Package: effectclass (via r-universe)

October 2, 2024

Type Package

Title Classification and Visualisation of Effects

Version 0.1.5

Description Classify effects by comparing the confidence intervals with thresholds.

License GPL-3

URL https://inbo.github.io/effectclass/

BugReports https://github.com/inbo/effectclass/issues

Depends R (>= 4.1.0)

Imports assertthat, ggplot2, plotly

Suggests crosstalk, knitr, rmarkdown, testthat

VignetteBuilder knitr

Config/checklist/communities inbo

Config/checklist/keywords classification; effect size; uncertainty; visualisation

Encoding UTF-8

Language en-GB

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.1

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

RemoteUrl https://github.com/inbo/effectclass

RemoteRef HEAD

RemoteSha a7aa52cd9b37eac733c48113f35438ed51e2dbff

Contents

`add_classification` *Add a point symbol with classification to a* `plotly` *object*

Description

See `classification()` for an explication on how the classification is done.

Usage

```
add_classification(
  p,
  x = NULL,
  y = NULL,
  ...,
  data = NULL,
  inherit = TRUE,
  sd,
  lcl = NULL,
  ucl = NULL,
  threshold,
  reference = 0,
  prob = 0.9,
  size = 20,
  link = c("identity", "log", "logit"),
  detailed = TRUE,
  signed = TRUE,
  labels = class_labels(lang = "en", detailed = detailed, signed = signed),
```

```
  text = NULL,
  hoverinfo = "text",
  ref_label = "reference",
  ref_colour = "#C04384"
)
```

Arguments

p	a plotly object
x	the x variable.
y	the y variable.
...	Arguments (i.e., attributes) passed along to the trace type. See schema() for a list of acceptable attributes for a given trace type (by going to traces -> type -> attributes). Note that attributes provided at this level may override other arguments (e.g. plot_ly(x = 1:10, y = 1:10, color = I("red"), marker = list(color = "blue"))).
data	A data frame (optional) or crosstalk::SharedData object.
inherit	inherit attributes from plot_ly()?
sd	the variable of the standard error on the link scale.
lcl	A vector of lower confidence limits.
ucl	A vector of upper confidence limits.
threshold	A vector of either 1 or 2 thresholds. A single threshold will be transformed into reference + c(-abs(threshold), abs(threshold)).
reference	The null hypothesis. Defaults to 0.
prob	The coverage of the confidence interval when calculated from the mean y and standard error sd. Note that the function assumes a normal distribution at the link scale.
size	Size of the point symbol.
link	the link between the natural scale and the link scale. Defaults to "identity".
detailed	TRUE indicates a detailed classification(); FALSE a coarse_classification(). Defaults to TRUE.
signed	TRUE indicates a signed classification; FALSE a classification with remove_sign(). Defaults to TRUE.
labels	a vector of labels for the classification hover information. See class_labels() for inspiration.
text	textual labels.
hoverinfo	Which hover information to display. Defaults to "text". When no "text" variable is specified, the function displays a formatted confidence interval.
ref_label	The label for the reference point. Will be used for the points where is.na(sd) or both is.na(lcl) and is.na(ucl).
ref_colour	The colour for the reference point.

See Also

Other plotly add-ons: add_fan(), reference_shape(), reference_text()

Examples

```
# All possible classes
z <- data.frame(
  estimate = c(-0.5, 0, 0.5, 1.5, 1, 0.5, 0, -0.5, -1, -1.5),
  sd = c(rep(0.8, 3), rep(0.3, 7))
)
z$lcl <- qnorm(0.05, z$estimate, z$sd)
z$ucl <- qnorm(0.95, z$estimate, z$sd)
classification(z$lcl, z$ucl, threshold = 1) -> z$effect
c(
  "?" = "unknown\neffect", "?+" = "potential\npositive\neffect",
  "?-" = "potential\nnegative\neffect", "~" = "no effect",
  "+" = "positive\neffect", "-" = "negative\neffect",
  "+~" = "moderate\npositive\neffect", "-~" = "moderate\nnegative\neffect",
  "++" = "strong\npositive\neffect", "--" = "strong\nnegative\neffect"
)[as.character(z$effect)] -> z$x
z$x <- factor(z$x, z$x)
z$display <- paste(
  "estimate:", format_ci(z$estimate, lcl = z$lcl, ucl = z$ucl)
)

# Simulated trend
set.seed(20190521)
base_year <- 2000
n_year <- 20
trend <- data.frame(
  dt = seq_len(n_year),
  change = rnorm(n_year, sd = 0.2),
  sd = rnorm(n_year, mean = 0.1, sd = 0.01)
)
trend$index <- cumsum(trend$change)
trend$lcl <- qnorm(0.025, trend$index, trend$sd)
trend$ucl <- qnorm(0.975, trend$index, trend$sd)
trend$year <- base_year + trend$dt
trend$display <- paste(
  "index:", format_ci(trend$index, lcl = trend$lcl, ucl = trend$ucl)
)
th <- 0.25
ref <- 0
library(plotly)
plot_ly(z, x = ~x, y = ~estimate) |>
  add_fan(sd = ~sd, text = ~display) |>
  add_classification(lcl = ~lcl, ucl = ~ucl, threshold = 1) |>
  layout(
    hovermode = "x unified",
    shapes = reference_shape(threshold = 1),
    annotations = reference_text(threshold = 1)
  )
```

```
plot_ly(z, x = ~x, y = ~estimate) |>
  add_fan(sd = ~sd, step = 0.1, text = ~display) |>
  add_classification(
    lcl = ~lcl, ucl = ~ucl, threshold = 1, detailed = FALSE
   ) |>
  layout(
    shapes = reference_shape(threshold = 1, line = TRUE),
    annotations = reference_text(threshold = 1)
   )
plot_ly(z, x = ~x, y = ~estimate) |>
  add_fan(sd = ~sd, step = 0.2, hoverinfo = "none") |>
  add_classification(
    lcl = ~lcl, ucl = ~ucl, threshold = 1, signed = FALSE
  ) |>
  layout(shapes = reference_shape(threshold = 1))
plot_ly(z, x = ~x, y = ~estimate) |>
  add_fan(sd = ~sd, step = 0.3) |>
  add_classification(
    lcl = ~lcl, ucl = ~ucl, threshold = 1, detailed = FALSE, signed = FALSE,
    text = ~display
  ) |>
  layout(
    shapes = reference_shape(threshold = 1, line = TRUE)
  )

# trend
plot_ly(data = trend, x = ~year, y = ~index) |>
  add_fan(sd = ~sd, text = ~display, hoverinfo = "text") |>
  add_classification(sd = ~sd, threshold = th) |>
  layout(
    hovermode = "x unified", hoverdistance = 1,
    shapes = reference_shape(threshold = th, reference = ref),
    annotations = reference_text(threshold = th, reference = ref)
  )
```

`add_fan` *Add a fan plot to a* `plotly` *object*

Description

A fan plot consist of a set of transparent ribbons each representing a different coverage of the uncertainty around an estimate. The coverages are based on the assumption of a normal distribution with mean `link(y)` and standard error `sd`.

Usage

```
add_fan(
  p,
  x = NULL,
  y = NULL,
```

```
  ...,
  sd,
  link = c("identity", "log", "logit"),
  max_prob = 0.9,
  step = 0.05,
  fillcolor = coarse_unsigned_palette[2],
  data = NULL,
  inherit = TRUE,
  text = NULL,
  hoverinfo = "text",
  name
)
```

Arguments

p	a plotly object
x	the x variable.
y	the variable median on the natural scale.
...	Arguments (i.e., attributes) passed along to the trace type. See schema() for a list of acceptable attributes for a given trace type (by going to traces -> type -> attributes). Note that attributes provided at this level may override other arguments (e.g. plot_ly(x = 1:10, y = 1:10, color = I("red"), marker = list(color = "blue"))).
sd	the variable of the standard error on the link scale.
link	the link between the natural scale and the link scale. Defaults to "identity".
max_prob	The coverage of the widest band. Defaults to 0.9.
step	the step size between consecutive bands. The function adds all bands with coverage max_prob - i * step for all positive integer values i resulting in a positive coverage. Defaults to 0.05.
fillcolor	The fill colour of the fan. Defaults to a greyish blue.
data	A data frame (optional) or crosstalk::SharedData object.
inherit	inherit attributes from plot_ly()?
text	textual labels.
hoverinfo	Which hover information to display. Defaults to "text". When no "text" variable is specified, the function displays a formatted confidence interval.
name	Optional name of the trace for the legend.

See Also

Other plotly add-ons: add_classification(), reference_shape(), reference_text()

Examples

```
# All possible classes
z <- data.frame(
```

```
  estimate = c(-0.5, 0, 0.5, 1.5, 1, 0.5, 0, -0.5, -1, -1.5),
  sd = c(rep(0.8, 3), rep(0.3, 7))
)
z$lcl <- qnorm(0.05, z$estimate, z$sd)
z$ucl <- qnorm(0.95, z$estimate, z$sd)
classification(z$lcl, z$ucl, threshold = 1) -> z$effect
c(
  "?" = "unknown\neffect", "?+" = "potential\npositive\neffect",
  "?-" = "potential\nnegative\neffect", "~" = "no effect",
  "+" = "positive\neffect", "-" = "negative\neffect",
  "+~" = "moderate\npositive\neffect", "-~" = "moderate\nnegative\neffect",
  "++" = "strong\npositive\neffect", "--" = "strong\nnegative\neffect"
)[as.character(z$effect)] -> z$x
z$x <- factor(z$x, z$x)
z$display <- paste(
  "estimate:", format_ci(z$estimate, lcl = z$lcl, ucl = z$ucl)
)

# Simulated trend
set.seed(20190521)
base_year <- 2000
n_year <- 20
trend <- data.frame(
  dt = seq_len(n_year),
  change = rnorm(n_year, sd = 0.2),
  sd = rnorm(n_year, mean = 0.1, sd = 0.01)
)
trend$index <- cumsum(trend$change)
trend$lcl <- qnorm(0.025, trend$index, trend$sd)
trend$ucl <- qnorm(0.975, trend$index, trend$sd)
trend$year <- base_year + trend$dt
trend$display <- paste(
  "index:", format_ci(trend$index, lcl = trend$lcl, ucl = trend$ucl)
)
th <- 0.25
ref <- 0
library(plotly)
plot_ly(z, x = ~x, y = ~estimate) |>
  add_fan(sd = ~sd, text = ~display) |>
  add_classification(lcl = ~lcl, ucl = ~ucl, threshold = 1) |>
  layout(
    hovermode = "x unified",
    shapes = reference_shape(threshold = 1),
    annotations = reference_text(threshold = 1)
  )
plot_ly(z, x = ~x, y = ~estimate) |>
  add_fan(sd = ~sd, step = 0.1, text = ~display) |>
  add_classification(
    lcl = ~lcl, ucl = ~ucl, threshold = 1, detailed = FALSE
   ) |>
  layout(
    shapes = reference_shape(threshold = 1, line = TRUE),
    annotations = reference_text(threshold = 1)
```

```
   )
plot_ly(z, x = ~x, y = ~estimate) |>
  add_fan(sd = ~sd, step = 0.2, hoverinfo = "none") |>
  add_classification(
    lcl = ~lcl, ucl = ~ucl, threshold = 1, signed = FALSE
  ) |>
  layout(shapes = reference_shape(threshold = 1))
plot_ly(z, x = ~x, y = ~estimate) |>
  add_fan(sd = ~sd, step = 0.3) |>
  add_classification(
    lcl = ~lcl, ucl = ~ucl, threshold = 1, detailed = FALSE, signed = FALSE,
    text = ~display
  ) |>
  layout(
    shapes = reference_shape(threshold = 1, line = TRUE)
  )

# trend
plot_ly(data = trend, x = ~year, y = ~index) |>
  add_fan(sd = ~sd, text = ~display, hoverinfo = "text") |>
  add_classification(sd = ~sd, threshold = th) |>
  layout(
    hovermode = "x unified", hoverdistance = 1,
    shapes = reference_shape(threshold = th, reference = ref),
    annotations = reference_text(threshold = th, reference = ref)
  )
```

`change_breaks` *Logarithmic breaks for changes*

Description

Breaks a set of pretty breaks for changes.

Usage

```
change_breaks(n = 2, extra = NULL)
```

Arguments

`n`	the number of breaks on either side of the reference
`extra`	An optional vector of additional breaks. The function always appends these breaks. Use this option when you want to force this values to be a part of the breaks.

See Also

Other utils: `change_labels()`, `is_effectclass()`, `unlist()`

change_labels *Display logarithmic changes as percentage*

Description

Display logarithmic changes as percentage

Usage

```
change_labels(x)
```

Arguments

x	the logarithmic changes

See Also

Other utils: `change_breaks()`, `is_effectclass()`, `unlist()`

classification *Classify Effects by Comparing the Confidence Intervals with a Reference and Thresholds*

Description

- `++` **strong positive effect**: `max(threshold) < lcl`
- `+` **positive effect**: `reference < lcl < max(threshold)` and `max(threshold) < ucl`
- `+~` **moderate positive effect**: `reference < lcl` and `ucl < max(threshold)`
- `~` **no effect**: `min(threshold) < lcl < reference` and `reference < ucl < max(threshold)`
- `-~` **moderate negative effect**: `min(threshold) < lcl` and `ucl < reference`
- `-` **negative effect**: `lcl < min(threshold)` and `min(threshold) < ucl < reference`
- `--` **strong negative effect**: `ucl < min(threshold)`
- `?+` **potential positive effect**: `min(threshold) < lcl < reference` and `max(threshold) < ucl`
- `?-` **potential negative effect**: `lcl < min(threshold)` and `reference < ucl < max(threshold)`
- `?` **unknown effect**: `lcl < min(threshold)` and `max(threshold) < ucl`

Usage

```
classification(lcl, ucl, threshold, reference = 0)
```

Arguments

lcl	A vector of lower confidence limits.
ucl	A vector of upper confidence limits.
threshold	A vector of either 1 or 2 thresholds. A single threshold will be transformed into reference + c(-abs(threshold), abs(threshold)).
reference	The null hypothesis. Defaults to 0.

See Also

Other classification functions: coarse_classification(), remove_sign()

---

class_labels *Return a standardised set of labels for the classification*

Description

Return a standardised set of labels for the classification

Usage

```
class_labels(
  type = c("trend", "effect"),
  lang = c("en", "nl"),
  detailed = TRUE,
  signed = TRUE
)
```

Arguments

type	What type of effect. Currently available are "trend" and "effect".
lang	The language. Currently available are "en" (English) and "nl" (Dutch). Defaults to "en". Please contact the maintainer if you have suggestions for more languages.
detailed	TRUE indicates a detailed classification(); FALSE a coarse_classification(). Defaults to TRUE.
signed	TRUE indicates a signed classification; FALSE a classification with remove_sign(). Defaults to TRUE.

See Also

Other display functions: format_ci()

coarse_classification *Use a Lower Scale Classification*

Description

`coarse_classification(y)` reduces the 10 scales from `y <- classification(x)` to the 4 scales below.

- `+` **positive effect**: `reference < lcl`
- `~` **no effect**: `min(threshold) < lcl < reference` and `reference < ucl < max(threshold)`
- `-` **negative effect**: `ucl < reference`
- `?` **unknown effect**: `lcl < min(threshold)` or `max(threshold) < ucl`

`coarse_classification(y)` reduces the 6 scales from `y <- remove_sign(classification(x))` into 3 scales.

Usage

```
coarse_classification(classification)
```

Arguments

`classification`	The classification

See Also

Other classification functions: `classification()`, `remove_sign()`

format_ci *Format an Estimate and Confidence Interval as Text*

Description

The function rounds the estimate, lower and upper confidence interval to the same magnitude. The magnitude shows the width of the confidence interval with two significant digits.

Usage

```
format_ci(
  estimate,
  se,
  lcl,
  ucl,
  interval = 0.95,
  link = c("identity", "log", "logit"),
```

```
  max_digit = 4,
  percent = FALSE,
  sign = FALSE,
  change = FALSE
)
```

Arguments

estimate	The estimate in the link scale.
se	The standard error in the link scale. If missing, you must provide values for lcl and ucl.
lcl	The lower confidence limit. Ignored when se is given.
ucl	The upper confidence limit. Ignored when se is given.
interval	The coverage of the confidence interval. Only used when se is given. Defaults to 0.95 (95%).
link	The transformation of estimate, se, lcl and ucl. The appropriate back transformation is applied before formatting.
max_digit	The maximum number of significant digits to display. Defaults to 4.
percent	Display the interval as a percentage (= multiply by 100 and append %). Defaults to FALSE.
sign	Always add the sign to the text. (e.g. +1 instead of 1). Defaults to FALSE.
change	Display interval as a change. Subtract 1 after applying the link and before applying percent. Use it to display 0.9 (0.85; 0.95) as -10% (-15%; -5%). Defaults to FALSE. Implies sign == TRUE.

See Also

Other display functions: class_labels()

Examples

```
format_ci(0.512345, 1)
format_ci(0.512345, 1, interval = 0.9)
format_ci(0.512345, 1, link = "log")
format_ci(0.512345, 1, link = "logit")
format_ci(0.512345, 10)
format_ci(0.512345, 0.1)
format_ci(0.512345, 0.01)
format_ci(0.512345, 0.001)
format_ci(0.512345, 0.0001)
format_ci(0.512345, 0.00001)
format_ci(0.512345, 0.00001, max_digit = 10)
format_ci(0.512345, 0.5)
format_ci(-0.1, lcl = -0.1999, ucl = 0.1234)
format_ci(-0.1, lcl = -0.1999, ucl = 0.1234, percent = TRUE)
format_ci(-0.1, lcl = -0.1999, ucl = 0.1234, sign = TRUE)
format_ci(-0.1, lcl = -0.1999, ucl = 0.1234, percent = TRUE, sign = TRUE)
format_ci(-0.1, lcl = -0.1999, ucl = 0.1234)
```

```
format_ci(0.512345e-6, 1e-6)
format_ci(0.512345e-7, 1e-7)
format_ci(0.512345e-7, 1e-8)
format_ci(0.512345e-7, 1e-9)
format_ci(0.512345, 0.1, link = "log", percent = TRUE, change = FALSE)
format_ci(0.512345, 0.1, link = "log", percent = TRUE, change = TRUE)
format_ci(0, lcl = 0, ucl = 0)
format_ci(1, lcl = 1, ucl = 1)
```

`is_effectclass` *Check If an Object Is a Valid Effectclass Object*

Description

Check If an Object Is a Valid Effectclass Object

Usage

```
is_effectclass(x, message = c("none", "warning", "error"))
```

Arguments

`x`	The object to test.
`message`	What to do when the object is not a valid effectclass object. `"none"`: return `FALSE` with a message. `"warning"`: return `FALSE` with a `warning()`. `"error"`: return an error.

Value

A single `TRUE` or `FALSE` value.

See Also

Other utils: `change_breaks()`, `change_labels()`, `unlist()`

`reference_shape` *Create* `plotly`*references Returns a list shapes you can pass to the* `shapes` *argument of* `plotly::layout()`

Description

Create `plotly`references Returns a list shapes you can pass to the `shapes` argument of `plotly::layout()`

Usage

```
reference_shape(
  threshold,
  reference = 0,
  colour = "black",
  line = FALSE,
  horizontal = TRUE
)
```

Arguments

threshold	A vector of either 1 or 2 thresholds. A single threshold will be transformed into reference + c(-abs(threshold), abs(threshold)).
reference	The null hypothesis. Defaults to 0.
colour	The colour for the references. Defaults to "black".
line	display the threshold as a line (TRUE) or a ribbon (FALSE). Defaults to FALSE.
horizontal	Display horizontal reference when TRUE (default). Display vertical reference when FALSE.

See Also

Other plotly add-ons: add_classification(), add_fan(), reference_text()

Examples

```
# All possible classes
z <- data.frame(
  estimate = c(-0.5, 0, 0.5, 1.5, 1, 0.5, 0, -0.5, -1, -1.5),
  sd = c(rep(0.8, 3), rep(0.3, 7))
)
z$lcl <- qnorm(0.05, z$estimate, z$sd)
z$ucl <- qnorm(0.95, z$estimate, z$sd)
classification(z$lcl, z$ucl, threshold = 1) -> z$effect
c(
  "?" = "unknown\neffect", "?+" = "potential\npositive\neffect",
  "?-" = "potential\nnegative\neffect", "~" = "no effect",
  "+" = "positive\neffect", "-" = "negative\neffect",
  "+~" = "moderate\npositive\neffect", "-~" = "moderate\nnegative\neffect",
  "++" = "strong\npositive\neffect", "--" = "strong\nnegative\neffect"
)[as.character(z$effect)] -> z$x
z$x <- factor(z$x, z$x)
z$display <- paste(
  "estimate:", format_ci(z$estimate, lcl = z$lcl, ucl = z$ucl)
)

# Simulated trend
set.seed(20190521)
base_year <- 2000
n_year <- 20
```

```
trend <- data.frame(
  dt = seq_len(n_year),
  change = rnorm(n_year, sd = 0.2),
  sd = rnorm(n_year, mean = 0.1, sd = 0.01)
)
trend$index <- cumsum(trend$change)
trend$lcl <- qnorm(0.025, trend$index, trend$sd)
trend$ucl <- qnorm(0.975, trend$index, trend$sd)
trend$year <- base_year + trend$dt
trend$display <- paste(
  "index:", format_ci(trend$index, lcl = trend$lcl, ucl = trend$ucl)
)
th <- 0.25
ref <- 0
library(plotly)
plot_ly(z, x = ~x, y = ~estimate) |>
  add_fan(sd = ~sd, text = ~display) |>
  add_classification(lcl = ~lcl, ucl = ~ucl, threshold = 1) |>
  layout(
    hovermode = "x unified",
    shapes = reference_shape(threshold = 1),
    annotations = reference_text(threshold = 1)
  )
plot_ly(z, x = ~x, y = ~estimate) |>
  add_fan(sd = ~sd, step = 0.1, text = ~display) |>
  add_classification(
    lcl = ~lcl, ucl = ~ucl, threshold = 1, detailed = FALSE
   ) |>
  layout(
    shapes = reference_shape(threshold = 1, line = TRUE),
    annotations = reference_text(threshold = 1)
   )
plot_ly(z, x = ~x, y = ~estimate) |>
  add_fan(sd = ~sd, step = 0.2, hoverinfo = "none") |>
  add_classification(
    lcl = ~lcl, ucl = ~ucl, threshold = 1, signed = FALSE
  ) |>
  layout(shapes = reference_shape(threshold = 1))
plot_ly(z, x = ~x, y = ~estimate) |>
  add_fan(sd = ~sd, step = 0.3) |>
  add_classification(
    lcl = ~lcl, ucl = ~ucl, threshold = 1, detailed = FALSE, signed = FALSE,
    text = ~display
  ) |>
  layout(
    shapes = reference_shape(threshold = 1, line = TRUE)
  )

# trend
plot_ly(data = trend, x = ~year, y = ~index) |>
  add_fan(sd = ~sd, text = ~display, hoverinfo = "text") |>
  add_classification(sd = ~sd, threshold = th) |>
  layout(
```

```
  hovermode = "x unified", hoverdistance = 1,
  shapes = reference_shape(threshold = th, reference = ref),
  annotations = reference_text(threshold = th, reference = ref)
)
```

`reference_text` *Create* `plotly`*reference text Returns a list text you can pass to the* `annotations` *argument of* `plotly::layout()`

Description

Create `plotly`reference text Returns a list text you can pass to the `annotations` argument of `plotly::layout()`

Usage

```
reference_text(
  threshold,
  reference = 0,
  offset,
  text = c("reference", "important decrease", "important increase")
)
```

Arguments

`threshold`	A vector of either 1 or 2 thresholds. A single threshold will be transformed into `reference + c(-abs(threshold), abs(threshold))`.
`reference`	The null hypothesis. Defaults to 0.
`offset`	An numeric vector with the offset between `text` and the lines. In units of the `y` variable. Defaults to 10% of the difference between reference and threshold.
`text`	A character vector with three elements with the text to display on the reference line, bottom threshold line and upper threshold line. Defaults to `c("reference", "important decrease", "important increase")`.

See Also

Other plotly add-ons: `add_classification()`, `add_fan()`, `reference_shape()`

Examples

```
# All possible classes
z <- data.frame(
  estimate = c(-0.5, 0, 0.5, 1.5, 1, 0.5, 0, -0.5, -1, -1.5),
  sd = c(rep(0.8, 3), rep(0.3, 7))
)
z$lcl <- qnorm(0.05, z$estimate, z$sd)
z$ucl <- qnorm(0.95, z$estimate, z$sd)
classification(z$lcl, z$ucl, threshold = 1) -> z$effect
```

```
c(
  "?" = "unknown\neffect", "?+" = "potential\npositive\neffect",
  "?-" = "potential\nnegative\neffect", "~" = "no effect",
  "+" = "positive\neffect", "-" = "negative\neffect",
  "+~" = "moderate\npositive\neffect", "-~" = "moderate\nnegative\neffect",
  "++" = "strong\npositive\neffect", "--" = "strong\nnegative\neffect"
)[as.character(z$effect)] -> z$x
z$x <- factor(z$x, z$x)
z$display <- paste(
  "estimate:", format_ci(z$estimate, lcl = z$lcl, ucl = z$ucl)
)

# Simulated trend
set.seed(20190521)
base_year <- 2000
n_year <- 20
trend <- data.frame(
  dt = seq_len(n_year),
  change = rnorm(n_year, sd = 0.2),
  sd = rnorm(n_year, mean = 0.1, sd = 0.01)
)
trend$index <- cumsum(trend$change)
trend$lcl <- qnorm(0.025, trend$index, trend$sd)
trend$ucl <- qnorm(0.975, trend$index, trend$sd)
trend$year <- base_year + trend$dt
trend$display <- paste(
  "index:", format_ci(trend$index, lcl = trend$lcl, ucl = trend$ucl)
)
th <- 0.25
ref <- 0
library(plotly)
plot_ly(z, x = ~x, y = ~estimate) |>
  add_fan(sd = ~sd, text = ~display) |>
  add_classification(lcl = ~lcl, ucl = ~ucl, threshold = 1) |>
  layout(
    hovermode = "x unified",
    shapes = reference_shape(threshold = 1),
    annotations = reference_text(threshold = 1)
  )
plot_ly(z, x = ~x, y = ~estimate) |>
  add_fan(sd = ~sd, step = 0.1, text = ~display) |>
  add_classification(
    lcl = ~lcl, ucl = ~ucl, threshold = 1, detailed = FALSE
   ) |>
  layout(
    shapes = reference_shape(threshold = 1, line = TRUE),
    annotations = reference_text(threshold = 1)
   )
plot_ly(z, x = ~x, y = ~estimate) |>
  add_fan(sd = ~sd, step = 0.2, hoverinfo = "none") |>
  add_classification(
    lcl = ~lcl, ucl = ~ucl, threshold = 1, signed = FALSE
  ) |>
```

```
  layout(shapes = reference_shape(threshold = 1))
plot_ly(z, x = ~x, y = ~estimate) |>
  add_fan(sd = ~sd, step = 0.3) |>
  add_classification(
    lcl = ~lcl, ucl = ~ucl, threshold = 1, detailed = FALSE, signed = FALSE,
    text = ~display
  ) |>
  layout(
    shapes = reference_shape(threshold = 1, line = TRUE)
  )

# trend
plot_ly(data = trend, x = ~year, y = ~index) |>
  add_fan(sd = ~sd, text = ~display, hoverinfo = "text") |>
  add_classification(sd = ~sd, threshold = th) |>
  layout(
    hovermode = "x unified", hoverdistance = 1,
    shapes = reference_shape(threshold = th, reference = ref),
    annotations = reference_text(threshold = th, reference = ref)
  )
```

`remove_sign` *Remove the sign of a classification*

Description

- `**` **strong effect**: `++` or `--`
- `*` **effect**: `+` or `-`
- `*~` **moderate effect**: `+~` or `-~`
- `~` **no effect**: `~`
- `?+` **potential effect**: `?+` or `?-`
- `?` **unknown effect**: `?`

Usage

```
remove_sign(classification)
```

Arguments

`classification`	The classification

See Also

Other classification functions: `classification()`, `coarse_classification()`

scale_effect *A scale for effect points*

Description

A scale for effect points

Usage

```
scale_effect(
  ...,
  detailed = TRUE,
  signed = TRUE,
  fill = TRUE,
  colour = TRUE,
  drop = FALSE,
  labels = class_labels(lang = "en", detailed = detailed, signed = signed)
)
```

Arguments

...	Arguments passed on to ggplot2::scale_shape_manual values a set of aesthetic values to map data values to. The values will be matched in order (usually alphabetical) with the limits of the scale, or with breaks if provided. If this is a named vector, then the values will be matched based on the names instead. Data values that don't match will be given na.value. breaks One of: • NULL for no breaks • waiver() for the default breaks (the scale limits) • A character vector of breaks • A function that takes the limits as input and returns breaks as output na.value The aesthetic value to use for missing (NA) values
detailed	TRUE indicates a detailed classification(); FALSE a coarse_classification(). Defaults to TRUE.
signed	TRUE indicates a signed classification; FALSE a classification with remove_sign(). Defaults to TRUE.
fill	return ggplot2::scale_fill_manual()
colour	return ggplot2::scale_colour_manual()
drop	Drop unused levels. This is always FALSE. Changing this argument has no effect. We provide the argument to avoid errors in case the user sets the argument.
labels	the labels for the legend.

See Also

Other ggplot2 add-ons: stat_effect(), stat_fan()

Examples

```
# All possible classes
z <- data.frame(
  estimate = c(-0.5, 0, 0.5, 1.5, 1, 0.5, 0, -0.5, -1, -1.5),
  sd = c(rep(0.8, 3), rep(0.3, 7))
)
z$lcl <- qnorm(0.05, z$estimate, z$sd)
z$ucl <- qnorm(0.95, z$estimate, z$sd)
classification(z$lcl, z$ucl, threshold = 1) -> z$effect
c(
  "?" = "unknown\neffect", "?+" = "potential\npositive\neffect",
  "?-" = "potential\nnegative\neffect", "~" = "no effect",
  "+" = "positive\neffect", "-" = "negative\neffect",
  "+~" = "moderate\npositive\neffect", "-~" = "moderate\nnegative\neffect",
  "++" = "strong\npositive\neffect", "--" = "strong\nnegative\neffect"
)[as.character(z$effect)] -> z$x
z$x <- factor(z$x, z$x)
z$display <- paste(
  "estimate:", format_ci(z$estimate, lcl = z$lcl, ucl = z$ucl)
)

# Simulated trend
set.seed(20190521)
base_year <- 2000
n_year <- 20
trend <- data.frame(
  dt = seq_len(n_year),
  change = rnorm(n_year, sd = 0.2),
  sd = rnorm(n_year, mean = 0.1, sd = 0.01)
)
trend$index <- cumsum(trend$change)
trend$lcl <- qnorm(0.025, trend$index, trend$sd)
trend$ucl <- qnorm(0.975, trend$index, trend$sd)
trend$year <- base_year + trend$dt
trend$display <- paste(
  "index:", format_ci(trend$index, lcl = trend$lcl, ucl = trend$ucl)
)
th <- 0.25
ref <- 0
oldw <- getOption("warn")
options(warn = -1)
library(ggplot2)
theme_set(theme_grey(base_family = "Helvetica"))
update_geom_defaults("point", list(size = 5))
ggplot(z, aes(x = effect, y = estimate, ymin = lcl, ymax = ucl)) +
  stat_effect(threshold = 1) +
  coord_flip()
ggplot(z[3:5, ], aes(x = effect, y = estimate, ymin = lcl, ymax = ucl)) +
```

```
  stat_effect(threshold = 1, ref_line = "none") +
  coord_flip()
ggplot(z[3:5, ], aes(x = effect, y = estimate, ymin = lcl, ymax = ucl)) +
  stat_effect(threshold = 1, errorbar = FALSE) +
  coord_flip()

# plot indices
ggplot(trend, aes(x = year, y = index, ymin = lcl, ymax = ucl, sd = sd)) +
  geom_line() +
  stat_effect(threshold = th, reference = ref)

# plot pairwise differences
change_set <- function(z, base_year) {
  n_year <- max(z$dt)
  total_change <- lapply(
    seq_len(n_year) - 1,
    function(i) {
      if (i > 0) {
        y <- tail(z, -i)
      } else {
        y <- z
      }
      data.frame(
        from = base_year + i, to = base_year + y$dt,
        total = cumsum(y$change), sd = sqrt(cumsum(y$sd ^ 2))
      )
    }
  )
  total_change <- do.call(rbind, total_change)
  total_change <- rbind(
    total_change,
    data.frame(
      from = total_change$to, to = total_change$from,
      total = -total_change$total, sd = total_change$sd
    )
  )
  total_change$lcl <- qnorm(0.025, total_change$total, total_change$sd)
  total_change$ucl <- qnorm(0.975, total_change$total, total_change$sd)
  return(total_change)
}
head(trend, 10) |>
  change_set(base_year) |>
  ggplot(aes(x = from, y = to, ymin = lcl, ymax = ucl)) +
  stat_effect(
    threshold = th, reference = ref, aes(colour = total), ref_line = "none",
    errorbar = FALSE, shape_colour = FALSE
  ) +
  scale_colour_gradient2()
head(trend, 10) |>
  change_set(base_year) |>
  ggplot(aes(x = from, y = to, ymin = lcl, ymax = ucl)) +
  stat_effect(
    threshold = th, reference = ref, ref_line = "none", errorbar = FALSE
```

```
  )
options(warn = oldw)
```

`stat_effect` *Display points with classified effect*

Description

Display points with classified effect

Usage

```
stat_effect(
  mapping = NULL,
  data = NULL,
  position = "identity",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  ...,
  threshold,
  reference = 0,
  detailed = TRUE,
  signed = TRUE,
  shape_colour = TRUE,
  errorbar = TRUE,
  error_colour = TRUE,
  size = 6,
  labels = class_labels(lang = "en", detailed = detailed, signed = signed),
  ref_line = c("all", "ref", "none")
)
```

Arguments

`mapping`	Set of aesthetic mappings created by `aes()`. If specified and `inherit.aes = TRUE` (the default), it is combined with the default mapping at the top level of the plot. You must supply `mapping` if there is no plot mapping.
`data`	The data to be displayed in this layer. There are three options: If `NULL`, the default, the data is inherited from the plot data as specified in the call to `ggplot()`. A `data.frame`, or other object, will override the plot data. All objects will be fortified to produce a data frame. See `fortify()` for which variables will be created. A `function` will be called with a single argument, the plot data. The return value must be a `data.frame`, and will be used as the layer data. A `function` can be created from a `formula` (e.g. `~ head(.x, 10)`).

position — A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The position argument accepts the following:

- The result of calling a position function, such as position_jitter(). This method allows for passing extra arguments to the position.
- A string naming the position adjustment. To give the position as a string, strip the function name of the position_ prefix. For example, to use position_jitter(), give the position as "jitter".
- For more information and other ways to specify the position, see the layer position documentation.

na.rm — If FALSE, the default, missing values are removed with a warning. If TRUE, missing values are silently removed.

show.legend — logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display.

inherit.aes — If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. borders().

... — Other arguments passed on to layer()'s params argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the position argument, or aesthetics that are required can *not* be passed through .... Unknown arguments that are not part of the 4 categories below are ignored.

- Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, colour = "red" or linewidth = 3. The geom's documentation has an **Aesthetics** section that lists the available options. The 'required' aesthetics cannot be passed on to the params. Please note that while passing unmapped aesthetics as vectors is technically possible, the order and required length is not guaranteed to be parallel to the input data.
- When constructing a layer using a stat_*() function, the ... argument can be used to pass on parameters to the geom part of the layer. An example of this is stat_density(geom = "area", outline.type = "both"). The geom's documentation lists which parameters it can accept.
- Inversely, when constructing a layer using a geom_*() function, the ... argument can be used to pass on parameters to the stat part of the layer. An example of this is geom_area(stat = "density", adjust = 0.5). The stat's documentation lists which parameters it can accept.
- The key_glyph argument of layer() may also be passed on through .... This can be one of the functions described as key glyphs, to change the display of the layer in the legend.

threshold — A vector of either 1 or 2 thresholds. A single threshold will be transformed into reference + c(-abs(threshold), abs(threshold)).

reference — The null hypothesis. Defaults to 0.

detailed — TRUE indicates a detailed classification(); FALSE a coarse_classification(). Defaults to TRUE.

signed	TRUE indicates a signed classification; FALSE a classification with remove_sign(). Defaults to TRUE.
shape_colour	Colour the background of the labels according to the classification. Defaults to TRUE.
errorbar	Display the uncertainty as error bars. Defaults to TRUE.
error_colour	Colour the error bars according to the classification. Defaults to TRUE.
size	Size of the symbols.
labels	the labels for the legend.
ref_line	Which reference lines to display. "all" displays a dashed horizontal line at the reference and a dotted horizontal line at the threshold. "ref" displays a dashed horizontal line at the reference. "none" displays no horizontal lines.

See Also

classification

Other ggplot2 add-ons: scale_effect(), stat_fan()

Examples

```
# All possible classes
z <- data.frame(
  estimate = c(-0.5, 0, 0.5, 1.5, 1, 0.5, 0, -0.5, -1, -1.5),
  sd = c(rep(0.8, 3), rep(0.3, 7))
)
z$lcl <- qnorm(0.05, z$estimate, z$sd)
z$ucl <- qnorm(0.95, z$estimate, z$sd)
classification(z$lcl, z$ucl, threshold = 1) -> z$effect
c(
  "?" = "unknown\neffect", "?+" = "potential\npositive\neffect",
  "?-" = "potential\nnegative\neffect", "~" = "no effect",
  "+" = "positive\neffect", "-" = "negative\neffect",
  "+~" = "moderate\npositive\neffect", "-~" = "moderate\nnegative\neffect",
  "++" = "strong\npositive\neffect", "--" = "strong\nnegative\neffect"
)[as.character(z$effect)] -> z$x
z$x <- factor(z$x, z$x)
z$display <- paste(
  "estimate:", format_ci(z$estimate, lcl = z$lcl, ucl = z$ucl)
)

# Simulated trend
set.seed(20190521)
base_year <- 2000
n_year <- 20
trend <- data.frame(
  dt = seq_len(n_year),
  change = rnorm(n_year, sd = 0.2),
  sd = rnorm(n_year, mean = 0.1, sd = 0.01)
)
trend$index <- cumsum(trend$change)
```

```
trend$lcl <- qnorm(0.025, trend$index, trend$sd)
trend$ucl <- qnorm(0.975, trend$index, trend$sd)
trend$year <- base_year + trend$dt
trend$display <- paste(
  "index:", format_ci(trend$index, lcl = trend$lcl, ucl = trend$ucl)
)
th <- 0.25
ref <- 0
oldw <- getOption("warn")
options(warn = -1)
library(ggplot2)
theme_set(theme_grey(base_family = "Helvetica"))
update_geom_defaults("point", list(size = 5))
ggplot(z, aes(x = effect, y = estimate, ymin = lcl, ymax = ucl)) +
  stat_effect(threshold = 1) +
  coord_flip()
ggplot(z[3:5, ], aes(x = effect, y = estimate, ymin = lcl, ymax = ucl)) +
  stat_effect(threshold = 1, ref_line = "none") +
  coord_flip()
ggplot(z[3:5, ], aes(x = effect, y = estimate, ymin = lcl, ymax = ucl)) +
  stat_effect(threshold = 1, errorbar = FALSE) +
  coord_flip()

# plot indices
ggplot(trend, aes(x = year, y = index, ymin = lcl, ymax = ucl, sd = sd)) +
  geom_line() +
  stat_effect(threshold = th, reference = ref)

# plot pairwise differences
change_set <- function(z, base_year) {
  n_year <- max(z$dt)
  total_change <- lapply(
    seq_len(n_year) - 1,
    function(i) {
      if (i > 0) {
        y <- tail(z, -i)
      } else {
        y <- z
      }
      data.frame(
        from = base_year + i, to = base_year + y$dt,
        total = cumsum(y$change), sd = sqrt(cumsum(y$sd ^ 2))
      )
    }
  )
  total_change <- do.call(rbind, total_change)
  total_change <- rbind(
    total_change,
    data.frame(
      from = total_change$to, to = total_change$from,
      total = -total_change$total, sd = total_change$sd
    )
  )
```

```
  total_change$lcl <- qnorm(0.025, total_change$total, total_change$sd)
  total_change$ucl <- qnorm(0.975, total_change$total, total_change$sd)
  return(total_change)
}
head(trend, 10) |>
  change_set(base_year) |>
  ggplot(aes(x = from, y = to, ymin = lcl, ymax = ucl)) +
  stat_effect(
    threshold = th, reference = ref, aes(colour = total), ref_line = "none",
    errorbar = FALSE, shape_colour = FALSE
  ) +
  scale_colour_gradient2()
head(trend, 10) |>
  change_set(base_year) |>
  ggplot(aes(x = from, y = to, ymin = lcl, ymax = ucl)) +
  stat_effect(
    threshold = th, reference = ref, ref_line = "none", errorbar = FALSE
  )
options(warn = oldw)
```

`stat_fan` *Display a fan plot*

Description

A fan plot consist of a set of transparent ribbons each representing a different coverage of the uncertainty around an estimate. The coverages are based on the assumption of a normal distribution with mean `link(y)` and standard error `link_sd`.

Usage

```
stat_fan(
  mapping = NULL,
  data = NULL,
  position = "identity",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  geom = "ribbon",
  ...,
  link = c("identity", "log", "logit"),
  max_prob = 0.9,
  step = 0.05
)
```

Arguments

`mapping`	Set of aesthetic mappings created by `aes()`. If specified and `inherit.aes = TRUE` (the default), it is combined with the default mapping at the top level of the plot. You must supply `mapping` if there is no plot mapping.

`data` The data to be displayed in this layer. There are three options:

If `NULL`, the default, the data is inherited from the plot data as specified in the call to `ggplot()`.

A `data.frame`, or other object, will override the plot data. All objects will be fortified to produce a data frame. See `fortify()` for which variables will be created.

A `function` will be called with a single argument, the plot data. The return value must be a `data.frame`, and will be used as the layer data. A `function` can be created from a `formula` (e.g. `~ head(.x, 10)`).

`position` A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The `position` argument accepts the following:

- The result of calling a position function, such as `position_jitter()`. This method allows for passing extra arguments to the position.
- A string naming the position adjustment. To give the position as a string, strip the function name of the `position_` prefix. For example, to use `position_jitter()`, give the position as `"jitter"`.
- For more information and other ways to specify the position, see the layer position documentation.

`na.rm` If `FALSE`, the default, missing values are removed with a warning. If `TRUE`, missing values are silently removed.

`show.legend` logical. Should this layer be included in the legends? `NA`, the default, includes if any aesthetics are mapped. `FALSE` never includes, and `TRUE` always includes. It can also be a named logical vector to finely select the aesthetics to display.

`inherit.aes` If `FALSE`, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. `borders()`.

`geom` Use a different `geom` than the default `"ribbon"`.

`...` Other arguments passed on to `layer()`'s `params` argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the `position` argument, or aesthetics that are required can *not* be passed through `....` Unknown arguments that are not part of the 4 categories below are ignored.

- Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, `colour = "red"` or `linewidth = 3`. The geom's documentation has an **Aesthetics** section that lists the available options. The 'required' aesthetics cannot be passed on to the `params`. Please note that while passing unmapped aesthetics as vectors is technically possible, the order and required length is not guaranteed to be parallel to the input data.
- When constructing a layer using a `stat_*()` function, the `...` argument can be used to pass on parameters to the `geom` part of the layer. An example of this is `stat_density(geom = "area", outline.type = "both")`. The geom's documentation lists which parameters it can accept.
- Inversely, when constructing a layer using a `geom_*()` function, the `...` argument can be used to pass on parameters to the `stat` part of the layer.

An example of this is `geom_area(stat = "density", adjust = 0.5)`. The stat's documentation lists which parameters it can accept.

- The `key_glyph` argument of `layer()` may also be passed on through `...`. This can be one of the functions described as key glyphs, to change the display of the layer in the legend.

`link` the link function to apply on the `y` before calculating the coverage intervals. Note that `link_sd` is the standard error on the link scale, while `y` is on the natural scale. Defaults to `'identify'` which implies no transformation (`link(y) == y`). Other options are `'log'` and `'logit'`.

`max_prob` The coverage of the widest band. Defaults to `0.9`.

`step` the step size between consecutive bands. The function adds all bands with coverage `max_prob - i * step` for all positive integer values `i` resulting in a positive coverage. Defaults to `0.05`.

See Also

Other ggplot2 add-ons: `scale_effect()`, `stat_effect()`

Examples

```
set.seed(20191218)
z <- data.frame(
  year = 1990:2019,
  dx = rnorm(30, sd = 0.2),
  s = rnorm(30, 0.5, 0.01)
 )
z$index <- 3 + cumsum(z$dx)
library(ggplot2)
ggplot(z, aes(x = year, y = index, link_sd = s)) + stat_fan()
ggplot(z, aes(x = year, y = index, link_sd = s)) + stat_fan() + geom_line()
ggplot(z, aes(x = year, y = index, link_sd = s)) + stat_fan(step = 0.3)
ggplot(z, aes(x = year, y = exp(index), link_sd = s)) +
  stat_fan(link = "log") + geom_line()
ggplot(z, aes(x = year, y = plogis(index), link_sd = s)) +
  stat_fan(link = "logit") + geom_line()
ggplot(z, aes(x = year, y = index, link_sd = s)) + stat_fan(geom = "rect")
ggplot(z, aes(x = year, y = index, link_sd = s)) + stat_fan(geom = "bar")
ggplot(z, aes(x = year, y = index, link_sd = s)) +
  stat_fan(geom = "errorbar")
ggplot(z, aes(x = year, y = index, link_sd = s)) +
  stat_fan(geom = "linerange")
ggplot(z, aes(x = year, y = index, link_sd = s)) +
  stat_fan(geom = "pointrange")

z <- expand.grid(year = 1990:2019, category = c("A", "B"))
z$dx <- rnorm(60, sd = 0.1)
z$index <- rep(c(0, 2), each = 30) + cumsum(z$dx)
z$s <- rnorm(60, rep(c(0.5, 1), each = 30), 0.05)
ggplot(z, aes(x = year, y = index, link_sd = s)) + stat_fan() + geom_line() +
  facet_wrap(~category)
```

```
ggplot(z, aes(x = year, y = index, link_sd = s)) +
  stat_fan(aes(fill = category)) + geom_line(aes(colour = category))
```

`unlist` *Flatten Lists*

Description

Flatten Lists

Usage

```
unlist(x, recursive = TRUE, use.names = TRUE)
```

Arguments

`x`	an R object, typically a list or vector.
`recursive`	logical. Should unlisting be applied to list components of `x`?
`use.names`	logical. Should names be preserved?

See Also

base::unlist

Other utils: `change_breaks()`, `change_labels()`, `is_effectclass()`

Index