to scores equal to the cutoff: under `>=` a cutoff of 1 alerts units with score 1.

Weights are frequency weights only (whole numbers meaning replicated units); survey or importance weights are refused as `unsupported`. With a `cluster` column, binomial and DeLong intervals are not reported and the bootstrap resamples whole clusters.

Value

An immutable object of class `classification_spec`.

See Also

`evaluate_cutoff()`, `select_cutoff()`, `cutoff_sweep()`

- `evaluate_cutoff()`
- `plot_classification()`

Examples

```
# Aggregate counts at one fixed operating point
classification_spec(counts = c(tp = 40, fn = 10, fp = 20, tn = 130), cutoff = 0.5)

# Scored data (synthetic)
d <- data.frame(y = c(1, 1, 1, 0, 0, 0, 0, 0), risk = c(.9, .8, .4, .7, .3, .2, .2, .1))
spec <- classification_spec(d, truth = "y", positive_label = 1, scores = "risk",
                            cutoff = 0.5, score_scale = "probability")
spec

# Binormal theory: AUC 0.8, unequal variance, 10 % prevalence
classification_spec(theory = list(auc = 0.8, sigma = 1.5, n_total = 1000),
                    cutoff = 1, prevalence = 0.1)
```

---

classification_table *Tables of a classification result*

Description

Returns one table of a `classification_result` as a data frame, with one block per model where the table is per model. Reason texts and denominator labels are translated with `lang`.

Usage

```
classification_table(
  x,
  table = c("metrics", "intervals", "summaries", "operating_point", "curve", "selection",
    "bounds", "tied", "scenario", "calibration", "net_benefit", "comparisons",
    "limitations"),
  model_id = NULL,
  lang = c("en", "de")
)
```

Arguments

x	A classification_result (also one rebuilt from JSON).
table	One of "metrics" (all operating-point metrics with status, reason, denominator, unit and estimate kind), "intervals", "summaries" (ROC AUC, partial AUC, average precision, trapezoidal PR area, Brier score and calibration), "operating_point", "curve" (decision sets or theoretical display grid), "selection", "bounds" (constraint feasibility), "tied" (tied operating points), "scenario" (prevalence-scenario metrics), "calibration", "net_benefit", "comparisons" or "limitations".
model_id	Optional model id(s) to keep.
lang	"en" or "de".

Value

A data frame.

See Also

- classification_spec()
- evaluate_cutoff()
- plot_classification()

Examples

```
res <- evaluate_cutoff(classification_spec(counts = c(tp = 40, fn = 10, fp = 20, tn = 130)))
classification_table(res, "intervals")[1:4, c("metric_id", "method", "lower", "upper")]
```

---

classification_uncertainty

Uncertainty settings of a classification analysis

Description

Intervals are conditional on the scores and, for operating-point metrics, on the fixed cutoff. They do not include the uncertainty of a data-driven cutoff selection.

Usage

```
classification_uncertainty(
  level = 0.95,
  proportion_methods = c("wilson", "clopper_pearson"),
  auc_method = c("delong", "none"),
  bootstrap_replicates = 0,
  seed = NULL,
  bootstrap_strata = c("class", "none"),
 bootstrap_metrics = c("tpr", "fpr", "ppv", "npv", "youden_j", "f1", "mcc", "roc_auc",
    "average_precision")
)
```

Arguments

`level`	Confidence level in (0, 1).
`proportion_methods`	Binomial interval methods for proportions: any of `"wilson"` and `"clopper_pearson"`, or `character(0)` for none.
`auc_method`	`"delong"` (asymptotic interval, own implementation) or `"none"`.
`bootstrap_replicates`	Number of bootstrap replicates (0 = no bootstrap). The bootstrap needs a `seed`.
`seed`	Integer seed for the bootstrap. The global random-number state is restored afterwards.
`bootstrap_strata`	`"class"` (resample events and non-events separately, the default) or `"none"`. With declared clusters, whole clusters are resampled instead.
`bootstrap_metrics`	Metric ids that receive percentile bootstrap intervals: operating-point metrics (evaluated at the fixed cutoff in every replicate) and the curve summaries `roc_auc`, `partial_auc`, `average_precision`, `pr_area_trapezoid` and `brier_score`.

Details

Bootstrap limits use type-7 empirical quantiles. Undefined replicates are excluded only when at most 10 percent are undefined; otherwise the interval is explicitly unavailable. Class stratification conditions on the observed class totals, so it does not estimate population-prevalence uncertainty. Frequency weights represent replicated units; declared clusters are sampled as whole clusters. These are percentile intervals, without bias correction or studentisation. Their nominal level is not a coverage guarantee: small samples and boundary estimates can have severe undercoverage. Independent arithmetic checks and a binomial coverage diagnostic are provided in `tools/product/check-bootstrap.R`; bootstrap validation remains development.

Value

A list of class `ig_cls_uncertainty`.

See Also

- classification_spec()
- evaluate_cutoff()
- plot_classification()

Examples

```
classification_uncertainty(level = 0.9, bootstrap_replicates = 500, seed = 1)
```

---

classifier_model *Model metadata for a classification specification*

Description

Describes one already-scored model (or test). inferguidR does not train models and never deserialises model objects; it evaluates the scores you supply.

Usage

```
classifier_model(
  model_id,
  name = model_id,
  version = NULL,
  score_meaning = NULL,
  population = NULL,
  unit = NULL,
  source_hash = NULL
)
```

Arguments

model_id	Stable identifier (letters, digits, _, ., -).
name	Display name.
version	Optional model version string.
score_meaning	What the score is (for example "predicted 30-day risk"). Defaults depend on the input kind.
population	Optional description of the population the scores come from.
unit	Optional unit noun of an evaluation unit (for example "patient" or "order").
source_hash	Optional SHA-256 (64 lower-case hex characters) of the source data set or model artefact.

Value

A list of class ig_classifier_model.

See Also

- classification_spec()
- evaluate_cutoff()
- plot_classification()

Examples

```
classifier_model("logit_v2", name = "Logistic model", version = "2.0",
                 score_meaning = "predicted probability of readmission")
```

---

compare_classifiers *Compare scored models on the same units*

Description

Pairwise comparison of the ROC AUC of models evaluated on the same units (paired design), using the DeLong covariance of the placement values (own implementation). Exclusions of missing values are matched across models by classification_spec().

Usage

```
compare_classifiers(spec, level = NULL)
```

Arguments

spec	A scored-data classification_spec with at least two models and attached data.
level	Confidence level (default: the spec's uncertainty level).

Value

A data frame with one row per model pair: model_a, model_b, auc_a, auc_b, difference, se, lower, upper, z, p_value, status, reason_code, reason, method.

See Also

- classification_spec()
- evaluate_cutoff()
- plot_classification()

Examples

```
d <- data.frame(y = rep(c(1, 0), each = 6),
                a = c(8, 7, 7, 5, 4, 2, 6, 3, 3, 2, 1, 1),
                b = c(5, 6, 4, 4, 3, 3, 5, 4, 2, 2, 2, 1))
spec <- classification_spec(d, truth = "y", positive_label = 1, scores = c("a", "b"),
                            cutoff = 4)
compare_classifiers(spec)
```

compare_designs *Compare several designs*

Description

Computes (where needed) and tabulates several analyses side by side. Specifications are solved with `calculate_power()`; existing results are used as they are. Results that are not `"ok"` keep their status and reason and contribute no numbers.

Usage

```
compare_designs(..., designs = list())
```

Arguments

`...`	`power_spec` or `power_result` objects, optionally named.
`designs`	A list of further `power_spec` or `power_result` objects.

Value

A data frame with one row per design and columns `design`, `method_id`, `mode`, `target`, `status`, `reason`, `achieved_power`, `achieved_beta`, `alpha_nominal`, `alpha_attained`, `n_total`, `df`, `ncp`, `spec_hash`, `result_hash` and the list columns `solved`, `group_sizes`, `critical` and `result`.

See Also

`sweep_design()`

Examples

```
base <- analysis_spec("t_means_two_independent", "post_hoc",
  inputs = list(alpha = 0.05, n1 = 30, n2 = 30, d = 0.5), options = list(tails = "two"))
compare_designs(equal = base, unequal = update(base, inputs = list(n1 = 20, n2 = 40)))[,
  c("design", "status", "achieved_power", "n_total")]
```

create_report_snapshot *Freeze a portable analysis report*

Description

Creates an immutable report document from matching specifications and results. All export formats use this same frozen content. The portable document omits attached classifier rows and score-derived curve tables. Aggregate-cell mosaics and prevalence scenarios retain their plotted values; undefined conditional rates remain explicitly unavailable. Selected ROC and precision-recall points use only retained aggregate cells; undefined denominators remain unavailable. Prevalence-scenario precision is labelled separately and is not a new observed measurement. These selected points do not reconstruct full curves or imply an AUC from one cutoff. Completed Boschloo power reports retain the full possible-table lattice when it fits the 5000-table display limit, separately for H0 and H1. The frozen rejection set is checked against the result's region hash; import and rendering do not rerun the region solver. Lattice reports use schema 0.4 and retain an explicit unavailable display above the limit. Original private scored data must be reattached for recomputation. User-supplied titles and metadata are included as literal text. Observed Boschloo reports retain independent-group counts and numerical optimisation bounds; those bounds are not sampling confidence intervals.

Usage

```
create_report_snapshot(
  envelopes,
  recommendation = NULL,
  report_kind = "single_analysis",
  title = NULL,
  locale = "en",
  display_timezone = "UTC",
  sweeps = list(),
  resources = list(),
  include_probability_summaries = FALSE
)
```

Arguments

envelopes	An analysis_envelope or a nonempty list of envelopes.
recommendation	Optional advisor recommendation.
report_kind	One of single_analysis, model_cutoff, comparison, session or incomplete_settings.
title	Report title; defaults to an analysis report title.
locale	Label language, en or de.
display_timezone	Display timezone recorded with the UTC timestamp.
sweeps	Optional list of entries with analysis (one-based envelope index) and sweep (a sweep_design() result). At most20sweeps and1000rows per sweep are retained; requested values must be finite. Each sweep must match its completed frozen base. Failed rows require report_kind = "incomplete_settings". Specifications, results, failures and individually evaluated points are retained without interpolation. Documents with sweeps use schema0.2; existing schema0.1 documents remain readable.

`resources` Optional list of entries with analysis (one-based index) and resource_id. Currently inferguidR_boschloo_minlike is available only for the exact minlike observed test or power variant. Frozen titles, scope, file hashes and source references use a versioned bundled catalog. No resource code is executed and no website is requested. Attached resources use report schema0.3.

`include_probability_summaries`
Logical, FALSE by default. Explicitly include frozen calibration-bin counts and means, and aggregates at the explicitly requested net-benefit thresholds, with point figures. Small groups may disclose sensitive information; review before sharing. No individual rows, score ranges, full ROC/PR curves or density tables are included. Opted-in documents use schema 0.4; older documents remain readable.

Details

Comparison scope accepts separate analysis envelopes or multiple classifier models in one completed envelope. Paired ROC-AUC interval figures use only retained comparison aggregates, with marginal intervals and no multiplicity adjustment. Missing intervals are not replaced or clipped.

Value

A report_document inheriting from report_snapshot, suitable for export_report() and safe ig_to_json()/ig_from_json() round trips.

---

cutoff_rule *Operating rule of a classification analysis*

Description

Declares how the cutoff (operating point) is obtained: a manual cutoff, or a selection objective with optional constraints. Policy thresholds are never built in; every constraint is a user decision.

Usage

```
cutoff_rule(
  cutoff = NULL,
  selection = NULL,
  selection_data = NULL,
  constraints = NULL,
  optimise = NULL,
  capacity = NULL,
  costs = NULL,
  tie_break = NULL,
  prevalence_basis = NULL
)
```

Arguments

cutoff	Numeric score cutoff for a manual rule. It is a classifier score threshold, never a significance critical value.
selection	"manual" (default when cutoff is given), "max_youden", "max_f1", "max_mcc", "constraint" (best optimise value subject to constraints), "capacity" (most alerts within capacity) or "cost_utility" (smallest declared expected cost).
selection_data	Where the cutoff comes from relative to the evaluated data: "same_data" (chosen on these data; the default because it claims least), "held_out_tuning" (chosen on separate tuning data) or "predeclared_rule" (fixed before any data were seen; manual rules only).
constraints	Optional named list of bounds: tpr_min, tnr_min, ppv_min, npv_min, fpr_max, fnr_max, fdr_max, alert_rate_max (values in [0, 1]). An undefined metric never satisfies a bound. With a selection objective the bounds restrict the admissible operating points; with a manual cutoff they are only checked and reported.
optimise	For selection = "constraint": the metric id to optimise among feasible operating points (default "tpr"), in its preferred direction from metric_dictionary().
capacity	For selection = "capacity": list(max_alerts = k) or list(max_alert_rate = r).
costs	For selection = "cost_utility": named numeric costs per unit for fn and fp (required) and optionally tp and tn (negative values are benefits).
tie_break	Rule that chooses among operating points with the same objective value: "fewest_alerts" (default) or "most_alerts". All tied operating points are reported.
prevalence_basis	Counts on which a selection objective and the constraints are evaluated: "observed" (the evaluated sample; default for scored data) or "scenario" (expected counts at the specification's scenario prevalence under fixed class-conditional score distributions; the only basis of a theoretical model). It matters for prevalence-dependent quantities such as PPV, F1, MCC or costs.

Value

A list of class ig_cutoff_rule.

See Also

- classification_spec()
- evaluate_cutoff()
- plot_classification()

Examples

```
cutoff_rule(cutoff = 0.3, selection_data = "predeclared_rule")
cutoff_rule(selection = "constraint", constraints = list(tpr_min = 0.9))
cutoff_rule(selection = "max_youden", constraints = list(fpr_max = 0.2))
cutoff_rule(selection = "cost_utility", costs = c(fn = 10, fp = 1))
```

`cutoff_sweep` *Sweep all distinct decision sets of a scored model*

Description

Enumerates every distinct empirical decision set (alert none, alert all and every set in between) from the original scores, with exact counts and the requested metrics. The cutoff of each set is an original score value under the declared comparator; a set that no score can represent (alert none under >=, alert all under >) has cutoff NA and a `cutoff_kind` that says so. For a theoretical model the sweep is a display grid of real-valued cutoffs with expected counts. Aggregate 2x2 counts cannot be swept.

Usage

```
cutoff_sweep(
  x,
  metrics = .ig_cls_default_sweep_metrics,
  model_id = NULL,
  grid = 401,
  basis = c("observed", "scenario")
)
```

Arguments

`x`	A `classification_spec` (scored data with attached data, or theory) or a `classification_result` that holds decision sets.
`metrics`	Metric ids (see `metric_dictionary()`) to add as columns; each value column has a companion `<id>_status` column.
`model_id`	Optional model id (default: all models).
`grid`	For theoretical models: number of display cutoffs (default 401) or a numeric vector of cutoffs.
`basis`	For results: `"observed"` (default) or `"scenario"` (expected counts of every decision set at the result's scenario prevalence, under fixed class-conditional score distributions).

Value

A data frame with `model_id`, `set_index`, `cutoff`, `cutoff_kind`, `alerts`, `tp`, `fn`, `fp`, `tn`, `count_kind` and the metric columns; for results also `selected` (the reported operating point, always included).

See Also

- `classification_spec()`
- `evaluate_cutoff()`
- `plot_classification()`

Examples

```
d <- data.frame(y = c(1, 1, 1, 0, 0, 0), s = c(5, 4, 4, 4, 2, 1))
spec <- classification_spec(d, truth = "y", positive_label = 1, scores = "s", cutoff = 4)
cutoff_sweep(spec, metrics = c("tpr", "fpr", "ppv"))
```

---

distribution_calculator

Distribution calculator

Description

Evaluates a script in the syntax of the 'G*Power' 3.1.9.7 calculator: arithmetic with `+ - * / ^`, assignments, several statements separated by `;` or line breaks, `#` comments, and the mathematical and distribution functions listed by `calculator_functions()` (central and noncentral t, F and chi-square, normal, beta, binomial, Poisson, hypergeometric, sample correlation, squared multiple correlation, log-normal, Laplace, exponential and uniform CDFs, densities and quantiles). The result is the value of the last statement.

Usage

```
distribution_calculator(expression)
```

Arguments

`expression`	A single string: the calculator script.

Details

The script is never evaluated as R code. It is parsed by an allowlisted parser; only numbers, variables, the five operators and the listed functions exist.

Syntax conventions follow 'G*Power' (captured from the running program): `^` is left-associative (`2^3^2` is 64) and a leading minus binds tighter than `^` (`-2^2` is 4, `2^-1` is 0.5); there is no unary `+`; names are letters and digits and are case-sensitive; there are no predefined constants; function arguments are separated by `,`. Where 'G*Power' computes something other than the documented function (for example `chi2pdf`, `ncx2pdf` and `unipdf`), the documented function is computed; see the `gpower_note` column of `calculator_functions()`.

Large sine, cosine and tangent arguments use a bundled high-precision argument reduction before evaluating a small angle, avoiding dependence on platform-specific large-argument reduction in the system math library.

Value

An object of class `ig_calc_result`: a list with `expression`, `status` (an `ig_status()`: `ok`; `invalid_input` for syntax and math errors; `unsupported` for valid parameters beyond the range the calculator computes, for example a noncentrality parameter above 1e8; or `numerical_failure`), `value` (the result, `NA` unless the status is `ok`; `Inf`, `-Inf` and `NaN` are results of IEEE arithmetic as

in 'G*Power'), `display` (the value as the 'G*Power' result field shows it: 15 significant digits, `inf`, `-inf`, `-nan(ind)`), `statements` (a data frame with one row per statement), `variables`, `functions_used`, `notes` (non-finite intermediate results and reference notes of the functions used) and `r_call`.

See Also

`calculator_functions()`

Examples

```
distribution_calculator("1 - normcdf(1.96, 0, 1)")
distribution_calculator("N = 16; tc = tinv(0.95, N - 1); 1 - nctcdf(tc, N - 1, 2)")$value
distribution_calculator("2^3^2")$value   # 64: '^' is left-associative
```

---

effect_size *Compute an effect size from raw inputs*

Description

Effect-size drawers turn descriptive quantities (for example group means and standard deviations) into the effect-size parameter of a method, as the "Determine" dialog of 'G*Power' does. The formula, its source and any caveats are returned with the value.

Usage

```
effect_size(method, input_mode, ..., inputs = list())
```

Arguments

`method`	Method id (see `list_methods()`).
`input_mode`	Drawer input mode (see Details).
`...`	Named drawer inputs.
`inputs`	Named list of drawer inputs (alternative to `...`; a name given in both places must carry the same value).

Details

Available drawers and input modes:

- `"t_means_two_independent"` (effect d):
  - `"common_sd"`: `mean_1`, `mean_2`, `sd` (one SD common to both groups), d = (mean_1 - mean_2) / sd;
  - `"two_sds"`: `mean_1`, `mean_2`, `sd_1`, `sd_2`, with the mean SD sqrt((sd_1^2 + sd_2^2) / 2) (intended for equal group sizes);
  - `"point_biserial_r"`: `r`, `n1`, `n2`, d = r / sqrt(1 - r^2) * (n1 + n2) / sqrt(n1 * n2) (the hand conversion of the 'G*Power' manual, section 16.3).

- "t_means_one_sample": "from_means" takes mean_h0, mean_h1, sd.
- "t_means_matched_pairs": "from_differences" takes mean_diff, sd_diff; "from_group_parameters" takes mean_1, mean_2, sd_1, sd_2, rho (the within-pair correlation).
- "t_wilcoxon_signed_rank_one_sample": "means_sd" takes mean_h0, mean_h1, sd.
- "t_wilcoxon_signed_rank_matched_pairs": "differences" takes mean_diff, sd_diff; "group_parameters" takes mean_1, mean_2, sd_1, sd_2, r (the within-pair correlation).
- "t_wilcoxon_mann_whitney": "common_sd" takes mean_1, mean_2, sd; "two_sds" takes mean_1, mean_2, sd_1, sd_2.

Repeated-measures drawers:

- "f_rm_anova_between", "f_rm_anova_within", "f_rm_anova_within_between", "f_rm_manova_between" and "f_rm_manova_within" offer "direct" (partial_eta2), "from_variances" (variance_effect, variance_within) and "convert". The two between-subjects methods additionally offer "from_means" (means, sd, optional group sizes n).
- "convert" takes f, from, to. ANOVA convention codes are "gpower30", "gpower30_implicit_rho", "spss", "cohen1988"; MANOVA codes are "on" and "off" for using the mean correlation. Supply measurements and rho for "gpower30" or "on", and measurements, n_total, groups for "spss". This conversion preserves noncentrality and is an extension of the reference application's drawer modes.
- "f_rm_manova_within_between" offers "from_criterion" (criterion, approximation, groups, measurements, optional n_total) and "from_means_cov" (a group-by-measurement matrix means, approximation, and either covariance matrix sigma or sd with correlation matrix corr or common correlation rho; optional group sizes n and n_total). Approximation codes are "pillai_v", "wilks_u", "hotelling_lawley_t1" and "hotelling_lawley_t2".

Use the full drawer input names, particularly groups and measurements. R can partially match names such as m or i to the formal arguments method or input_mode; alternatively supply drawer values in inputs. The reference application's mean-based t-test drawers display magnitudes, whereas its matched-pairs "From differences" drawer retains the sign.

The returned value is signed as defined by the method (for two independent means d = (mean_1 - mean_2) / sigma); magnitude is its absolute value.

Value

An object of class ig_effect_size: a list with method_id, input_mode, status (an ig_status()), effect (the parameter name), value, magnitude, formula, source, inputs and warnings, plus named lists outputs (derived quantities) and details (supporting calculations), empty when the drawer supplies none. A method without a drawer gives status "unsupported" and value = NA. Invalid inputs signal an inferguidR_invalid_input condition.

See Also

analysis_spec()

Examples

```
es <- effect_size("t_means_two_independent", "two_sds",
  mean_1 = 0, mean_2 = 1, sd_1 = 0.5, sd_2 = 0.5)
es$value
es$magnitude
```

`evaluate_cutoff` *Evaluate an operating rule (cutoff) of a classification specification*

Description

Applies the operating rule of `spec` to every model and returns an immutable `classification_result`. For scored data the counts come from the original scores under the declared comparator (never from histogram bins); every distinct decision set is enumerated, so ROC/PR summaries and sweeps are exact. A selection rule in `spec` (for example `"max_youden"`) is carried out on the evaluated data and labelled in-sample; see `select_cutoff()`.

Usage

```
evaluate_cutoff(
  spec,
  cutoff = NULL,
  model_id = NULL,
  partial_auc = NULL,
  threshold_probability = NULL,
  calibration_bins = 10,
  include_curves = TRUE
)
```

Arguments

`spec`	A `classification_spec`.
`cutoff`	Optional override: a number (a manual cutoff for all models), or a `classification_result` computed on *other* (tuning) data whose selected cutoff is then evaluated here as a held-out rule. Shared supplied unit or cluster identifiers are refused when both live specifications retain their data. Imported source results cannot supply this identity check. Identity absence is recorded; disjoint identifiers never prove independence or rule out model-training leakage. Identifiers must share the same meaning across splits; none are exported by this check.
`model_id`	With a result as `cutoff`: the model whose cutoff is used (needed when the result has several models).
`partial_auc`	Optional partial AUC range, `list(focus = "fpr" | "tpr", range = c(lo, hi))`.
`threshold_probability`	Optional decision threshold probabilities for net benefit (probability scores only); see `net_benefit()`.

`calibration_bins`
: Number of quantile groups of the calibration table.

`include_curves`
: Store the decision sets (or the theoretical display grid) in the result (needed for sweeps and curve plots).

Details

The result holds, per model: the operating point (cutoff, comparator, TP/FN/FP/TN and totals with their count kind), every operating-point metric of the `metric_dictionary()` with status, reason, denominator and unit, Wilson and Clopper-Pearson intervals (conditional on the fixed cutoff), the ROC AUC with its own DeLong interval, optional partial AUC, average precision and the trapezoidal PR area as distinct quantities, optional bootstrap intervals, calibration and net benefit for probability scores, a prevalence scenario when one is declared, the decision sets (or the theoretical display grid) and the availability of each view. It also states how the cutoff was chosen relative to the data and lists the limitations that apply.

Value

A `classification_result`; see `classification_table()` for its tables. Its status is `"ok"`, or `"infeasible"` when a selection rule has no admissible operating point (no number is reported then). The specification it was evaluated with (which differs from `spec` after a cutoff override or a transferred rule) is returned by `classification_evaluated_spec()`.

See Also

`select_cutoff()`, `classification_table()`, `cutoff_sweep()`, `prevalence_scenario()`

- `classification_spec()`
- `plot_classification()`

Examples

```
# Fixed 2x2 table
res <- evaluate_cutoff(classification_spec(counts = c(tp = 40, fn = 10, fp = 20, tn = 130)))
res

# Scored data, manual cutoff, partial AUC over FPR 0-0.2
ex <- classification_examples("scored_synthetic")
r2 <- evaluate_cutoff(ex$spec, cutoff = 0.3, partial_auc = list(focus = "fpr", range = c(0, 0.2)))
classification_table(r2, "summaries")

# Rule chosen on tuning data, evaluated on test data
tune <- classification_examples("scored_synthetic", part = "tuning", models = "model_a")$spec
test <- classification_examples("scored_synthetic", part = "test", models = "model_a")$spec
chosen <- select_cutoff(tune, "max_youden")
held_out <- evaluate_cutoff(test, cutoff = chosen)
held_out$evaluation$label
```

`export_report` *Export a frozen report as PDF, JSON or UTF-8 text*

Description

Export a frozen report as PDF, JSON or UTF-8 text

Usage

```
export_report(
  snapshot,
  file,
  format = c("pdf", "json", "txt"),
  overwrite = FALSE
)
```

Arguments

`snapshot`	A report_document from create_report_snapshot().
`file`	Destination file. Existing files are preserved unless overwrite is TRUE.
`format`	One of pdf, json or txt.
`overwrite`	Allow replacement of an existing file.

Details

Rendering uses a temporary file in the destination directory. The completed file is checked and hashed before a single checked rename replaces the destination. Rendering or failed-rename errors leave an existing destination intact; staging files are removed on exit. This follows the filesystem's same-directory rename semantics, including on Windows, and does not promise crash durability or concurrent-writer isolation. Explicit replacement replaces a symbolic link itself, leaving its target untouched; a dangling link also requires overwrite = TRUE. JSON and TXT remain available if the offline Cairo PDF capability is unavailable. Text is rendered literally; report content is never evaluated as R or markup. PDF and TXT begin with a readable summary rounded to six significant digits; stored decisions and the technical appendix retain full precision. PDF figures precede the appendix and have adjacent captions. Very long plot labels may be abbreviated with `[...]` while full labels remain in the appendix. PDF text wrapping uses measured font widths.

Value

Invisibly, the destination and its SHA-256 hash with the snapshot ID.

format_metric_values *Format classification metric values for display*

Description

Adds display strings to a table of classification metrics without changing its values. Finite values are formatted by unit: proportions as percentages, counts as whole numbers (expected counts with an approximately-equal sign and one decimal, because they are not observed units), ratios with three significant digits (whole numbers from 100 on), rates per 1,000 with one decimal and all other quantities with three decimals. Positive infinity is shown as an infinity sign. An undefined, not applicable or not computed value is shown as a dash, and its reason is always given in `display_reason`: a dash is never left unexplained. German output uses a decimal comma.

Usage

```
format_metric_values(x, lang = c("en", "de"), percent = TRUE, digits = NULL)
```

Arguments

`x`	A data frame with `metric_id`, `value` and `status`, and optionally `reason_code`, `reason`, `unit` and `estimate_kind`, as returned by `classification_metrics()`, `classification_table()` (tables `"metrics"`, `"scenario"` and `"summaries"`) or `prevalence_scenario()`.
`lang`	`"en"` or `"de"`.
`percent`	Show proportions as percentages (default `TRUE`); otherwise with three decimals.
`digits`	Optional fixed number of decimals (significant digits for ratios) instead of the magnitude-dependent default.

Details

Formatting never feeds a decision: cutoff selection, constraints and every export use the unrounded values.

Value

`x` with the added columns `display` and `display_reason`.

See Also

- `classification_spec()`
- `evaluate_cutoff()`
- `plot_classification()`

Examples

```
m <- format_metric_values(classification_metrics(c(tp = 0, fn = 5, fp = 0, tn = 95)))
m[m$metric_id %in% c("tn", "tpr", "ppv", "lr_neg"), c("metric_id", "display", "display_reason")]
sc <- prevalence_scenario(c(tp = 8, fn = 2, fp = 9, tn = 81), prevalence = 0.02)
format_metric_values(sc, lang = "de")[1:4, c("metric_id", "display")]
```

---

ig_config *Current inferguidR configuration*

Description

Lists the R options that inferguidR reads, with their current values.

Usage

```
ig_config()
```

Details

- `inferguidR.reference_archive_dir`: path of a local reference archive (development use; default unset, i.e. `NULL`). It is never needed for calculations.
- `inferguidR.validate_json_schema`: if `TRUE` (default), `ig_from_json()` validates documents against the bundled JSON Schemas whenever the 'jsonvalidate' package is installed.

Value

A named list of option values (`NULL` for unset options).

Examples

```
ig_config()
```

---

ig_domain *Parameter domains*

Description

Domain descriptors used by `ig_param()`. A domain checks the type, the length and the admissible range of an input value.

Usage

```
ig_domain_real(
  lower = -Inf,
  upper = Inf,
  lower_inclusive = TRUE,
  upper_inclusive = TRUE,
  length = 1L,
  scan = NULL
)

ig_domain_integer(lower = -Inf, upper = Inf, length = 1L)

ig_domain_logical(length = 1L)

ig_domain_choice(values, length = 1L)
```

Arguments

`lower`, `upper`	Bounds of the admissible range. Use `-Inf`/`Inf` for an unbounded side.
`lower_inclusive`, `upper_inclusive`	Whether the bound itself is admissible.
`length`	Required vector length; `NA` accepts any length of at least one.
`scan`	Optional two positive finite distances, in increasing order, for the numerical effect scan of an unbounded real domain. Distances are measured from its finite bound, or from zero when both sides are unbounded. This controls the search, not admissible input values.
`values`	Allowed character codes (for `ig_domain_choice()`).

Value

An object of class `ig_domain`.

See Also

Other method definition API: `ig_method()`, `ig_option()`, `ig_options()`, `ig_param()`, `ig_params()`, `ig_standard_params`

Examples

```
ig_domain_real(0, 1, lower_inclusive = FALSE, upper_inclusive = FALSE)
ig_domain_integer(lower = 2)
```

ig_from_json *Read inferguidR objects from JSON safely*

Description

Parses JSON produced by `ig_to_json()` and rebuilds the object through the same validators that construct it in R. Imported text is treated as data: nothing is evaluated, URLs are never fetched, duplicate keys are rejected, and unknown object types or schema versions are refused with a typed `invalid_input` error. When the 'jsonvalidate' package is installed and `validate` is `TRUE`, the document is also checked against the bundled JSON Schema.

Usage

```
ig_from_json(
  json = NULL,
  file = NULL,
  type = NULL,
  validate = getOption("inferguidR.validate_json_schema", TRUE)
)
```

Arguments

`json`	A JSON string (or `NULL` when `file` is given).
`file`	Path of a local JSON file (alternative to `json`).
`type`	Optional expected object type (`"power_spec"`, `"power_result"`, `"analysis_envelope"`, `"report_snapshot"`, `"recommendation"`).
`validate`	Validate against the JSON Schema when possible. Defaults to the JSON-schema validation option described in `ig_config()`.

Value

The rebuilt object. Errors are conditions of class `inferguidR_invalid_input` carrying an `ig_status()` in `$status`.

See Also

`ig_to_json()`

Other serialisation: `ig_hash()`, `ig_to_json()`

Examples

```
spec <- analysis_spec("t_means_one_sample", "post_hoc",
  inputs = list(alpha = 0.05, n_total = 20, d = 0.5),
  options = list(tails = "two"))
identical(ig_from_json(ig_to_json(spec)), spec)
```

ig_hash *Content hash of an inferguidR object*

Description

Returns the SHA-256 identity hash of an object: the specification hash of a `power_spec`, the result hash of a `power_result`, the analysis hash of an envelope or report snapshot, and the recommendation hash of a recommendation. For plain data it is the SHA-256 of the canonical JSON.

Usage

```
ig_hash(x)
```

Arguments

`x`	An inferguidR object or plain data.

Value

A 64-character lower-case hexadecimal string.

See Also

Other serialisation: `ig_from_json()`, `ig_to_json()`

Examples

```
ig_hash(list(a = 1, b = "x")) == ig_hash(list(b = "x", a = 1))
```

ig_method *Define a statistical method*

Description

Creates and validates a method definition. Each definition corresponds to one procedure of the reference inventory (its `id` must be a registry method id) and is registered when the package loads.

Usage

```
ig_method(
  id,
  family,
  label,
  params,
  options = ig_options(),
  modes,
```

```
  power,
  design = NULL,
  discrete = FALSE,
  effect_direction = "unknown",
 engine = list(package = NA_character_, fun = NA_character_, version = NA_character_),
  checker = NA_character_,
  reference = list(),
  status = "development",
  targets = NULL,
  constraints = NULL,
  a_priori_search = NULL,
  sensitivity_search = NULL,
  sensitivity_domain = NULL
)
```

Arguments

id	Method id, as in the registry (see list_methods()).
family	Test family id ("exact", "F", "t", "z", "chi2").
label	Display label: a list with en and optionally de.
params	An ig_params() object. It must declare alpha and power, every solve target, and error_ratio when the compromise mode is offered.
options	An ig_options() object with the result-affecting options.
modes	Offered analysis modes, a subset of "a_priori", "post_hoc", "sensitivity", "criterion" and "compromise".
power	Forward model function(p); p is the fully specified named list of inputs and options. It returns a named list with at least power and alpha_attained; undefined values are NA with a reason. Optional elements: beta (computed directly, not as 1 - power), critical, df, ncp, n_total, group_sizes, statistic_distribution, outputs (named list), notes (reasons for NA values) and assumptions; see the forward-model contract in R/num-modes.R.
design	Allocation rule function(p, n_total) returning integer group sizes named by their parameters (for example c(n1 = 10, n2 = 10)), or NULL when n_total has no admissible design (for example an odd N under equal allocation). NULL as the argument value means a single-sample design (N is the design). The rule may accept an optional argument continuous = TRUE returning fractional group sizes, used only as the starting point of the a priori search, and may signal a typed condition (for example reference_pending) for an unsupported branch.
discrete	TRUE for exact or otherwise discrete tests.
effect_direction	Monotone direction of power in the effect: "increasing", "decreasing" or "unknown", or a function function(p) returning one of those strings from the validated inputs and options.
engine	Backend provenance: a list with package, fun and version (use NA for version when it is taken at run time).

checker
: Identifier of the independent checker (a path under `validation/independent/`), or `NA`.

reference
: Reference metadata, for example `list(gpower_path = "...", manual_pages = "...")`. A status of `"gpower_validated"` requires `reference$fixture_ids` naming authentic application-capture fixture records.

status
: Validation status: `"development"`, `"independently_checked"` (requires a checker), `"reference_pending"` or `"gpower_validated"`.

targets
: Optional named list of solve targets per mode (character vectors; the first element is the default). Defaults: `n_total` (a priori), `power` (post hoc), the single effect-role parameter (sensitivity), `alpha` (criterion) and the joint pair `alpha, power` (compromise).

constraints
: Optional `function(p)` returning a character vector of cross-parameter inconsistencies (empty when consistent), for example group sizes that do not add up to `n_total`. A callback may additionally declare `mode` and/or `target` arguments to receive the validated analysis mode and selected solve target(s). Existing one-argument callbacks work unchanged.

a_priori_search
: Optional method-specific search of the a priori mode, `function(p, n_lo, n_max)`, used instead of the generic integer search (for example a discrete test that can exclude designs by a proven bound). `p` holds the specification's inputs and options, and `[n_lo, n_max]` is the admissible range of `n_total`. Return a list with `status = "ok"`, `n` (the smallest admissible total reaching the target), `power` (its power), and a `diagnostics` list. When there is no answer, return a list with `status` (the status code), `reason` (explanatory text) and `diagnostics`. The solver checks the returned design against `design` and recomputes its power with `power`.

sensitivity_search
: Optional `function(p, target_name)` replacing the generic sensitivity scan. Return a status/reason for failure, or a list with `status = "ok"`, `value` (the effect), a `diagnostics` list, and a character vector of `warnings`. The effect must be in its parameter domain and its recomputed power must reach the target within solver tolerance. A method-specific maximum below target is allowed only with `diagnostics$target_reached = FALSE`, `diagnostics$max_power` agreeing with recomputed power, and a non-empty explanatory warning.

sensitivity_domain
: Optional `function(p)` returning a scalar real `ig_domain_real()` domain for the generic sensitivity scan. It must be contained in the effect parameter domain. This restricts the search using validated inputs/options, for example to admissible correlations.

Value

An object of class `ig_method`.

See Also

Other method definition API: `ig_domain`, `ig_option()`, `ig_options()`, `ig_param()`, `ig_params()`, `ig_standard_params`

`ig_option` *Declare a result-affecting option*

Description

Options are discrete, result-affecting choices identified by machine codes (for example `tails = "two"`). Presentation settings are not options and do not belong in a specification. An option without a default must be supplied whenever it applies.

Usage

```
ig_option(
  name,
  values,
  default = NULL,
  labels = NULL,
  applies_when = NULL,
  label = NULL
)
```

Arguments

`name`	Option name.
`values`	Allowed codes.
`default`	Default code, or `NULL` when the value must be supplied.
`labels`	Optional display labels, named by code.
`applies_when`	Optional applicability condition (see `ig_param()`).
`label`	Display label, a list with `en` and optionally `de`.

Value

An object of class `ig_option`.

See Also

Other method definition API: `ig_domain`, `ig_method()`, `ig_options()`, `ig_param()`, `ig_params()`, `ig_standard_params`

Examples

```
ig_option("tails", values = c("one", "two"))
```

ig_options *Collect method options*

Description

Collect method options

Usage

```
ig_options(...)
```

Arguments

`...`	`ig_option()` objects (or lists of them). May be empty.

Value

An object of class `ig_options`: a list named by option name.

See Also

Other method definition API: `ig_domain`, `ig_method()`, `ig_option()`, `ig_param()`, `ig_params()`, `ig_standard_params`

Examples

```
ig_options(ig_option("tails", c("one", "two")))
```

ig_param *Declare a method parameter*

Description

Parameters are the numeric (or flag/choice) inputs of a method definition. Each has a role, a domain, a unit and an optional default. A default is written into every specification that does not supply the parameter, so the serialised specification always shows the value that was used.

Usage

```
ig_param(
  name,
  role,
  domain,
  unit = NA_character_,
  default = NULL,
  label = NULL,
```

```
  description = NULL,
  aliases = character(),
  modes = NULL,
  applies_when = NULL
)
```

Arguments

name	Parameter name (lower case, digits and underscores).
role	One of "effect", "nuisance", "design", "error" or "control" (computational settings such as an exact-computation threshold or solver bounds).
domain	An ig_domain object.
unit	Unit label (for example "subjects"); NA when unitless.
default	Default value (validated against domain), or NULL when the parameter must be supplied.
label	Display label, a list with elements en and optionally de.
description	Optional longer description (English).
aliases	Alternative input names accepted by analysis_spec().
modes	Modes in which the parameter is an input; NULL means every mode in which it is not the solve target.
applies_when	Optional applicability condition (see Details).

Details

Applicability conditions use a small data grammar shared with the registry: list(mode = c(...)), list(option = "<id>", "in" = c(...)), list(all = list(...)) and list(any = list(...)). A condition that refers to an option which is itself not applicable is false.

Value

An object of class ig_param.

See Also

Other method definition API: ig_domain, ig_method(), ig_option(), ig_options(), ig_params(), ig_standard_params

Examples

```
ig_param("d", role = "effect", domain = ig_domain_real(),
         label = list(en = "Effect size d"))
```

ig_params *Collect method parameters*

Description

Collect method parameters

Usage

```
ig_params(...)
```

Arguments

...	ig_param() objects (or lists of them).

Value

An object of class ig_params: a list named by parameter name.

See Also

Other method definition API: ig_domain, ig_method(), ig_option(), ig_options(), ig_param(), ig_standard_params

Examples

```
ig_params(ig_param_alpha(), ig_param_power(), ig_param_n_total())
```

ig_standard_params *Standard error-rate and sample-size parameters*

Description

Convenience constructors for the inputs shared by all power analyses: significance level alpha, power power (1 - beta), the compromise ratio error_ratio (q = beta/alpha) and the total sample size n_total. None has a default: they must be supplied whenever the mode needs them.

Usage

```
ig_param_alpha()

ig_param_power()

ig_param_error_ratio()

ig_param_n_total(lower = 2)
```

Arguments

lower	Smallest admissible total sample size.

Value

An ig_param() object.

See Also

Other method definition API: ig_domain, ig_method(), ig_option(), ig_options(), ig_param(), ig_params()

Examples

```
ig_param_alpha()
ig_param_n_total(lower = 3)
```

---

ig_status *Create a typed status*

Description

Create a typed status

Usage

```
ig_status(code, reason = "", details = list())
```

Arguments

code	One of ig_status_codes().
reason	A human-readable explanation. Required (non-empty) for every code except "ok".
details	A named list of structured details (for example the list of problems found during validation). Values must be serialisable data.

Value

An object of class ig_status.

See Also

Other status helpers: ig_status_codes(), is_ig_status()

Examples

```
ig_status("unsupported", "engine for t_generic not implemented yet")
```

`ig_status_codes` *Status codes used by inferguidR*

Description

The six typed outcome codes shared by specifications, results and solvers:

Usage

```
ig_status_codes()
```

Details

- `ok`: the requested quantity was computed and passed its checks.
- `infeasible`: the request is valid but has no solution (for example a target power that cannot be reached within the admissible designs).
- `unsupported`: the request is valid but the capability is not implemented (for example a method without an engine yet).
- `invalid_input`: the request violates a domain, mode or consistency rule.
- `numerical_failure`: a numerical routine failed or its checks did not pass; no number is reported.
- `reference_pending`: the computation depends on reference evidence that does not exist yet.

Value

A character vector of the six codes.

See Also

Other status helpers: `ig_status()`, `is_ig_status()`

Examples

```
ig_status_codes()
```

ig_to_json *Canonical JSON for inferguidR objects*

Description

Serialises a specification, result, envelope, report snapshot or recommendation (or any plain list of data) to canonical JSON: sorted keys, shortest round-trip numbers and explicit markers for non-finite values. The same object always produces the same text, so the text can be hashed.

Usage

```
ig_to_json(x, pretty = FALSE)
```

Arguments

x	An inferguidR object or a plain list/vector of data.
pretty	If TRUE, indent the output (two spaces). Pretty output contains the same tokens; hashes are always computed on the compact form.

Value

A character string of class json.

See Also

ig_from_json(), ig_hash()

Other serialisation: ig_from_json(), ig_hash()

Examples

```
ig_to_json(list(b = 1, a = c(0.1 + 0.2, Inf)))
```

is_ig_status *Test for a status object*

Description

Test for a status object

Usage

```
is_ig_status(x)
```

Arguments

x	Any object.

Value

TRUE if x is an ig_status.

See Also

Other status helpers: ig_status(), ig_status_codes()

---

launch_app *Launch the inferguidR application*

Description

Starts the bundled 'shiny' application. The application directory is located inside the installed package (system.file("app", package = "inferguidR")), never through a development path.

Usage

```
launch_app(..., launch.browser = interactive())
```

Arguments

...	Further arguments passed to shiny::runApp().
launch.browser	Passed to shiny::runApp().

Details

While the application is not part of the installed build, the function stops with an error of class inferguidR_unsupported (its $status is an ig_status() with code "unsupported").

Value

Called for its side effect; returns the value of shiny::runApp() when the application exists.

Examples

```
app_bundled <- nzchar(system.file("app", "app.R", package = "inferguidR"))
if (!app_bundled) {
  # Without a bundled application the call fails with a typed condition.
  tryCatch(launch_app(), inferguidR_unsupported = function(e) e$status)
}
if (interactive() && app_bundled) launch_app()
```

learning_lesson *Find a bundled worked lesson*

Description

Returns paths to an original bilingual lesson and its executable synthetic example. Reading this metadata does not execute the example, access the network or write files. The script writes to an explicit output directory when called. External links in the lesson are optional background sources.

Usage

```
learning_lesson(topic = "boschloo", locale = "en")
```

Arguments

topic	Lesson identifier; currently "boschloo".
locale	Language, "en" or "de".

Details

The Boschloo lesson covers only the two-sided minlike variant for independent binomial groups. It distinguishes numerical optimisation bounds from sampling uncertainty and uses the same variant for prospective planning. A successful teaching example is not full numerical release certification.

Value

A list with topic, locale, title, exact analysis variant, local Markdown/HTML/script paths and SHA-256 content hashes. Paths belong to the installed package and should be treated as read-only.

Examples

```
lesson <- learning_lesson("boschloo", "en")
basename(lesson$script)
```

list_methods *List the registered statistical methods*

Description

Returns one row per registered method with its implementation and validation state: the 49 procedures of the G*Power reference inventory, followed by inferguidR extensions that are not G*Power procedures (family "boschloo": Boschloo's unconditional exact test). implementation_status is "not_implemented" (no method definition), "definition_only" (forward model registered, no mode solver yet), "partial" or "implemented". An implemented method has a registered engine and mode solvers; individual parameter branches can still return a typed refusal. validation_status is the method's validation status; inventory methods without an engine report "reference_pending", and extensions without an engine report "not_applicable" (no G*Power reference can exist for them). No method is G*Power-validated until authentic application captures exist.

Usage

```
list_methods(family = NULL)
```

Arguments

family	Optional family filter ("exact", "F", "t", "z", "chi2", or "boschloo" for the extension).

Value

A data frame with columns id, family, label, modes (comma-separated), implementation_status and validation_status.

See Also

validation_status()

Examples

```
head(list_methods())
list_methods(family = "chi2")
```

---

metric_dictionary *Metric dictionary for binary classification*

Description

Returns the documented definition of every classification metric that inferguidR computes: stable id, label, the formula actually computed, its denominator, the valid domain, whether the metric depends on prevalence, the preferred direction and the typed statuses it can take when it is undefined or infinite. The dictionary is bundled as inst/metadata/metric-dictionary.json; formulas are documentation and are never evaluated.

Usage

```
metric_dictionary(lang = c("en", "de"), scope = NULL)
```

Arguments

lang	Label language, "en" or "de".
scope	Optional filter: "operating_point", "curve", "probability" or "decision".

Details

A metric that is "invariant given fixed class-conditional distributions" does not change when only the prevalence changes while the score distributions within the true classes stay fixed. That is an assumption of prevalence scenarios, not a claim about real population shift.

Value

A data frame with one row per metric and the columns `id`, `short`, `name`, `scope`, `group`, `family`, `formula`, `denominator`, `unit`, `lower`, `upper`, `prevalence_dependence`, `prevalence_dependent`, `preferred_direction`, `complement`, `inputs`, `undefined_statuses`, `interval_methods`, `source_feature_ids` and `correction`. List-like fields are collapsed with `"; "`.

See Also

- `classification_spec()`
- `evaluate_cutoff()`
- `plot_classification()`

Examples

```
d <- metric_dictionary()
d[d$id %in% c("ppv", "lr_pos"), c("id", "formula", "undefined_statuses")]
```

`net_benefit` *Decision-curve net benefit*

Description

Net benefit of the rule "alert if predicted probability >= pt" at explicit decision threshold probabilities `pt`, with the harm weight $w = p_t/(1-p_t)$ (the odds at the threshold: one false alert is weighed as w missed true alerts):

$$NB = TP/N - (FP/N)w,$$

together with the references "alert all" ($\pi - (1-\pi)w$) and "alert none" (0). Scores must be predicted probabilities of the positive class; a generic score cutoff is not a threshold probability. Net benefit presumes calibrated probabilities, see `calibration_summary()`.

Usage

```
net_benefit(spec, threshold_probability)
```

Arguments

`spec`	A scored-data `classification_spec` with attached data and probability scores.
`threshold_probability`	Numeric threshold probabilities in (0, 1). No default: thresholds express a declared harm-benefit trade-off.

Value

A data frame with `model_id`, `threshold_probability`, `harm_weight`, `tp`, `fp`, `n_total`, `net_benefit`, `net_benefit_alert_all`, `net_benefit_alert_none`, `rule` and `estimate_kind`.

See Also

- classification_spec()
- evaluate_cutoff()
- plot_classification()

Examples

```
ex <- classification_examples("scored_synthetic")
net_benefit(ex$spec, threshold_probability = c(0.1, 0.2, 0.3))
```

---

plot_classification *Draw a classification panel*

Description

Static views of classification_plot_data() for classification results. ROC paths retain the ordering of distinct decision sets, including tied scores. Precision-recall points are not interpolated: a straight segment need not correspond to any attainable cutoff. The selected operating point is a ring. The source plot data are retained in the `inferguidR_plot_data` attribute.

Usage

```
plot_classification(
  x,
  panel = c("roc", "pr", "mosaic", "score_distribution", "prevalence_curve",
    "model_comparison", "metric_vs_cutoff", "metric_pair", "metric_network",
    "calibration", "net_benefit"),
  model_id = NULL,
  basis = c("observed", "scenario"),
  lang = c("en", "de"),
  mosaic_mode = c("joint", "within_class"),
  bins = 40,
  metric_family = "within_truth",
  metric_x = "fpr",
  metric_y = "tpr",
  log_x = FALSE,
  log_y = FALSE,
  metric_focus = "ppv",
  interval_method = NULL,
  ppv_levels = c(0.25, 0.5, 0.75),
  f1_levels = c(0.2, 0.4, 0.6, 0.8),
  realised_rules = NULL
)
```

Arguments

x	A classification_result from evaluate_cutoff().
panel	ROC, precision-recall, mosaic, score distribution, prevalence curve, or model comparison (see the supported values below).
model_id	Model identifier; defaults to the first evaluated model.
basis	"observed" or "scenario"; see classification_plot_data().
lang	Label language, "en" or "de".
mosaic_mode	Joint population shares or within-class conditional rates.
bins	Number of display bins for scored-data distributions; no metric or cutoff is computed from these bins.
metric_family	For metric-versus-cutoff views: within_truth, within_alerts, summary, burden, or likelihood_ratios. Likelihood ratios use a logarithmic axis; other families keep their declared domains.
metric_x, metric_y	Operating-point metric identifiers for a metric-pair view.
log_x, log_y	Logarithmic axes for the metric-pair view. Non-positive, undefined and infinite coordinates are counted and omitted, never clamped.
metric_focus	Operating-point metric whose direct dictionary dependencies are shown in a metric-network view. This is a calculation graph, not a causal graph.
interval_method	Optional marginal interval bars at the current ROC or PR point: wilson, clopper_pearson or bootstrap_percentile. The method must have been computed in the result. These are separate fixed-cutoff intervals, not a joint confidence region or a curve band. Scenario intervals are not drawn.
ppv_levels, f1_levels	Reference levels strictly between zero and one for ROC iso-PPV and PR iso-F1 curves.
realised_rules	Optional independently supplied counts for varying-cutoff rules; see classification_plot_data(). Drawn as separate triangles, never as fixed-threshold curve points.

Value

A ggplot object. Requires the suggested package 'ggplot2'. Fixed counts cannot provide ROC, PR or score-distribution curves. Score distributions and model comparisons always use the observed/model basis. Prevalence curves hold the current class-conditional rates fixed.

See Also

- classification_spec()
- evaluate_cutoff()

Examples

```
if (requireNamespace("ggplot2", quietly = TRUE)) {
  r <- evaluate_cutoff(classification_examples("scored_synthetic")$spec)
  plot_classification(r, "roc", "model_b")
}
```

plot_design_sweep *Draw an evaluated design sweep without crossing missing results*

Description

Draw an evaluated design sweep without crossing missing results

Usage

```
plot_design_sweep(x, lang = c("en", "de"))
```

Arguments

x	Data returned by xy_plot_data().
lang	Label language, "en" or "de".

Value

A static ggplot object retaining all point data and result identities.

plot_power *Draw a power analysis panel*

Description

Uses power_plot_data() for both the plotted values and availability checks. The plot is a static counterpart for application and report use. Distribution curves are display samples; decision probabilities come from the engine. The returned plot retains its source data in the inferguidR_plot_data attribute.

Usage

```
plot_power(
  result,
  panel = c("distribution", "decision_table", "mosaic", "lattice"),
  lang = c("en", "de")
)
```

Arguments

result	An unmodified power_result.
panel	"distribution", "decision_table", "mosaic", or "lattice".
lang	Label language, "en" or "de".

Value

A ggplot object. Requires the suggested package 'ggplot2'. An unavailable panel raises a typed unsupported condition.

Examples

```
if (requireNamespace("ggplot2", quietly = TRUE)) {
  s <- analysis_spec("t_means_one_sample", "post_hoc",
    inputs = list(d = .5, alpha = .05, n_total = 30),
    options = list(tails = "two"))
  plot_power(calculate_power(s))
}
```

---

power_plot_data *Plot-ready power distributions and conditional decision probabilities*

Description

All decision probabilities are taken from the computed result. Display ordinates are densities or probability masses, never estimates of power. Unsupported distribution descriptors return an explicit unavailable panel. The decision table conditions on each hypothesis separately: its columns must not be interpreted as joint probabilities or predictive values. The Boschloo lattice uses the solved group sizes and recorded numerical settings. Its H0 panel uses a size-maximising common event probability; its H1 panel uses the specified alternative. Crosses mark rejection tables, not observed data. Larger lattices return an explicit unavailable status.

Usage

```
power_plot_data(
  result,
  panel = c("distribution", "decision_table", "mosaic", "lattice"),
  points = 401L
)
```

Arguments

result	An unmodified power_result.
panel	One of "distribution", "decision_table", "mosaic", or "lattice" (Boschloo table probabilities).
points	Sampling resolution (51 to 2001). Each continuous law uses this many quantile points and equally spaced points, plus critical values. The lattice enumerates all tables (limit 5000), independently of this resolution.

Value

A list with status, source hashes, plain data frames, and notes.

Examples

```
s <- analysis_spec("t_means_one_sample", "post_hoc",
  inputs = list(d = .5, alpha = .05, n_total = 30),
  options = list(tails = "two"))
power_plot_data(calculate_power(s), "decision_table")
```

---

ppv_gap_max_prevalence

Prevalence that maximises PPV minus prevalence

Description

For a fixed operating point with true-positive rate a and false-positive rate b, PPV as a function of prevalence is $PPV(\pi) = a\pi/(b + (a - b)\pi)$. It is concave, so it rises fastest at $\pi \to 0$ (slope LR+), not at an interior prevalence. The expression $(\sqrt{ab} - b)/(a - b) = \sqrt{b}/(\sqrt{a} + \sqrt{b})$, shown by some tools as the prevalence "where PPV rises fastest", is the prevalence that maximises the gap $PPV(\pi) - \pi$: the point where the PPV curve has slope 1 and where $PPV = 1 - \pi$. This function returns that value under its correct name, using the numerically stable form.

Usage

```
ppv_gap_max_prevalence(tpr, fpr)
```

Arguments

`tpr`, `fpr`	True- and false-positive rates in [0, 1] (vectors of equal length, or length 1). `NA` gives an undefined result.

Details

Domain: a value only for $a > b > 0$. $b = 0 < a$: the supremum 1 of the gap is approached as $\pi \to 0$ but not attained (status `undefined`, reason `fpr_zero_supremum`). $a = b$: the gap is 0 for every prevalence (`equal_tpr_fpr`). $a < b$: the expression would be a minimiser (`not_applicable`, `tpr_below_fpr`).

Value

A data frame with `tpr`, `fpr`, `value`, `status`, `reason_code` and `reason`.

See Also

- `classification_spec()`
- `evaluate_cutoff()`
- `plot_classification()`

Examples

```
ppv_gap_max_prevalence(tpr = c(0.8, 0.5, 0.3, 0.6), fpr = c(0.1, 0.5, 0.4, 0))
```

pr_summary *Precision-recall summaries of scored models*

Description

Returns average precision (step sum over the decision sets) and the trapezoidal PR area (precision at recall 0 set to 1) as two distinct quantities. Both refer to the observed prevalence; with `prevalence`, the average precision is also transported to that prevalence under fixed class-conditional score distributions.

Usage

```
pr_summary(spec, prevalence = NULL)
```

Arguments

`spec`	A scored-data `classification_spec` with attached data.
`prevalence`	Optional scenario prevalence in (0, 1).

Value

A data frame with `model_id`, `metric_id`, `value`, `status`, `reason_code`, `reason`, `estimate_kind` and `setting`.

See Also

- `classification_spec()`
- `evaluate_cutoff()`
- `plot_classification()`

Examples

```
d <- data.frame(y = c(1, 0, 1, 1, 0, 0, 1, 0), s = c(.9, .8, .7, .6, .55, .5, .4, .3))
spec <- classification_spec(d, truth = "y", positive_label = 1, scores = "s", cutoff = .5)
pr_summary(spec, prevalence = 0.1)
```

predictive_values_at_prevalence *Predictive values at a given prevalence (Bayes transform)*

Description

Transports a fixed operating point to other prevalences: with TPR and FPR held fixed (fixed class-conditional score distributions),

$$PPV(\pi) = \frac{TPR\,\pi}{TPR\,\pi + FPR\,(1-\pi)}, \quad NPV(\pi) = \frac{(1-FPR)(1-\pi)}{(1-FPR)(1-\pi) + (1-TPR)\pi}.$$

The result is a scenario, not an observed measurement; real population shifts can also change TPR and FPR.

Usage

```
predictive_values_at_prevalence(tpr, fpr, prevalence)
```

Arguments

tpr, fpr	Rates of one operating point (length 1).
prevalence	Vector of prevalences in [0, 1].

Value

A data frame with prevalence, ppv, ppv_status, ppv_reason_code, npv, npv_status and npv_reason_code.

See Also

- classification_spec()
- evaluate_cutoff()
- plot_classification()

Examples

```
# Worked example: TPR 20 %, FPR 1 %
predictive_values_at_prevalence(0.2, 0.01, c(0.1, 0.2, 0.31))
```

`prevalence_for_ppv_target` *Prevalence needed to reach a PPV target*

Description

At fixed TPR and FPR, PPV increases with prevalence (for TPR > 0). The smallest prevalence at which PPV reaches a target t is

$$\pi_t = \frac{t\,FPR}{(1-t)\,TPR + t\,FPR}.$$

PPV is at least t for every prevalence in $[\pi_t, 1]$. The target is never reachable when TPR = 0; when FPR = 0 and TPR > 0, PPV is 1 at every prevalence above 0, so no positive minimum exists.

Usage

```
prevalence_for_ppv_target(tpr, fpr, target)
```

Arguments

`tpr`, `fpr`	Rates of one operating point (vectors, recycled).
`target`	PPV target in (0, 1). No default: targets are user decisions.

Value

A data frame with `tpr`, `fpr`, `target`, `prevalence_needed`, `status`, `reason_code` and `reason`.

See Also

- `classification_spec()`
- `evaluate_cutoff()`
- `plot_classification()`

Examples

```
prevalence_for_ppv_target(tpr = c(0.2, 0.04, 0), fpr = c(0.01, 0.10, 0.1), target = 0.9)
```

prevalence_scenario *Prevalence scenarios for a fixed operating point*

Description

Transports operating points to assumed prevalences under fixed class-conditional score distributions (TPR and FPR kept) and recomputes all operating-point metrics from the expected counts. The values are scenarios, not new observed measurements: they assume that only the prevalence changes, which real population shifts need not satisfy. Expected counts are fractional, are not observed units and have no sampling interval. Without events or without non-events at the operating point, the scenario is undefined rather than computed with an invented rate.

Usage

```
prevalence_scenario(
  x,
  prevalence,
  n_total = NULL,
  source = c("assumed", "external"),
  lang = c("en", "de")
)
```

Arguments

x	Named counts tp, fn, fp, tn of one operating point, or a classification_result (its observed operating point per model).
prevalence	Numeric vector of scenario prevalences in [0, 1].
n_total	Total of the scenario population (default: the total of the counts); it scales the expected counts, not the rates.
source	Provenance of the prevalence: "assumed" (default) or "external" (for example a registry estimate).
lang	Language of reason texts and denominator labels.

Value

A data frame with model_id, scenario_prevalence, scenario_n_total, scenario_source, scenario_status and the metric columns of classification_metrics() (estimate kind "expected_scenario").

See Also

- classification_spec()
- evaluate_cutoff()
- plot_classification()

Examples

```
# Case-control style sample (50 % events) transported to 2 % prevalence
prevalence_scenario(c(tp = 80, fn = 20, fp = 10, tn = 90), prevalence = c(0.02, 0.5))
```

`protocol_record` *Protocol record of an analysis*

Description

Builds the record that the 'G*Power' protocol window shows for a calculation: test family and procedure, analysis type, the inputs and the outputs with their 'G*Power' labels and display formatting. The labels are the live labels of the method and analysis type in the method registry. Use `protocol_text()` to format records.

Usage

```
protocol_record(result)
```

Arguments

`result`	A `power_result` (see `calculate_power()`).

Value

An object of class `ig_protocol_record`: a list with `method_id`, `mode`, `status`, `family_line`, `options_line` (`NULL` without options), `analysis_line`, `inputs` and `outputs` (data frames with `label`, `text` and `value`), `unresolved` (labels inferguidR could not fill), `notes` (where the record may differ from G*Power's layout), `validation` (validation status of the method), `spec_hash`, `result_hash` and `r_call`.

See Also

`protocol_text()`

Examples

```
res <- calculate_power(analysis_spec("t_means_two_independent", "post_hoc",
  inputs = list(d = 0.5, alpha = 0.05, n1 = 50, n2 = 50), options = list(tails = "two")))
rec <- protocol_record(res)
rec$outputs
```

protocol_text *Protocol of power analyses as text*

Description

Formats one or more analyses as the protocol window of 'G*Power' 3.1.9.7 does: a numbered, time-stamped record per analysis with the test, the analysis type, and the inputs and outputs by their 'G*Power' labels and display formatting. With `provenance = TRUE` (default) each record ends with the inferguidR version, the validation status of the method, the specification and result hashes and the reproducible R call.

Usage

```
protocol_text(
  ...,
  results = list(),
  time = Sys.time(),
  start = 1L,
  provenance = TRUE,
  eol = "\n"
)
```

Arguments

`...`	`power_result` objects (or `ig_protocol_record` objects).
`results`	A list of further results.
`time`	Time stamp(s) of the records: one `POSIXct` value used for all records or one per record. Defaults to the current time.
`start`	Number of the first record.
`provenance`	Append the inferguidR provenance lines to each record.
`eol`	Line end: `"\n"` (default) or `"\r\n"` as in the 'G*Power' export.

Value

A single string.

See Also

`protocol_record()`

Examples

```
res <- calculate_power(analysis_spec("t_means_two_independent", "post_hoc",
  inputs = list(d = 0.5, alpha = 0.05, n1 = 50, n2 = 50), options = list(tails = "two")))
cat(protocol_text(res, time = as.POSIXct("2026-10-05 13:57:49", tz = "UTC")))
```

report_pdf_capabilities *Check the offline PDF report renderer*

Description

Check the offline PDF report renderer

Usage

```
report_pdf_capabilities()
```

Value

A list with available and reason. No software is installed.

select_cutoff *Select a cutoff by an objective or constraints*

Description

Chooses the operating point of each model by a declared rule and evaluates it with `evaluate_cutoff()`. For scored data the search is exhaustive over every distinct decision set of the original scores (including alert none and alert all); for a theoretical binormal model the optimum over real-valued cutoffs is found numerically (and analytically for Youden's J). Ties are reported in full and broken by the declared `tie_break` among actual operating points.

Usage

```
select_cutoff(
  spec,
  objective = NULL,
  constraints = NULL,
  optimise = NULL,
  capacity = NULL,
  costs = NULL,
  tie_break = NULL,
  prevalence_basis = NULL,
  ...
)
```

Arguments

spec	A classification_spec (scored data or theory).
objective	One of "max_youden", "max_f1", "max_mcc", "constraint", "capacity", "cost_utility"; NULL uses the selection rule already in spec.
constraints, optimise, capacity, costs, tie_break, prevalence_basis	See cutoff_rule().
...	Evaluation settings passed to evaluate_cutoff() (partial_auc, threshold_probability, calibration_bins, include_curves).

Details

No policy threshold is built in: every constraint is a user decision. Conflicting constraints, an objective that is undefined everywhere or a supremum that no cutoff attains give a result with status "infeasible" and a per-constraint feasibility report instead of a number.

A cutoff chosen on the evaluated data is descriptive: the result is labelled in-sample and its intervals are conditional on the selected cutoff. To evaluate a rule chosen on tuning data, pass the tuning result as cutoff to evaluate_cutoff() on the evaluation specification.

Value

A classification_result. The specification it was evaluated with (the input with the selection rule) is returned by classification_evaluated_spec(); pair the two in an analysis_envelope().

See Also

evaluate_cutoff(), cutoff_rule()

- classification_spec()
- evaluate_cutoff()
- plot_classification()

Examples

```
ex <- classification_examples("scored_synthetic")
res <- select_cutoff(ex$spec, "max_youden")
classification_table(res, "selection")

# Highest TPR among cutoffs with FPR at most 10 %
res2 <- select_cutoff(ex$spec, "constraint", constraints = list(fpr_max = 0.1))
res2$models[[1]]$operating_point$cutoff

# Theory: analytic Youden optimum of an unequal-variance binormal model
th <- classification_spec(theory = list(auc = 0.8, sigma = 1.5), cutoff = 0, prevalence = 0.1)
select_cutoff(th, "max_youden")$models[[1]]$operating_point$cutoff
```

solve_design *Solve for a study design*

Description

Solves an a priori specification for the smallest admissible sample size (and group sizes) that reaches the target power. Specifications in other modes are answered with status `"invalid_input"`; use `calculate_power()` for them.

Usage

```
solve_design(spec)
```

Arguments

spec	A `power_spec` created by `analysis_spec()`.

Value

A `power_result`.

See Also

`calculate_power()`

Examples

```
spec <- analysis_spec("t_means_one_sample", "a_priori",
  inputs = list(alpha = 0.05, power = 0.8, d = 0.5),
  options = list(tails = "two"))
solve_design(spec)$status
```

sweep_design *Sweep one input of a specification*

Description

Varies one input of a specification over the given values and solves each resulting specification with `calculate_power()`. Every row is a complete, separately validated specification with its own result. Values are kept in the order given; rows that are not computable (an inadmissible value, an infeasible target, an unsupported branch) keep their status and reason and have no numbers, so gaps and jumps are preserved rather than interpolated.

Usage

```
sweep_design(spec, free, values)
```

Arguments

spec	A power_spec created by analysis_spec().
free	Name of the input to vary. It must be an input of spec (not its solve target).
values	Values of the free input.

Value

A data frame with one row per value and columns value, admissible (the value gives a valid specification), status, reason, one solved_<target> column per solve target, achieved_power, achieved_beta, alpha_attained, n_total, spec_hash, result_hash and the list columns spec and result. Attributes free, fixed_inputs, fixed_options, method_id, mode, target and base_spec_hash describe the sweep.

See Also

compare_designs(), calculate_power()

Examples

```
spec <- analysis_spec("t_means_two_independent", "a_priori",
  inputs = list(alpha = 0.05, power = 0.8, d = 0.5), options = list(tails = "two"))
sw <- sweep_design(spec, "d", c(0.4, 0.5, 0.6))
sw[, c("value", "status", "solved_n_total", "achieved_power")]
```

---

update.classification_spec

Update a classification specification

Description

Specifications are immutable; update() builds a new, fully validated specification. For a theoretical model, changing auc or sigma keeps the numerical cutoff unchanged.

Usage

```
## S3 method for class 'classification_spec'
update(
  object,
  ...,
  rule = NULL,
  cutoff = NULL,
  prevalence = NULL,
  prevalence_source = NULL,
  sampling = NULL,
  theory = NULL,
  evaluation_split = NULL,
  uncertainty = NULL
)
```

Arguments

object	A classification_spec.
...	Not used.
rule, cutoff	A new operating rule (see cutoff_rule()) or a new manual cutoff (which keeps the declared selection data).
prevalence, prevalence_source, sampling	New prevalence settings; use prevalence = NA to return to the observed prevalence.
theory	For theoretical models: a list with any of auc, sigma, n_total.
evaluation_split	New evaluation split for all models.
uncertainty	New settings from classification_uncertainty().

Value

A new classification_spec.

See Also

- classification_spec()
- evaluate_cutoff()
- plot_classification()

Examples

```
th <- classification_spec(theory = list(auc = 0.8, sigma = 1), cutoff = 1.2, prevalence = 0.1)
th2 <- update(th, theory = list(auc = 0.9))
th2$operating_rule$score_cutoff
```

---

update.power_spec *Derive a new specification from an existing one*

Description

Specifications are immutable; update() creates a new, fully validated specification. Inputs and options given here replace the old ones; a value of NULL removes an input or option. Applied defaults are re-derived. When the mode changes, inputs that become solve targets of the new mode are dropped.

Usage

```
## S3 method for class 'power_spec'
update(
  object,
  ...,
  inputs = list(),
  options = list(),
  mode = NULL,
  target = NULL
)
```

Arguments

object	A power_spec.
...	Further named inputs to set (or remove with NULL).
inputs	Named list of inputs to set (or remove with NULL).
options	Named list of options to set (or remove with NULL).
mode	New analysis mode (optional).
target	New solve target (optional).

Value

A new power_spec.

Examples

```
spec <- analysis_spec("t_means_one_sample", "a_priori",
  inputs = list(alpha = 0.05, power = 0.8, d = 0.5), options = list(tails = "two"))
update(spec, inputs = list(alpha = 0.01))
update(spec, mode = "post_hoc", inputs = list(n_total = 40))
```

---

validation_status *Validation state of a method*

Description

Reports what is known about the correctness evidence of one method: whether an engine exists, its validation status, the state of the G*Power reference comparison (*"validated"*,* "reference_pending"*, or* "not_applicable"* for inferguidR extensions that are not G*Power procedures), the independent checker and the documented manual examples that can serve as (non-authoritative) evidence. Pending reference status means that complete comparison is not certified, even when captured cases and independent numerical checks already exist.

Usage

```
validation_status(method_id)
```

Arguments

method_id	A method id from list_methods().

Value

An object of class ig_validation_status (a list).

Examples

```
validation_status("t_means_two_independent")
```

---

workbench_app *Create the interactive study workbench*

Description

Creates the bundled Shiny application without starting a server. The current research preview links a registry-driven planning workspace, synthetic classifier examples and the guided method advisor. Input controls follow the method definitions, including option applicability and solve targets. Pending numerical domains are not presented as released or externally validated calculations.

Usage

```
workbench_app()
```

Value

A Shiny application object. Requires 'shiny' and 'ggplot2'.

See Also

launch_app()

---

xy_plot_data *Data of an X-Y plot for a range of values*

Description

Computes the data of the 'G*Power' "X-Y plot for a range of values": one of power, total sample size, effect size and alpha (y) as a function of another (x), optionally as a family of curves over a third quantity, with the fourth held constant at its value in the base analysis (or at fixed). Every point is a complete specification solved with calculate_power(): y = "power" uses post hoc mode, "sample_size" a priori, "effect_size" sensitivity and "alpha" criterion mode. The options and all other inputs come from the base analysis.

Usage

```
xy_plot_data(
  base,
  y,
  x,
  from,
  to,
  by,
  curves = NULL,
  fixed = list(),
  max_points = 2000L
)
```

Arguments

base	A power_result with status "ok" or a power_spec (it is calculated first).
y, x	Quantities on the axes: "power", "sample_size", "effect_size" or "alpha"; they must differ.
from, to, by	Range of x ("from", "through to", "in steps of").
curves	Optional family of curves: a list with parameter (one of the two remaining quantities) and either values (1 to 10 values) or from, by and n (1 to 10 curves).
fixed	Optional named list with a value for the quantity that is held constant (default: its value in the base analysis).
max_points	Largest number of points per curve.

Details

Sample sizes are integer designs. A sample-size curve is therefore a staircase (render = "step"); the fractional totals of the continuous relaxation (what 'G*Power' tabulates with "interpolating points") are in column n_total_continuous and are not achievable designs. With the total sample size on the x axis only whole numbers are allowed, and totals without an admissible split into groups (for example an odd total under equal allocation) are kept as points with status invalid_input.

Value

An object of class ig_xy_plot_data: a list with points (a data frame with one row per point: curve, curve_value, x, y, status, reason, n_total, n_total_continuous, achieved_power, spec_hash, result_hash and the list columns group_sizes and result), axes, curves, fixed (every input and option held constant, with labels), method_id, mode (mode used for y), title, subtitle, render ("step" for sample-size curves, "line" otherwise), validation and notes. Invalid ranges signal an inferguidR_invalid_input condition.

See Also

sweep_design(), compare_designs()

Examples

```
base <- calculate_power(analysis_spec("t_means_two_independent", "post_hoc",
  inputs = list(d = 0.5, alpha = 0.05, n1 = 50, n2 = 50), options = list(tails = "two")))
xy <- xy_plot_data(base, y = "power", x = "sample_size", from = 20, to = 200, by = 20)
xy$points[, c("x", "y", "status")]
xy2 <- xy_plot_data(base, y = "sample_size", x = "power", from = 0.6, to = 0.95, by = 0.05,
  curves = list(parameter = "effect_size", values = c(0.4, 0.5, 0.6)))
head(xy2$points[, c("curve_value", "x", "y", "n_total_continuous")])
```

Index

∗ **Boschloo**
boschloo_compare_fisher, 24
boschloo_control, 26
boschloo_plotdata_allocation, 27
boschloo_plotdata_lattice, 28
boschloo_plotdata_nuisance, 30
boschloo_plotdata_power_effect, 31
boschloo_plotdata_power_n, 32
boschloo_power, 33
boschloo_probabilities, 34
boschloo_region, 35
boschloo_sample_size, 36
boschloo_test, 38
boschloo_variants, 39
∗ **advisor**
advisor_answer, 4
advisor_override, 5
advisor_power_spec, 6
advisor_question, 7
advisor_r_code, 8
advisor_recommendation, 12
advisor_replay, 13
advisor_rules, 14
advisor_save, 16
advisor_session, 17
advisor_set_language, 18
advisor_validate_rules, 18
∗ **classification**
attach_classification_data, 22
binormal_model, 23
binormal_rates, 24
calibration_summary, 42
classification_auc, 43
classification_evaluated_spec, 44
classification_examples, 45
classification_metrics, 46
classification_plot_data, 47
classification_spec, 49
classification_table, 51
classification_uncertainty, 52
classifier_model, 54
compare_classifiers, 55
cutoff_rule, 58
cutoff_sweep, 60
evaluate_cutoff, 64
format_metric_values, 67
metric_dictionary, 83
net_benefit, 84
plot_classification, 85
ppv_gap_max_prevalence, 89
pr_summary, 90
predictive_values_at_prevalence, 91
prevalence_for_ppv_target, 92
prevalence_scenario, 93
select_cutoff, 96
update.classification_spec, 99
∗ **method definition API**
ig_domain, 68
ig_method, 71
ig_option, 74
ig_options, 75
ig_param, 75
ig_params, 77
ig_standard_params, 77
∗ **serialisation**
ig_from_json, 70
ig_hash, 71
ig_to_json, 80
∗ **status helpers**
ig_status, 78
ig_status_codes, 79
is_ig_status, 80

advisor_answer, 4
advisor_answer(), *6–8*, *12–18*, *20*
advisor_back *(*advisor_answer*)*, 4
advisor_back(), *17*

advisor_clear_override (advisor_override), 5
advisor_override, 5
advisor_override(), 5, 7, 8, 12–18, 20
advisor_power_spec, 6
advisor_power_spec(), 5, 6, 8, 12–18, 20
advisor_question, 7
advisor_question(), 4–7, 12–18, 20
advisor_r_code, 8
advisor_r_code(), 5–8, 13–18, 20
advisor_recommendation, 12
advisor_recommendation(), 5–8, 12, 14–18, 20
advisor_replay, 13
advisor_replay(), 5–8, 12, 13, 15–18, 20
advisor_restore (advisor_save), 16
advisor_rules, 14
advisor_rules(), 5–8, 12–14, 16–20
advisor_save, 16
advisor_save(), 5–8, 12–15, 17, 18, 20
advisor_session, 17
advisor_session(), 5–8, 12–16, 18, 20
advisor_set_language, 18
advisor_set_language(), 5–8, 12–17, 20
advisor_validate_rules, 18
advisor_validate_rules(), 5–8, 12–18
analysis_envelope, 20
analysis_envelope(), 44, 97
analysis_spec, 21
analysis_spec(), 40, 41, 63, 76, 98, 99
attach_classification_data, 22

binormal_model, 23
binormal_rates, 24
boschloo_compare_fisher, 24
boschloo_compare_fisher(), 27–31, 33–35, 37, 39, 40
boschloo_control, 26
boschloo_control(), 7, 25, 28–40
boschloo_plotdata_allocation, 27
boschloo_plotdata_allocation(), 25, 27, 29–31, 33–35, 37, 39, 40
boschloo_plotdata_lattice, 28
boschloo_plotdata_lattice(), 25, 27, 28, 30, 31, 33–35, 37, 39, 40
boschloo_plotdata_nuisance, 30
boschloo_plotdata_nuisance(), 25, 27–29, 31, 33–35, 37, 39, 40
boschloo_plotdata_power_effect, 31
boschloo_plotdata_power_effect(), 25, 27–30, 33–35, 37, 39, 40
boschloo_plotdata_power_n, 32
boschloo_plotdata_power_n(), 25, 27–31, 34, 35, 37, 39, 40
boschloo_power, 33
boschloo_power(), 25, 27–31, 33, 35, 37, 39, 40
boschloo_probabilities, 34
boschloo_probabilities(), 25, 27–37, 39, 40
boschloo_region, 35
boschloo_region(), 25, 27–31, 33–37, 39, 40
boschloo_sample_size, 36
boschloo_sample_size(), 25, 27–31, 33–35, 39, 40
boschloo_test, 38
boschloo_test(), 25, 27–37, 40
boschloo_variants, 39
boschloo_variants(), 25, 27–31, 33–35, 37, 39

calculate_power, 40
calculate_power(), 7, 22, 56, 94, 98, 99, 102
calculator_functions, 41
calculator_functions(), 61, 62
calibration_summary, 42
calibration_summary(), 84
classification_auc, 43
classification_evaluated_spec, 44
classification_evaluated_spec(), 65, 97
classification_examples, 45
classification_metrics, 46
classification_metrics(), 67, 93
classification_plot_data, 47
classification_plot_data(), 85, 86
classification_spec, 49
classification_spec(), 23, 24, 42–46, 48, 52, 54, 55, 59, 60, 65, 67, 84–86, 89–93, 97, 100
classification_table, 51
classification_table(), 65, 67
classification_uncertainty, 52
classification_uncertainty(), 50, 100
classifier_model, 54
classifier_model(), 50
compare_classifiers, 55
compare_designs, 56

compare_designs(), *99*, *103*
create_report_snapshot, 56
cutoff_rule, 58
cutoff_rule(), *50*, *97*, *100*
cutoff_sweep, 60
cutoff_sweep(), *51*, *65*

distribution_calculator, 61
distribution_calculator(), *41*

effect_size, 62
evaluate_cutoff, 64
evaluate_cutoff(), *23, 24*, *42–46*, *48*, *51, 52*, *54, 55*, *59, 60*, *67*, *84–86*, *89–93*, *96, 97*, *100*
export_report, 66

format_metric_values, 67

ig_config, 68
ig_config(), *70*
ig_domain, 68, *73–78*
ig_domain_choice *(*ig_domain*)*, 68
ig_domain_integer *(*ig_domain*)*, 68
ig_domain_logical *(*ig_domain*)*, 68
ig_domain_real *(*ig_domain*)*, 68
ig_domain_real(), *73*
ig_from_json, 70
ig_from_json(), *68*, *71*, *80*
ig_hash, 71
ig_hash(), *70*, *80*
ig_method, 71
ig_method(), *69*, *74–78*
ig_option, 74
ig_option(), *69*, *73*, *75–78*
ig_options, 75
ig_options(), *69*, *72–74*, *76–78*
ig_param, 75
ig_param(), *68, 69*, *73–75*, *77, 78*
ig_param_alpha *(*ig_standard_params*)*, 77
ig_param_error_ratio *(*ig_standard_params*)*, 77
ig_param_n_total *(*ig_standard_params*)*, 77
ig_param_power *(*ig_standard_params*)*, 77
ig_params, 77
ig_params(), *69*, *72–76*, *78*
ig_standard_params, *69*, *73–76*, 77, *77*
ig_status, 78
ig_status(), *61*, *63*, *70*, *79*, *81*
ig_status_codes, 79
ig_status_codes(), *78*, *81*
ig_to_json, 80
ig_to_json(), *13*, *22*, *70, 71*
is_ig_status, 80
is_ig_status(), *78, 79*

launch_app, 81
launch_app(), *102*
learning_lesson, 82
list_methods, 82
list_methods(), *21*, *62*, *72*, *102*

metric_dictionary, 83
metric_dictionary(), *46*, *59, 60*, *65*

net_benefit, 84
net_benefit(), *64*

plot_classification, 85
plot_classification(), *23, 24*, *42–46*, *48*, *51, 52*, *54, 55*, *59, 60*, *65*, *67*, *84, 85*, *89–93*, *97*, *100*
plot_design_sweep, 87
plot_power, 87
power_plot_data, 88
power_plot_data(), *87*
ppv_gap_max_prevalence, 89
pr_summary, 90
predictive_values_at_prevalence, 91
prevalence_for_ppv_target, 92
prevalence_scenario, 93
prevalence_scenario(), *46*, *65*, *67*
protocol_record, 94
protocol_record(), *95*
protocol_text, 95
protocol_text(), *94*

report_pdf_capabilities, 96

select_cutoff, 96
select_cutoff(), *44*, *51*, *64, 65*
solve_design, 98
solve_design(), *7*, *41*
stats::fisher.test(), *24*, *26*, *38*
stats::optimize(), *26*
sweep_design, 98
sweep_design(), *56*, *103*

update.classification_spec, 99
update.power_spec, 100
update.power_spec(), *22*

validation_status, 101
validation_status(), *83*

workbench_app, 102

xy_plot_data, 102
xy_plot_data(), *87*