Package ‘unsum’

July 22, 2025

Title Reconstruct Raw Data from Summary Statistics

Version 0.2.0

Description Reconstructs all possible raw data that could have led to reported summary statistics. Provides a wrapper for the 'Rust' implementation of the 'CLOSURE' algorithm.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.2

Depends R (>= 4.2.0)

Imports cli, ggplot2, nanoparquet, readr, rlang, roundwork, scales, tibble, utils

Collate 'utils.R' 'count.R' 'extendr-wrappers.R' 'performance.R' 'horns.R' 'generate.R' 'horns-analyze.R' 'plot.R' 'read-write.R'

Config/rextendr/version 0.4.0.9000

SystemRequirements Cargo (Rust's package manager), rustc

URL `https://github.com/lhdjung/unsum`, `https://lhdjung.github.io/unsum/`

BugReports `https://github.com/lhdjung/unsum/issues`

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

VignetteBuilder knitr

Config/testthat/edition 3

NeedsCompilation yes

Author Lukas Jung [aut, cre], Nathanael Larigaldie [ctb]

Maintainer Lukas Jung <jung-lukas@gmx.net>

Repository CRAN

Date/Publication 2025-06-19 19:20:02 UTC

Contents

`closure_count_initial` *Count CLOSURE samples in advance*

Description

Determine how many samples `closure_generate()` would find for a given set of summary statistics.

- `closure_count_initial()` only counts the first round of samples, from which all other ones would be generated.
- There is currently no `closure_count_all()` function.

This can help predict how much time `closure_generate()` would take, and avoid prohibitively long runs.

Usage

```
closure_count_initial(scale_min, scale_max)
```

Arguments

`scale_min, scale_max`
Integers (length 1 each). Minimum and maximum of the scales to which the reported statistics refer.

Value

Integer (length 1).

Examples

```
closure_count_initial(scale_min = 1, scale_max = 5)
```

closure_gauge_complexity *Heuristic to predict CLOSURE runtime*

Description

Before you run `closure_generate()`, you may want to get a sense of the time it will take to run. Use `closure_gauge_complexity()` to compute a heuristics-based complexity score. For reference, here is how it determines the messages in `closure_generate()`:

Value	**Message**
if < 1	(no message)
else if < 2	"Just a second..."
else if < 3	"This could take a minute..."
else	"NOTE: Long runtime ahead!"

Usage

```
closure_gauge_complexity(mean, sd, n, scale_min, scale_max)
```

Arguments

`mean` String (length 1). Reported mean.

`sd` String (length 1). Reported sample standard deviation.

`n` Numeric (length 1). Reported sample size.

`scale_min`, `scale_max`
Numeric (length 1 each). Minimal and maximal possible values of the measurement scale. For example, with a 1-7 Likert scale, use `scale_min = 1` and `scale_max = 7`. Prefer the empirical min and max if available: they constrain the possible values further.

Details

The result of this function is hard to interpret. All it can do is to convey an idea about the likely runtime of CLOSURE. This is because the input parameters interact in highly dynamic ways, which makes prediction difficult.

In addition, even progress bars or updates at regular intervals (e.g., "10% complete") prove to be extremely challenging: the Rust code computes CLOSURE results in parallel, which makes it hard to get an overview of the total progress across all cores; and especially to display such information on the R level.

Value

Numeric (length 1).

Examples

```
# Low SD, N, and scale range:
closure_gauge_complexity(
  mean = 2.55,
  sd = 0.85,
  n = 84,
  scale_min = 1,
  scale_max = 5
)

# Somewhat higher:
closure_gauge_complexity(
  mean = 4.26,
  sd = 1.58,
  n = 100,
  scale_min = 1,
  scale_max = 7
)

# Very high:
closure_gauge_complexity(
  mean = 3.81,
  sd = 3.09,
  n = 156,
  scale_min = 1,
  scale_max = 7
)
```

---

closure_generate *Generate CLOSURE samples*

Description

Call `closure_generate()` to run the CLOSURE algorithm on a given set of summary statistics.

This can take seconds, minutes, or longer, depending on the input. Wide variance and large `n` often lead to many samples, i.e., long runtimes. These effects interact dynamically. For example, with large `n`, even very small increases in `sd` can greatly increase runtime and number of values found.

If the inputs are inconsistent, there is no solution. The function will then return empty results and throw a warning.

Usage

```
closure_generate(
  mean,
  sd,
  n,
  scale_min,
  scale_max,
```

```
  rounding = "up_or_down",
  threshold = 5,
  warn_if_empty = TRUE,
  ask_to_proceed = TRUE,
  rounding_error_mean = NULL,
  rounding_error_sd = NULL
)
```

Arguments

mean String (length 1). Reported mean.

sd String (length 1). Reported sample standard deviation.

n Numeric (length 1). Reported sample size.

scale_min, scale_max
Numeric (length 1 each). Minimal and maximal possible values of the measurement scale. For example, with a 1-7 Likert scale, use scale_min = 1 and scale_max = 7. Prefer the empirical min and max if available: they constrain the possible values further.

rounding String (length 1). Rounding method assumed to have created mean and sd. See *Rounding options*, but also the *Rounding limitations* section below. Default is "up_or_down" which, e.g., unrounds 0.12 to 0.115 as a lower bound and 0.125 as an upper bound.

threshold Numeric (length 1). Number from which to round up or down, if rounding is any of "up_or_down", "up", and "down". Default is 5.

warn_if_empty Logical (length 1). Should a warning be shown if no samples are found? Default is TRUE.

ask_to_proceed Logical (length 1). If the runtime is predicted to be very long, should the function prompt you to proceed or abort in an interactive setting? Default is TRUE.

rounding_error_mean, rounding_error_sd
Numeric (length 1 each). Option to manually set the rounding error around mean and sd. This is meant for development and might be removed in the future, so most users can ignore it.

Value

Named list of four tibbles (data frames):

- inputs: Arguments to this function.
- metrics:
  - samples_initial: integer. The basis for computing CLOSURE results, based on scale range only. See closure_count_initial().
  - samples_all: integer. Number of all samples. Equal to the number of rows in results.
  - values_all: integer. Number of all individual values found. Equal to n * samples_all.
  - horns: double. Measure of dispersion for bounded scales; see horns().
  - horns_uniform: double. Value horns would have if the reconstructed sample was uniformly distributed.

- frequency:
  - value: integer. Scale values derived from scale_min and scale_max.
  - f_average: Count of scale values in the mean results sample.
  - f_absolute: integer. Count of individual scale values found in the results samples.
  - f_relative: double. Values' share of total values found.
- results:
  - id: integer. Runs from 1 to samples_all.
  - sample: list of integer vectors. Each of these vectors has length n. It is a sample (or distribution) of individual scale values found by CLOSURE.

Rounding limitations

The rounding and threshold arguments are not fully implemented. For example, CLOSURE currently treats all rounding bounds as inclusive, even if the rounding specification would imply otherwise.

Many specifications of the two arguments will not make any difference, and those that do will most likely lead to empty results.

Examples

```
# High spread often leads to many samples --
# here, 3682.
data_high <- closure_generate(
  mean = "3.5",
  sd = "1.7",
  n = 70,
  scale_min = 1,
  scale_max = 5
)

data_high

# Get a clear picture of the distribution
# by following up with `closure_plot_bar()`:
closure_plot_bar(data_high)

# Low spread, only 3 samples, and not all
# scale values are possible.
data_low <- closure_generate(
  mean = "2.9",
  sd = "0.5",
  n = 70,
  scale_min = 1,
  scale_max = 5
)

data_low

# This can also be shown by `closure_plot_bar()`:
closure_plot_bar(data_low)
```

`closure_horns_analyze` *Horns index for each CLOSURE sample*

Description

Following up on `closure_generate()`, you can call `closure_horns_analyze()` to compute the horns index for each individual sample and compute summary statistics on the distribution of these indices. See `horns()` for the metric itself.

This adds more detail to the `"horns"` and `"horns_uniform"` columns in the output of `closure_generate()`, where `"horns"` is the overall mean of the per-sample indices found here.

`closure_horns_histogram()` draws a quick barplot to reveal the distribution of horns values. The scale is fixed between 0 and 1.

Usage

```
closure_horns_analyze(data)

closure_horns_histogram(
  data,
  bar_alpha = 0.8,
  bar_color = "#5D3FD3",
  bar_binwidth = 0.0025,
  text_size = 12
)
```

Arguments

`data`	For `closure_horns_analyze()`, a list returned by `closure_generate()`. For `closure_horns_histogram()`, a list returned by `closure_horns_analyze()`.
`bar_alpha`	Numeric (length 1). Opacity of the bars. Default is `0.8`.
`bar_color`	String (length 1). Color of the bars. Default is `"#5D3FD3"`, a purple color.
`bar_binwidth`	Width of the bins that divide up the x-axis, passed on to `ggplot2::geom_histogram()`. Default is `0.0025`.
`text_size`	Numeric. Base font size in pt. Default is `12`.

Details

The `"mad"` column overrides a default of `stats::mad()`: adjusting the result via multiplication by a constant (about 1.48). This assumes a normal distribution, which generally does not seem to be the case with horns index values. Here, the constant is set to 1.

Value

closure_horns_analyze() returns a named list of two tibbles (data frames):

- **horns_metrics**: Summary statistics of the distribution of horns index values:
  - mean, uniform: same as horns and horns_uniform from closure_generate()'s output.
  - sd: double. Standard deviation.
  - cv: double. Coefficient of variation, i.e., sd / mean.
  - mad: double. Median absolute deviation; see stats::mad().
  - min, median, max: double. Minimum, median, and maximum horns index.
  - range: double. Equal to max - min.
- **horns_results**:
  - id: integer. Uniquely identifies each horns index, just like their corresponding samples in closure_generate().
  - horns: double. Horns index for each individual sample.

closure_horns_histogram() returns a ggplot object.

Examples

```
data <- closure_generate(
  mean = "2.9",
  sd = "0.5",
  n = 70,
  scale_min = 1,
  scale_max = 5
)

data_horns <- closure_horns_analyze(data)
data_horns

closure_horns_histogram(data_horns)
```

---

closure_plot_bar *Visualize CLOSURE data in a histogram*

Description

Call closure_plot_bar() to get a barplot of CLOSURE results.

For each scale value, the bars show how often this value appears in the full list of possible raw data samples found by the CLOSURE algorithm.

Usage

```
closure_plot_bar(
  data,
  frequency = c("absolute-percent", "absolute", "relative", "percent"),
  samples = c("mean", "all"),
  bar_alpha = 0.75,
  bar_color = "#5D3FD3",
  show_text = TRUE,
  text_color = bar_color,
  text_size = 12,
  text_offset = 0.05,
  mark_thousand = ",",
  mark_decimal = "."
)
```

Arguments

`data`	List returned by `closure_generate()`.
`frequency`	String (length 1). What should the bars display? The default, `"absolute-percent"`, displays the count of each scale value and its percentage of all values. Other options are `"absolute"`, `"relative"`, and `"percent"`.
`samples`	String (length 1). How to aggregate the samples? Either take the average sample (`"mean"`, the default) or the sum of all samples (`"all"`). This only matters if absolute frequencies are shown.
`bar_alpha`	Numeric (length 1). Opacity of the bars. Default is `0.75`.
`bar_color`	String (length 1). Color of the bars. Default is `"#5D3FD3"`, a purple color.
`show_text`	Logical (length 1). Should the bars be labeled with the corresponding frequencies? Default is `TRUE`.
`text_color`	String (length 1). Color of the frequency labels. By default, the same as `bar_color`.
`text_size`	Numeric. Base font size in pt. Default is `12`.
`text_offset`	Numeric (length 1). Distance between the text labels and the bars. Default is `0.05`.
`mark_thousand`, `mark_decimal`	Strings (length 1 each). Delimiters between groups of digits in text labels. Defaults are `","` for `mark_thousand` (e.g., `"20,000"`) and `"."` for `mark_decimal` (e.g., `"0.15"`).

Value

A ggplot object.

See Also

`closure_plot_ecdf()`, an alternative visualization.

Examples

```
# Create CLOSURE data first:
data <- closure_generate(
  mean = "3.5",
  sd = "2",
  n = 52,
  scale_min = 1,
  scale_max = 5
)

# Visualize:
closure_plot_bar(data)
```

---

closure_plot_ecdf *Visualize CLOSURE data in an ECDF plot*

Description

Call `closure_plot_ecdf()` to visualize CLOSURE results using the data's empirical cumulative distribution function (ECDF).

A diagonal reference line benchmarks the ECDF against a hypothetical linear relationship.

See `closure_plot_bar()` for more intuitive visuals.

Usage

```
closure_plot_ecdf(
  data,
  samples = c("mean", "all"),
  line_color = "#5D3FD3",
  text_size = 12,
  reference_line_alpha = 0.6,
  pad = TRUE
)
```

Arguments

`data`	List returned by `closure_generate()`.
`samples`	String (length 1). How to aggregate the samples? Either draw a single ECDF line for the average sample (`"mean"`, the default); or draw a separate line for each sample (`"all"`). Note: the latter option can be very slow if many values were found.
`line_color`	String (length 1). Color of the ECDF line. Default is `"#5D3FD3"`, a purple color.
`text_size`	Numeric. Base font size in pt. Default is `12`.
`reference_line_alpha`	Numeric (length 1). Opacity of the diagonal reference line. Default is `0.6`.
`pad`	Logical (length 1). Should the ECDF line be padded on the x-axis so that it stretches beyond the data points? Default is `TRUE`.

Details

The present function was inspired by `rsprite2::plot_distributions()`. However, `plot_distributions()` shows multiple lines because it is based on SPRITE, which draws random samples of possible datasets. CLOSURE is exhaustive, so `closure_plot_ecdf()` shows all possible datasets in a single line by default.

Value

A ggplot object.

Examples

```
# Create CLOSURE data first:
data <- closure_generate(
  mean = "3.5",
  sd = "2",
  n = 52,
  scale_min = 1,
  scale_max = 5
)

# Visualize:
closure_plot_ecdf(data)
```

---

closure_write   *Write CLOSURE results to disk (and read them back in)*

Description

You can use `closure_write()` to save the results of `closure_generate()` on your computer. A message will show the exact location.

The data are saved in a new folder as four separate files, one for each tibble in `closure_generate()`'s output. The folder is named after the parameters of `closure_generate()`.

`closure_read()` is the opposite: it reads those files back into R, recreating the original CLOSURE list. This is useful for later analyses if you don't want to re-run a lengthy `closure_generate()` call.

Usage

```
closure_write(data, path)

closure_read(path)
```

Arguments

`data`	List returned by `closure_generate()`.
`path`	String (length 1). File path where `closure_write()` will create a new folder with the results. Set it to `"."` to choose the current working directory. For `closure_read()`, the path to an existing folder with results.

Details

`closure_write()` saves the first three tibbles as CSVs, but the `"results"` tibble becomes a Parquet file. This is much faster and takes up far less disk space — roughly 1% of a CSV file with the same data. Speed and disk space can be relevant with large result sets.

Use `closure_read()` to recreate the CLOSURE list from the folder. One of the reasons why it is convenient is that opening a Parquet file requires a special reader. For a more general tool, see `nanoparquet::read_parquet()`.

Value

`closure_write()` returns the path to the new folder it created, `closure_read()` returns a list.

Folder name

The new folder's name should be sufficient to recreate its CLOSURE results. Dashes separate values, underscores replace decimal periods. For example:

```
CLOSURE-3_5-1_0-90-1-5-up_or_down-5
```

The order is the same as in `closure_generate()`:

```
  closure_generate(
    mean = "3.5",
    sd = "1.0",
    n = 90,
    scale_min = 1,
    scale_max = 5,
    rounding = "up_or_down",  # default
    threshold = 5             # default
  )
```

Examples

```
data <- closure_generate(
  mean = "2.7",
  sd = "0.6",
  n = 45,
  scale_min = 1,
  scale_max = 5
)

# You should write to a real folder instead;
# or just leave `path` unspecified. I use a
# fake folder just for this example.
path_new_folder <- closure_write(data, path = tempdir())

# In a later session, conveniently read the files
# back into R. This returns the original list,
```

```
# identical except for floating-point error.
closure_read(path_new_folder)
```

---

horns *Horns index (h)*

Description

`horns()` measures the dispersion in a sample of clamped observations based on the scale limits. It ranges from 0 to 1:

- 0 means no variation, i.e., all observations have the same value.
- 1 means that the observations are evenly split between the extremes, with none in between.

`horns_uniform()` computes the value that `horns()` would return for a uniform distribution within given scale limits. This can be useful as a point of reference for `horns()`.

These two functions create the `horns` and `horns_uniform` columns in `closure_generate()`.

`horns_rescaled()` is a version of `horns()` that is normalized by scale length, such that `0.5` always indicates a uniform distribution, independent of the number of scale points. It is meant to enable comparison across scales of different lengths, but it is harder to interpret for an individual scale. Even so, the range and the meaning of `0` and `1` are the same as for `horns()`.

Usage

```
horns(freqs, scale_min, scale_max)

horns_uniform(scale_min, scale_max)

horns_rescaled(freqs, scale_min, scale_max)
```

Arguments

`freqs`	Numeric. Vector with the frequencies (relative or absolute) of binned observations; e.g., a vector with 5 elements for a 1-5 scale.
`scale_min, scale_max`	Numeric (length 1 each). Minimum and maximum of the scale on which the values were measured. These can be lower and upper bounds (e.g., with a 1-5 Likert scale) or empirical min and max reported in an article. The latter should be preferred if available because they constrain the scale further.

Details

The horns index h is defined as:

$$h = \frac{\sum_{i=1}^{k} f_i (s_i - \bar{s})^2}{\frac{1}{4}(s_{\max} - s_{\min})^2}$$

where k is the number of scale points (i.e., the length of `freqs` here), f_i is the relative frequency of the ith scale point, s_i; $\bar{s}$ is the sample mean, $s_{\max}$ is the upper bound of the scale, and $s_{\min}$ is its lower bound.

Its name was inspired by Heathers (2017a) which defines the "horns of no confidence" as a reconstructed sample "where an incorrect, impossible or unlikely value set has all its constituents stacked into its highest or lowest bins to try meet a ludicrously high SD". In its purest form, this is a case where `horns()` returns `1`. However, note that the implications for the plausibility of any given set of summary statistics depend on the substantive context of the data (Heathers 2017b).

Value

Numeric (length 1).

Examples

```
# For simplicity, all examples use a 1-5 scale and a total N of 300.

# ---- With all values at the extremes

horns(freqs = c(300, 0, 0, 0, 0), scale_min = 1, scale_max = 5)

horns(c(150, 0, 0, 0, 150), 1, 5)

horns(c(100, 0, 0, 0, 200), 1, 5)


# ---- With some values in between

horns(c(60, 60, 60, 60, 60), 1, 5)

horns(c(200, 50, 30, 20, 0), 1, 5)

horns(c(150, 100, 50, 0, 0), 1, 5)

horns(c(100, 40, 20, 40, 100), 1, 5)
```

Index