Package 'evaluatellm'

September 15, 2026

Title Statistical Inference for Language Model Evaluations

Version 0.1.0

Description Treats language model evaluations as statistical experiments and supplies the inference they require. Provides central limit theorem and cluster-robust standard errors for evaluation scores, paired and unpaired model comparisons, variance decomposition when several responses are drawn per question, control-variate variance reduction, multiplicity adjustment across benchmark suites, and power and minimum detectable effect calculations for planning evaluations, following Miller (2024) <doi:10.48550/arXiv.2411.00640>. For evaluations scored by a model judge, implements agreement statistics against a human gold standard and prediction-powered inference (Angelopoulos et al. 2023) <doi:10.1126/science.adi6000> with the power-tuned estimator of Angelopoulos, Bates and Jordan (2023) <doi:10.48550/arXiv.2311.01453>, so a small set of human labels debiases a large set of judge scores. Leaderboards are supported through bootstrap rank intervals and Bradley-Terry ratings (Bradley and Terry 1952) <doi:10.2307/2334029>. Accepts scores from any evaluation harness.

License MIT + file LICENSE

Encoding UTF-8

Language en-GB

URL https://charlescoverdale.github.io/evaluatellm/, https://github.com/charlescoverdale/evaluatellm

BugReports https://github.com/charlescoverdale/evaluatellm/issues

RoxygenNote 7.3.3

Depends R (>= 4.1.0)

Imports cli (>= 3.6.0), graphics, grDevices, stats, utils

Suggests testthat (>= 3.0.0), knitr, rmarkdown, sandwich

VignetteBuilder knitr

Config/testthat/edition 3

NeedsCompilation no

Author Charles Coverdale [aut, cre, cph]

Maintainer Charles Coverdale <charlesfcoverdale@gmail.com>

Repository CRAN

Date/Publication 2026-09-15 13:40:02 UTC

Contents

`as_eval` *Build an Evaluation Object*

Description

Wraps evaluation scores in the long-format structure the `ev_*()` functions expect: one row per scored response, identified by item, model, and optionally cluster and sample.

Usage

```
as_eval(
  data,
  score = NULL,
  item = NULL,
  model = NULL,
  cluster = NULL,
  sample = NULL
)
```

Arguments

`data`	A data frame of evaluation results, one row per scored response. Alternatively a bare numeric or logical vector of scores, in which case items are numbered sequentially and a single model is assumed.
`score`	Column holding the score. Numeric, or logical for pass or fail grading. Given unquoted, or as a string.
`item`	Column identifying the question. Defaults to `NULL`, meaning row order, which is correct only when each row is a distinct question.
`model`	Column identifying the model. Defaults to `NULL`, meaning a single unnamed model.
`cluster`	Column identifying groups of items that share structure, for example several questions asked about one reading passage, or several paraphrases of one prompt. Supply this whenever it exists: ignoring it understates standard errors, often severely.
`sample`	Column identifying repeated draws for the same item and model. Only needed if the same item and model appear on several rows and you want `ev_resample()` to decompose the variance.

Details

Every function in this package treats the *item* as the sampling unit, on the view that an evaluation is a sample of questions drawn from an unseen super-population of questions someone could have written (Miller 2024). Repeated responses to the same item are averaged within the item before inference, so drawing more responses per item never inflates the apparent sample size.

Value

An `evaluatellm_eval` object: a data frame with columns `item`, `model`, `score`, and, when supplied, `cluster` and `sample`.

References

Miller, E. (2024). Adding Error Bars to Evals: A Statistical Approach to Language Model Evaluations. doi:10.48550/arXiv.2411.00640

Examples

```
# A bare vector of pass or fail results
set.seed(1)
as_eval(rbinom(200, 1, 0.7))

# A full evaluation with clustered questions and two models
d <- data.frame(
  q       = rep(1:100, times = 2),
  passage = rep(rep(1:20, each = 5), times = 2),
  m       = rep(c("a", "b"), each = 100),
  correct = rbinom(200, 1, 0.6)
)
```

```
as_eval(d, score = correct, item = q, model = m, cluster = passage)
```

---

ev_bootstrap *Cluster Bootstrap for an Arbitrary Statistic*

Description

Resamples items, or whole clusters of items when a cluster column is present, and recomputes a user-supplied statistic on each replicate. Use this when the quantity of interest is not a mean and the analytic standard errors elsewhere in the package do not apply: medians, quantiles, pass rates above a threshold, or any custom score aggregation.

Usage

```
ev_bootstrap(
  data,
  model = NULL,
  statistic = mean,
  R = 2000,
  level = 0.95,
  type = c("percentile", "basic"),
  seed = NULL,
  ...
)
```

Arguments

`data`	An `evaluatellm_eval` object from `as_eval()`, a data frame, or a bare numeric or logical vector of scores.
`model`	Which model to score. Optional when the data holds only one.
`statistic`	A function taking a numeric vector of item scores and returning a single number. Default `mean()`.
`R`	Number of bootstrap replicates. Default `2000`.
`level`	Confidence level. Default `0.95`.
`type`	Interval type, `"percentile"` (default) or `"basic"`.
`seed`	Optional integer seed for reproducibility.
`...`	Passed to `as_eval()` when `data` is a plain data frame.

Details

Resampling clusters rather than items preserves the dependence structure, so the resulting interval carries the same protection as `ev_cluster()`.

Value

An `evaluatellm_bootstrap` object with elements `estimate`, `se`, `conf_low`, `conf_high`, `replicates`, `R`, `type`, and `level`.

See Also

Other single model: `ev_cluster()`, `ev_icc()`, `ev_resample()`, `ev_score()`

Examples

```
set.seed(5)
e <- as_eval(rbeta(300, 6, 3))
ev_bootstrap(e, statistic = median, R = 500, seed = 1)
```

---

ev_cluster *Cluster-Robust Standard Error for an Evaluation*

Description

Computes a cluster-robust standard error for a model's mean score, together with the design effect and intra-cluster correlation that explain how much precision the clustering costs.

Usage

```
ev_cluster(data, model = NULL, level = 0.95, ...)
```

Arguments

`data`	An `evaluatellm_eval` object carrying a cluster column, or a data frame passed to `as_eval()` along with `....`
`model`	Which model to score. Optional when the data holds only one.
`level`	Confidence level. Default `0.95`.
`...`	Passed to `as_eval()` when `data` is a plain data frame.

Details

Many evaluations draw several questions from one source: comprehension questions about a shared passage, variants of one prompt template, or items generated from a single seed document. Those questions are not independent draws, and treating them as though they were understates the standard error by a factor of `sqrt(design effect)`. With eight questions per passage and an intra-cluster correlation of 0.3, the honest interval is roughly 1.6 times wider than the naive one.

The estimator is the CR1-corrected cluster-robust variance of a mean, `(G / (G - 1)) * sum_g (sum_i u_i)^2 / n^2`, where `u_i` are deviations from the mean, `g` indexes clusters and `G` counts them. Inference uses the t distribution on `G - 1` degrees of freedom, so results are appropriately cautious when clusters are few.

Value

An `evaluatellm_cluster` object with elements `estimate`, `se`, `se_naive`, `conf_low`, `conf_high`, `design_effect`, `icc`, `n_items`, `n_clusters`, `mean_cluster_size`, `df`, and `level`.

See Also

Other single model: `ev_bootstrap()`, `ev_icc()`, `ev_resample()`, `ev_score()`

Examples

```
set.seed(2)
# 40 passages, 10 questions each, with a strong passage effect
passage_skill <- rnorm(40, 0, 1)
d <- data.frame(
  q       = 1:400,
  passage = rep(1:40, each = 10),
  correct = rbinom(400, 1, plogis(0.9 + rep(passage_skill, each = 10)))
)
e <- as_eval(d, score = correct, item = q, cluster = passage)
ev_cluster(e)
```

---

ev_elo *Bradley-Terry Ratings from Pairwise Preferences*

Description

Fits a Bradley-Terry model to head-to-head comparison data, the format used by arena-style evaluations where a judge or a human picks a winner between two responses. Returns each model's strength with a standard error, on both the log-odds and the Elo scale.

Usage

```
ev_elo(
  data,
  model_a,
  model_b,
  winner,
  ties = c("drop", "split"),
  anchor = 1500,
  level = 0.95
)
```

Arguments

`data`	A data frame of pairwise comparisons.
`model_a, model_b`	Columns naming the two models in each comparison.

winner	Column giving the outcome. Either the name of the winning model, or a numeric or logical column that is 1 or TRUE when model_a won. Ties are NA, "tie", or 0.5.
ties	How to treat ties, "drop" (default) or "split".
anchor	Elo rating of the average model. Strengths are centred, so this fixes the level of the whole table. Default 1500.
level	Confidence level. Default 0.95.

Details

Elo numbers are usually published bare, which hides how little separates adjacent models. Fitting the model properly gives standard errors, so a 15-point gap can be recognised as noise. The fit is a logistic regression of the win indicator on signed model indicators with no intercept, estimated by maximum likelihood, with the first model fixed at zero for identification. Strengths are centred to sum to zero and Elo is the linear rescaling `anchor + (400 / log(10)) * strength`, so `anchor` is the rating of an average model. Every model carries a standard error, obtained by the delta method from the fitted covariance, including the one held out for identification.

Ties are dropped by default, since Bradley-Terry has no tie term. Set `ties = "split"` to count each tie as half a win to each side.

Value

An `evaluatellm_elo` object with element `models`, a data frame ordered best first with columns `model`, `strength` (centred), `se`, `elo`, `elo_low`, `elo_high`, and `n_games`.

References

Bradley, R. A. and Terry, M. E. (1952). Rank Analysis of Incomplete Block Designs: I. The Method of Paired Comparisons. *Biometrika*. doi:10.2307/2334029

See Also

Other leaderboard: `ev_rank()`

Examples

```
set.seed(13)
strength <- c(a = 0.9, b = 0.5, c = 0.0, d = -0.6)
pairs <- t(replicate(1200, sample(names(strength), 2)))
p <- plogis(strength[pairs[, 1]] - strength[pairs[, 2]])
d <- data.frame(
  left  = pairs[, 1],
  right = pairs[, 2],
  won   = rbinom(1200, 1, p)
)
ev_elo(d, model_a = left, model_b = right, winner = won)
```

ev_icc *Intra-Cluster Correlation*

Description

Estimates the intra-cluster correlation of evaluation scores using the one-way random effects analysis of variance estimator. This is the quantity `ev_power()` needs in order to size a clustered evaluation, and the quantity that determines how much `ev_cluster()` will widen an interval.

Usage

```
ev_icc(data, model = NULL, ...)
```

Arguments

`data`	An `evaluatellm_eval` object carrying a cluster column, or a data frame passed to `as_eval()` along with `....`
`model`	Which model to score. Optional when the data holds only one.
`...`	Passed to `as_eval()` when `data` is a plain data frame.

Details

The estimator is `(MSB - MSW) / (MSB + (m0 - 1) * MSW)`, with `m0` the usual unbalanced-design correction for average cluster size. Sampling noise can push the raw estimate below zero, in which case it is reported as zero and a note is attached.

Value

A single number between 0 and 1, with attribute `raw` holding the uncensored estimate.

See Also

Other single model: `ev_bootstrap()`, `ev_cluster()`, `ev_resample()`, `ev_score()`

Examples

```
set.seed(3)
passage_skill <- rnorm(30, 0, 1.2)
d <- data.frame(
  q       = 1:300,
  passage = rep(1:30, each = 10),
  correct = rbinom(300, 1, plogis(0.5 + rep(passage_skill, each = 10)))
)
ev_icc(as_eval(d, score = correct, item = q, cluster = passage))
```

`ev_judge_agreement` *Agreement Between a Model Judge and a Human Gold Standard*

Description

Quantifies how far a model judge can be trusted, by comparing its scores against human labels on the items where both exist. Reports agreement, chance-corrected agreement, and whether the judge is systematically generous or harsh.

Usage

```
ev_judge_agreement(judge, gold, level = 0.95)
```

Arguments

`judge`	Numeric or logical vector of judge scores.
`gold`	Numeric or logical vector of human scores, the same length as `judge`. Entries may be `NA` where no human label exists; those items are ignored here.
`level`	Confidence level. Default `0.95`.

Details

Raw agreement flatters a judge whenever one label dominates: a judge that passes everything agrees 90 per cent of the time with a gold standard that passes 90 per cent of the time, while carrying no information at all. Cohen's kappa corrects for that and is the number to read.

Systematic bias matters more than noise, because noise averages out over items while bias does not. For binary scores the McNemar test asks whether the judge's disagreements run in one direction; a significant result means the judge's mean score is wrong, not merely noisy, and every evaluation graded by it inherits that error. The fix is `ev_judge_debias()`, not a better prompt.

The measurement type is detected from the data: binary when scores take two values, categorical when they take a few, continuous otherwise.

Value

An `evaluatellm_agreement` object. Always carries `n`, `type`, `bias`, `bias_conf_low`, `bias_conf_high`, `bias_p_value`, `correlation`, `mean_judge`, and `mean_gold`. Binary and categorical scores add `accuracy`, `accuracy_conf_low`, `accuracy_conf_high`, `kappa`, `kappa_se`, and, for binary, `sensitivity` and `specificity`.

See Also

Other judge: `ev_judge_debias()`, `ev_judge_power()`

Examples

```
set.seed(10)
truth <- rbinom(400, 1, 0.6)
# A judge that is right 85 per cent of the time and slightly too generous
judge <- ifelse(runif(400) < 0.85, truth, 1 - truth)
judge[truth == 0 & runif(400) < 0.1] <- 1
ev_judge_agreement(judge, truth)
```

---

ev_judge_debias *Debias a Model Judge with a Small Human Sample*

Description

Combines a large set of judge-scored items with a small set of human-labelled items to estimate the score a full human evaluation would have produced, with valid confidence intervals. This is prediction-powered inference.

Usage

```
ev_judge_debias(judge, gold, lambda = NULL, level = 0.95)
```

Arguments

judge	Numeric or logical vector of judge scores for every item.
gold	Numeric or logical vector of human scores, the same length as judge, with NA where no human label exists.
lambda	Optional fixed tuning coefficient in [0, 1]. Default NULL, meaning the variance-minimising value is estimated. lambda = 1 gives the original prediction-powered estimator, lambda = 0 the human-only mean.
level	Confidence level. Default 0.95.

Details

Grading with a model judge is cheap but biased, and the bias does not shrink as more items are judged. Grading by hand is unbiased but small. The prediction-powered estimator takes the judge's mean over everything and subtracts the judge's measured error on the human-labelled subset:

estimate = lambda * mean(judge, unlabelled) + mean(gold - lambda * judge, labelled)

This is unbiased for any lambda, so no assumption about judge quality is being smuggled in. The default tunes lambda to minimise variance (Angelopoulos, Bates and Jordan 2023), which guarantees the result is never less precise than ignoring the judge and using the human labels alone. When the judge is uninformative lambda goes to zero and the estimator falls back to the human-only mean; when the judge is good, a few hundred human labels can carry the precision of several thousand.

The reported effective_n is the headline: the number of human labels that would have been needed to reach this precision without the judge.

`gold` is expected to be `NA` for items that were not labelled by a human. The labelled items must be a random sample of the evaluated items. If humans were asked to label the cases the judge found hard, this estimator is not valid.

Value

An `evaluatellm_debias` object with elements `estimate`, `se`, `conf_low`, `conf_high`, `lambda`, `estimate_classical`, `se_classical`, `estimate_judge`, `judge_bias`, `effective_n`, `precision_gain`, `n_labelled`, `n_unlabelled`, `correlation`, and `level`.

References

Angelopoulos, A. N., Bates, S., Fannjiang, C., Jordan, M. I., and Zrnic, T. (2023). Prediction-powered inference. *Science*. doi:10.1126/science.adi6000

Angelopoulos, A. N., Bates, S., and Jordan, M. I. (2023). PPI++: Efficient Prediction-Powered Inference. doi:10.48550/arXiv.2311.01453

See Also

Other judge: `ev_judge_agreement()`, `ev_judge_power()`

Examples

```
set.seed(11)
n_total <- 5000
truth <- rbinom(n_total, 1, 0.62)
# Judge agrees 88 per cent of the time and leans generous
judge <- ifelse(runif(n_total) < 0.88, truth, 1 - truth)
judge[truth == 0 & runif(n_total) < 0.12] <- 1

# Humans labelled a random 250 of them
gold <- rep(NA_real_, n_total)
idx <- sample(n_total, 250)
gold[idx] <- truth[idx]

ev_judge_debias(judge, gold)
```

---

ev_judge_power *How Many Human Labels a Debiased Evaluation Needs*

Description

Sizes the human-labelling budget for an evaluation graded by a model judge. Given how many items the judge will score and how well it tracks human labels, returns the number of human labels needed to hit a target precision.

Usage

```
ev_judge_power(
  n_total,
  correlation = NULL,
  target_se = NULL,
  target_mde = NULL,
  sd_gold = 0.5,
  sd_judge = NULL,
  pilot = NULL,
  power = 0.8,
  alpha = 0.05
)
```

Arguments

n_total	Number of items the judge will score.
correlation	Correlation between judge and human scores, from a pilot. A fitted ev_judge_debias() result may be passed instead via pilot.
target_se	Target standard error for the final estimate. Supply this or target_mde.
target_mde	Target minimum detectable effect, converted to a standard error at the given power and alpha.
sd_gold	Standard deviation of the human scores. Defaults to 0.5, the maximum for a binary score, which is conservative.
sd_judge	Standard deviation of the judge scores. Defaults to sd_gold.
pilot	Optional ev_judge_debias() result to take correlation, sd_gold and sd_judge from.
power, alpha	Used only when target_mde is supplied. Defaults 0.8 and 0.05.

Details

This is the question that actually has a budget attached, since human labels are the expensive input. The function evaluates the variance of the tuned prediction-powered estimator across candidate label counts and returns the smallest one that meets the target, alongside the number of labels that would have been needed without a judge.

A judge correlated at 0.8 with human labels typically cuts the required labelling budget by around two thirds. Below about 0.4 the judge is barely worth the pipeline.

Value

An evaluatellm_judge_power object with elements n_labels, n_labels_without_judge, saving, achieved_se, target_se, correlation, lambda, and n_total.

See Also

Other judge: ev_judge_agreement(), ev_judge_debias()

Examples

```
# 20,000 judge-scored items, judge correlates 0.8 with humans,
# want a standard error of 0.01
ev_judge_power(n_total = 20000, correlation = 0.8, target_se = 0.01)

# A weaker judge needs far more human labels
ev_judge_power(n_total = 20000, correlation = 0.4, target_se = 0.01)
```

---

ev_mde *Smallest Difference an Evaluation Can Detect*

Description

The mirror of `ev_power()`. Given the number of questions available, returns the smallest true difference the evaluation has a decent chance of detecting.

Usage

```
ev_mde(
  n_items,
  sd_diff = NULL,
  pilot = NULL,
  p_a = NULL,
  p_b = NULL,
  correlation = NULL,
  power = 0.8,
  alpha = 0.05,
  icc = 0,
  cluster_size = 1
)
```

Arguments

`n_items`	Number of questions available.
`sd_diff`	Standard deviation of the per-item difference.
`pilot`	A result from `ev_paired()` to take `sd_diff` from.
`p_a, p_b`	Expected scores of the two models, for binary grading.
`correlation`	Expected correlation between the two models' item scores. Used with `p_a` and `p_b`. Default `0`, which is conservative.
`power`	Target power. Default `0.8`.
`alpha`	Two-sided significance level. Default `0.05`.
`icc`	Intra-cluster correlation, from `ev_icc()`. Default `0`.
`cluster_size`	Questions per cluster. Default `1`, meaning no clustering.

Details

Run this before an evaluation, not after. If the minimum detectable effect comes back larger than the gain you expect, the evaluation cannot answer the question and a null result will mean nothing. Reporting the minimum detectable effect alongside a null finding is what separates "the models are equivalent" from "this evaluation was too small to tell".

Value

An `evaluatellm_mde` object with elements `mde`, `n_items`, `n_effective`, `sd_diff`, `power`, `alpha`, `design_effect`, and `icc`.

See Also

Other planning: `ev_power()`

Examples

```
# 500 questions, binary scoring, models correlated at 0.7
ev_mde(n_items = 500, p_a = 0.72, p_b = 0.70, correlation = 0.7)

# Same questions, but clustered 8 to a passage: the MDE nearly doubles
ev_mde(n_items = 500, p_a = 0.72, p_b = 0.70, correlation = 0.7,
       icc = 0.25, cluster_size = 8)
```

---

ev_multi *Multiplicity Adjustment Across a Benchmark Suite*

Description

Takes the per-task comparisons that make up a benchmark suite and reports them together: multiplicity-adjusted p-values, simultaneous confidence intervals, a pooled effect, and a heterogeneity check on whether the tasks are telling the same story.

Usage

```
ev_multi(results, method = "holm", level = 0.95)
```

Arguments

`results`	A list of objects from `ev_paired()` or `ev_unpaired()`, optionally named by task. Alternatively a data frame with columns `estimate` and `se`, and optionally `task` and `df`.
`method`	Multiplicity adjustment passed to `stats::p.adjust()`. One of `"holm"` (default), `"BH"`, `"bonferroni"`, `"hochberg"`, `"BY"`, `"hommel"`, or `"none"`.
`level`	Confidence level for the per-task intervals. Default `0.95`. Simultaneous intervals use `1 - (1 - level) / k`.

Details

Running a model against twelve tasks and reporting the two that reached significance is a recipe for false positives: at the five per cent level, better than one task in two will produce at least one spurious win by chance across twelve independent nulls. Adjustment fixes this. Holm controls the family-wise error rate and is the default, appropriate when a single false win would be embarrassing. Benjamini-Hochberg controls the false discovery rate and is more permissive, appropriate for screening many tasks.

The pooled estimate is inverse-variance weighted, which is the minimum-variance combination when the tasks estimate a common effect. Cochran's Q and `i_squared` test that assumption: a large `i_squared` means the per-task effects genuinely differ and the pooled number is hiding something, so read the per-task rows instead.

Value

An `evaluatellm_multi` object with elements `tasks` (a data frame), `pooled`, `pooled_se`, `pooled_conf_low`, `pooled_conf_high`, `pooled_p_value`, `q_statistic`, `q_p_value`, `i_squared`, `n_tasks`, `n_significant`, `method`, and `level`.

See Also

Other comparison: `ev_paired()`, `ev_unpaired()`, `ev_variance_reduction()`

Examples

```
set.seed(9)
# Six tasks, a small true gain on each
runs <- lapply(1:6, function(i) {
  difficulty <- rnorm(250)
  d <- data.frame(
    q = rep(1:250, 2),
    m = rep(c("new", "old"), each = 250),
    correct = rbinom(500, 1,
      plogis(c(0.8, 0.65)[rep(1:2, each = 250)] - rep(difficulty, 2)))
  )
  ev_paired(as_eval(d, score = correct, item = q, model = m),
            model_a = "new", model_b = "old")
})
names(runs) <- paste0("task_", 1:6)
ev_multi(runs)
```

`ev_paired` *Paired Comparison of Two Models*

Description

Tests whether one model outscores another on the same set of questions. This is the correct comparison whenever both models were run on the same evaluation, and it is materially more powerful than comparing two independent means.

Usage

```
ev_paired(
  data,
  model_a = NULL,
  model_b = NULL,
  level = 0.95,
  cluster = TRUE,
  ...
)
```

Arguments

data	An evaluatellm_eval object holding both models, or a data frame passed to as_eval() along with ....
model_a	Name of the first model. The reported difference is model_a - model_b.
model_b	Name of the second model.
level	Confidence level. Default 0.95.
cluster	Logical. Use cluster-robust standard errors when a cluster column is present. Default TRUE.
...	Passed to as_eval() when data is a plain data frame.

Details

Working with the per-item difference d_i = score_A - score_B removes the item difficulty that both models face, so the variance of the difference is var_A + var_B - 2 * cov(A, B) rather than var_A + var_B. Because model scores on shared questions are usually strongly correlated, the paired standard error is routinely half the unpaired one or better, which is the difference between detecting a real gain and reporting a null result. The function reports the realised gain as variance_reduction.

When the data carry a cluster column the standard error of the mean difference is cluster-robust, and the test uses G - 1 degrees of freedom.

Value

An evaluatellm_paired object with elements estimate, se, se_unpaired, conf_low, conf_high, statistic, p_value, correlation, variance_reduction, mean_a, mean_b, n_items, df, and level.

References

Miller, E. (2024). Adding Error Bars to Evals: A Statistical Approach to Language Model Evaluations. doi:10.48550/arXiv.2411.00640

See Also

Other comparison: ev_multi(), ev_unpaired(), ev_variance_reduction()

Examples

```
set.seed(6)
# Shared item difficulty makes the two models' scores correlate
difficulty <- rnorm(500)
d <- data.frame(
  q = rep(1:500, 2),
  m = rep(c("new", "old"), each = 500),
  correct = rbinom(1000, 1,
    plogis(c(0.9, 0.7)[rep(1:2, each = 500)] - rep(difficulty, 2)))
)
ev_paired(as_eval(d, score = correct, item = q, model = m),
          model_a = "new", model_b = "old")
```

`ev_plot` *Plot Results with Error Bars*

Description

Draws the results as a horizontal interval plot, the format this package exists to make routine. Accepts the same input as `ev_table()`, and also plots `ev_rank()` and `ev_multi()` objects directly.

Usage

```
ev_plot(x, ..., reference = NULL, xlab = NULL, main = NULL, col = "#1b365d")
```

Arguments

`x`	An `evaluatellm` result, a list of them, or a data frame from `ev_table()`.
`...`	Further results, or graphical parameters passed to `graphics::plot()`.
`reference`	Optional vertical reference line, for example `0` for differences. Default `NULL`, which draws none.
`xlab`, `main`	Axis and title labels.
`col`	Colour for points and intervals.

Value

The plotted data frame, invisibly.

See Also

Other reporting: `ev_table()`

Examples

```
set.seed(15)
runs <- lapply(1:5, function(i) {
  d <- data.frame(
    q = rep(1:200, 2),
    m = rep(c("new", "old"), each = 200),
    correct = rbinom(400, 1, rep(c(0.72, 0.68), each = 200))
  )
  ev_paired(as_eval(d, score = correct, item = q, model = m),
            model_a = "new", model_b = "old")
})
names(runs) <- paste0("task ", 1:5)
ev_plot(runs, reference = 0)
```

`ev_power` *Number of Questions Needed to Detect a Difference*

Description

Plans a paired model comparison: given the size of the gain worth detecting and the variability of the per-item difference, returns how many questions the evaluation needs.

Usage

```
ev_power(
  delta,
  sd_diff = NULL,
  pilot = NULL,
  p_a = NULL,
  p_b = NULL,
  correlation = NULL,
  power = 0.8,
  alpha = 0.05,
  icc = 0,
  cluster_size = 1
)
```

Arguments

`delta`	The difference worth detecting, on the score scale.
`sd_diff`	Standard deviation of the per-item difference.
`pilot`	A result from `ev_paired()` to take `sd_diff` from.
`p_a`, `p_b`	Expected scores of the two models, for binary grading.
`correlation`	Expected correlation between the two models' item scores. Used with `p_a` and `p_b`. Default `0`, which is conservative.

power	Target power. Default 0.8.
alpha	Two-sided significance level. Default 0.05.
icc	Intra-cluster correlation, from ev_icc(). Default 0.
cluster_size	Questions per cluster. Default 1, meaning no clustering.

Details

The calculation is the standard one for a paired mean, n = (z_alpha/2 + z_beta)^2 * sd_diff^2 / delta^2, multiplied by the design effect 1 + (cluster_size - 1) * icc when questions come in clusters. Normal quantiles are used, which is conventional for planning and slightly optimistic for very small n.

Getting sd_diff right is the whole exercise. It is the standard deviation of the per-question difference, not of the scores, and it is much smaller than people expect because the two models face the same questions. The reliable way to get it is a pilot run passed through ev_paired() and handed back here as pilot. Failing that, supply p_a, p_b and a correlation for binary scoring: a correlation of 0.7 is typical for models of similar capability.

Value

An evaluatellm_power object with elements n_items, n_clusters, delta, sd_diff, power, alpha, design_effect, and icc.

See Also

Other planning: ev_mde()

Examples

```
# Detect a 2 point gain, binary scoring, models correlated at 0.7
ev_power(delta = 0.02, p_a = 0.72, p_b = 0.70, correlation = 0.7)

# The same target, but questions come 8 to a passage
ev_power(delta = 0.02, p_a = 0.72, p_b = 0.70, correlation = 0.7,
         icc = 0.25, cluster_size = 8)
```

---

ev_rank *Bootstrap Rank Intervals for a Leaderboard*

Description

Resamples questions to ask how stable a leaderboard actually is. Returns each model's score, its rank interval, and the probability it is genuinely the best model in the table.

Usage

```
ev_rank(data, R = 2000, level = 0.95, higher_better = TRUE, seed = NULL, ...)
```

Arguments

data	An evaluatellm_eval object holding several models, or a data frame passed to as_eval() along with ....
R	Number of bootstrap replicates. Default 2000.
level	Confidence level. Default 0.95.
higher_better	Logical. Whether a larger score means a better model. Default TRUE.
seed	Optional integer seed for reproducibility.
...	Passed to as_eval() when data is a plain data frame.

Details

Leaderboards are read as though the ordering were a fact, when it is an estimate like any other. Two models separated by half a point on a 400-question evaluation will often swap places on a different 400 questions. The rank interval says which orderings the data actually support, and p_best says how much confidence the top position deserves.

Clusters are resampled whole where a cluster column exists, matching the dependence structure that ev_cluster() handles analytically.

Value

An evaluatellm_rank object with element models, a data frame ordered best first with columns model, estimate, se, conf_low, conf_high, rank, rank_low, rank_high, and p_best.

See Also

Other leaderboard: ev_elo()

Examples

```
set.seed(12)
difficulty <- rnorm(300)
skill <- c(a = 1.0, b = 0.95, c = 0.6, d = 0.2)
d <- do.call(rbind, lapply(names(skill), function(m) {
  data.frame(q = 1:300, m = m,
             correct = rbinom(300, 1, plogis(skill[[m]] - difficulty)))
}))
ev_rank(as_eval(d, score = correct, item = q, model = m), R = 500, seed = 1)
```

`ev_resample` *Variance Decomposition for Repeated Sampling*

Description

When an evaluation draws several responses per question, the observed spread of question means mixes two sources: genuine variation between questions, and sampling noise in the model's own responses. This function separates them, and reports how far more sampling can take you.

Usage

```
ev_resample(
  data,
  model = NULL,
  level = 0.95,
  k_grid = c(1, 2, 4, 8, 16, Inf),
  ...
)
```

Arguments

`data`	An `evaluatellm_eval` object with several rows per item and model, or a data frame passed to `as_eval()` along with `...`.
`model`	Which model to score. Optional when the data holds only one.
`level`	Confidence level. Default `0.95`.
`k_grid`	Integer vector of `k` values at which to report the projected standard error. Default `c(1, 2, 4, 8, 16, Inf)`.
`...`	Passed to `as_eval()` when `data` is a plain data frame.

Details

Writing `s_ij` for the score of response `j` to item `i`, the estimator is the equal-weighted mean of item means. Its variance is `var(item means) / n`, which is unbiased whatever `k` is. The decomposition uses `E[var(item means)] = var_between + E[var_within / k]`, so `var_between = var(item means) - mean(var_within / k)`

The practical consequence is that more responses per question buys precision only against the within-question term, and there is a floor at `sqrt(var_between / n)` that no amount of resampling can beat. Reaching below it requires more questions. Sampling noise can push the estimated between component below zero, in which case it is reported as zero.

Value

An `evaluatellm_resample` object with elements `estimate`, `se`, `se_floor`, `var_between`, `var_within`, `share_within`, `n_items`, `k_mean`, `projection`, `df`, and `level`.

References

Miller, E. (2024). Adding Error Bars to Evals: A Statistical Approach to Language Model Evaluations. doi:10.48550/arXiv.2411.00640

See Also

Other single model: `ev_bootstrap()`, `ev_cluster()`, `ev_icc()`, `ev_score()`

Examples

```
set.seed(4)
# 150 questions, 8 sampled responses each, item difficulty varies
diff <- plogis(rnorm(150, 0.8, 1.1))
d <- data.frame(
  q       = rep(1:150, each = 8),
  draw    = rep(1:8, times = 150),
  correct = rbinom(1200, 1, rep(diff, each = 8))
)
ev_resample(as_eval(d, score = correct, item = q, sample = draw))
```

`ev_score` *Evaluation Score with a Standard Error*

Description

Reports the mean score of a model together with the standard error and confidence interval implied by treating the questions as a sample. This is the number that belongs next to every headline evaluation result, and the one most often omitted.

Usage

```
ev_score(data, model = NULL, level = 0.95, cluster = TRUE, ...)
```

Arguments

`data`	An `evaluatellm_eval` object from `as_eval()`, a data frame, or a bare numeric or logical vector of scores.
`model`	Which model to score. Optional when the data holds only one.
`level`	Confidence level. Default `0.95`.
`cluster`	Logical. Use cluster-robust standard errors when a cluster column is present. Default `TRUE`. Set to `FALSE` to force the independent calculation, which is useful only for comparison.
`...`	Passed to `as_eval()` when `data` is a plain data frame.

Details

The standard error follows the central limit theorem applied to the item means: `sd(score) / sqrt(n)`, where `n` counts items, not responses. When `cluster` is present in the data the calculation switches to a cluster-robust standard error, since questions sharing a passage or a source document are not independent draws. See `ev_cluster()` for the detail.

Confidence intervals use the t distribution with `n - 1` degrees of freedom, or `G - 1` when clustered, which is slightly wider than the normal interval used in Miller (2024) and better behaved on the short evaluations that appear in practice.

Value

An `evaluatellm_score` object with elements `estimate`, `se`, `conf_low`, `conf_high`, `n_items`, `n_responses`, `df`, `level`, `clustered`, and, when clustered, `n_clusters` and `design_effect`.

References

Miller, E. (2024). Adding Error Bars to Evals: A Statistical Approach to Language Model Evaluations. doi:10.48550/arXiv.2411.00640

See Also

Other single model: `ev_bootstrap()`, `ev_cluster()`, `ev_icc()`, `ev_resample()`

Examples

```
set.seed(1)
# 400 questions, 72 per cent correct
ev_score(rbinom(400, 1, 0.72))

# The same accuracy, but questions come in groups of 8 per passage
d <- data.frame(
  q       = 1:400,
  passage = rep(1:50, each = 8),
  correct = rbinom(400, 1, 0.72)
)
ev_score(as_eval(d, score = correct, item = q, cluster = passage))
```

---

ev_table *Collect Results into a Table*

Description

Flattens one or more `evaluatellm` results into a plain data frame with a common set of columns, ready for a paper table or a plot.

Usage

```
ev_table(...)
```

Arguments

`...`	One or more `evaluatellm` result objects, or a single list of them. Names, where given, become the `label` column.

Value

A data frame with columns `label`, `term`, `estimate`, `se`, `conf_low`, `conf_high`, `p_value`, and `n`.

See Also

Other reporting: `ev_plot()`

Examples

```
set.seed(14)
d <- data.frame(
  q = rep(1:300, 2),
  m = rep(c("new", "old"), each = 300),
  correct = rbinom(600, 1, rep(c(0.74, 0.68), each = 300))
)
e <- as_eval(d, score = correct, item = q, model = m)
ev_table(
  new = ev_score(e, model = "new"),
  old = ev_score(e, model = "old"),
  gain = ev_paired(e, model_a = "new", model_b = "old")
)
```

`ev_unpaired` *Unpaired Comparison of Two Models*

Description

Compares two models evaluated on different question sets, using a Welch two-sample interval that does not assume equal variances.

Usage

```
ev_unpaired(data, model_a = NULL, model_b = NULL, level = 0.95, ...)
```

Arguments

`data`	An `evaluatellm_eval` object holding both models, or a data frame passed to `as_eval()` along with `....`
`model_a`	Name of the first model. The reported difference is `model_a - model_b`.
`model_b`	Name of the second model.
`level`	Confidence level. Default `0.95`.
`...`	Passed to `as_eval()` when `data` is a plain data frame.

Details

Prefer `ev_paired()` whenever both models were run on the same questions. This function exists for the case where they were not, for instance when comparing a published score against your own run, and it will be substantially less powerful because item difficulty is left in the error term.

Value

An `evaluatellm_unpaired` object with elements `estimate`, `se`, `conf_low`, `conf_high`, `statistic`, `p_value`, `mean_a`, `mean_b`, `n_a`, `n_b`, `df`, and `level`.

See Also

Other comparison: `ev_multi()`, `ev_paired()`, `ev_variance_reduction()`

Examples

```
set.seed(7)
d <- data.frame(
  q = 1:600,
  m = rep(c("a", "b"), each = 300),
  correct = rbinom(600, 1, rep(c(0.72, 0.65), each = 300))
)
ev_unpaired(as_eval(d, score = correct, item = q, model = m),
            model_a = "a", model_b = "b")
```

---

ev_variance_reduction *Variance Reduction with a Reference Model*

Description

Uses a reference model whose score on the full question bank is already known to shrink the standard error of a new model's score, without changing what is being estimated.

Usage

```
ev_variance_reduction(
  data,
  model = NULL,
  reference = NULL,
  mu_reference = NULL,
  theta = NULL,
  level = 0.95,
  ...
)
```

Arguments

data	An evaluatellm_eval object holding both models, or a data frame passed to as_eval() along with ....
model	The model being scored.
reference	The reference model used as the control variate.
mu_reference	The reference model's known mean score on the full question bank.
theta	Optional fixed coefficient. Default NULL, meaning the variance-minimising value is estimated from the data.
level	Confidence level. Default 0.95.
...	Passed to as_eval() when data is a plain data frame.

Details

The idea is the control variate. If a reference model scores z_i on the same questions and its population mean mu_reference is known, then the questions this evaluation happened to draw can be recognised as easier or harder than average, and the new model's score corrected accordingly:

estimate = mean(y) - theta * (mean(z) - mu_reference)

The variance-minimising coefficient is theta = cov(y, z) / var(z), and the resulting variance is var(y) * (1 - rho^2) / n. The adjustment is unbiased for any theta, so nothing is assumed beyond mu_reference being correct. Because model scores on shared questions correlate strongly, a correlation of 0.8 removes roughly two thirds of the variance, which is equivalent to tripling the number of questions.

mu_reference has to come from outside this evaluation: a reference model run over the whole bank, or a published score on the full benchmark. Supplying the reference model's mean on these same questions would make the correction identically zero.

Value

An evaluatellm_vr object with elements estimate, se, conf_low, conf_high, estimate_raw, se_raw, theta, correlation, variance_reduction, effective_n, n_items, df, and level.

See Also

Other comparison: ev_multi(), ev_paired(), ev_unpaired()

Examples

```
set.seed(8)
difficulty <- rnorm(200)
d <- data.frame(
  q = rep(1:200, 2),
  m = rep(c("new", "ref"), each = 200),
  correct = rbinom(400, 1,
    plogis(c(1.0, 0.6)[rep(1:2, each = 200)] - rep(difficulty, 2)))
)
e <- as_eval(d, score = correct, item = q, model = m)
# The reference model is known to score 0.63 on the full bank
```

```
ev_variance_reduction(e, model = "new", reference = "ref",
                      mu_reference = 0.63)
```

Index