Package: BRcal (via r-universe)

September 15, 2024

Title Boldness-Recalibration of Binary Events

Version 1.0.0

Description Boldness-recalibration maximally spreads out probability predictions while maintaining a user specified level of calibration, facilitated the brcal() function. Supporting functions to assess calibration via Bayesian and Frequentist approaches, Maximum Likelihood Estimator (MLE) recalibration, Linear in Log Odds (LLO)-adjust via any specified parameters, and visualize results are also provided. Methodological details can be found in Guthrie & Franck (2024) <doi:10.1080/00031305.2024.2339266>.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.1

Depends R (>= 4.3)

LazyData true

Imports nloptr, fields, ggplot2, lifecycle

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

Config/testthat/edition 3

VignetteBuilder knitr

URL https://github.com/apguthrie/BRcal

BugReports https://github.com/apguthrie/BRcal/issues

NeedsCompilation no

Author Adeline P. Guthrie [aut, cre] (<https://orcid.org/0009-0000-1548-3648>), Christopher T. Franck [aut]

Maintainer Adeline P. Guthrie <apguthrie47@gmail.com>

Repository CRAN

Date/Publication 2024-09-13 19:10:02 UTC

Contents

`bayes_ms` *Bayesian Model Selection-Based Calibration Assessment*

Description

Perform Bayesian model selection-based approach to determine if a set of predicted probabilities `x` is well calibrated given the corresponding set of binary event outcomes `y` as described in Guthrie and Franck (2024).

Usage

```
bayes_ms(
  x,
  y,
  Pmc = 0.5,
  event = 1,
  optim_details = TRUE,
  epsilon = .Machine$double.eps,
  ...
)
```

Arguments

`x`	a numeric vector of predicted probabilities of an event. Must only contain values in [0,1].
`y`	a vector of outcomes corresponding to probabilities in `x`. Must only contain two unique values (one for "events" and one for "non-events"). By default, this function expects a vector of 0s (non-events) and 1s (events).
`Pmc`	The prior model probability for the calibrated model M_c.
`event`	Value in `y` that represents an "event". Default value is 1.
`optim_details`	Logical. If `TRUE`, the list returned by optim when minimizing the negative log likelihood is also returned by this function.

`epsilon` Amount by which probabilities are pushed away from 0 or 1 boundary for numerical stability. If a value in `x` < `epsilon`, it will be replaced with `epsilon`. If a value in `x` > `1-epsilon`, that value will be replaced with `1-epsilon`.

`...` Additional arguments to be passed to optim.

Details

This function compares a well calibrated model, M_c where $\delta = \gamma = 1$ to an uncalibrated model, M_u where $\delta > 0, \gamma \in \mathbb{R}$.

The posterior model probability of M_c given the observed outcomes `y` (returned as `posterior_model_prob`) is expressed as

$$P(M_c|\mathbf{y}) = \frac{P(\mathbf{y}|M_c)P(M_c)}{P(\mathbf{y}|M_c)P(M_c) + P(\mathbf{y}|M_u)P(M_u)}$$

where $P(\mathbf{y}|M_i)$ is the integrated likelihoof of `y` given M_i and $P(M_i)$ is the prior probability of model i, $i \in \{c, u\}$. By default, this function uses $P(M_c) = P(M_u) = 0.5$. To set a different prior for $P(M_c)$, use `Pmc`, and $P(M_u)$ will be set to `1 - Pmc`.

The Bayes factor (returned as `BF`) compares M_u to M_c. This value is approximated via the following large sample Bayesian Information Criteria (BIC) approximation (see Kass & Raftery 1995, Kass & Wasserman 1995)

$$BF = \frac{P(\mathbf{y}|M_u)}{P(\mathbf{y}|M_c)} =\approx exp\left\{-\frac{1}{2}(BIC_u - BIC_c)\right\}$$

where the BIC for the calibrated model (returned as `BIC_mc`) is

$$BIC_c = -2 \times log(\pi(\delta = 1, \gamma = 1|\mathbf{x}, \mathbf{y}))$$

and the BIC for the uncalibrated model (returned as `BIC_mu`) is

$$BIC_u = 2 \times log(n) - 2 \times log(\pi(\hat{\delta}_{MLE}, \hat{\gamma}_{MLE}|\mathbf{x}, \mathbf{y})).$$

Value

A list with the following attributes:

`Pmc` The prior model probability for the calibrated model M_c.

`BIC_Mc` The Bayesian Information Criteria (BIC) for the calibrated model M_c.

`BIC_Mu` The Bayesian Information Criteria (BIC) for the uncalibrated model M_u.

`BF` The Bayes Factor of uncalibrated model over calibrated model.

`posterior_model_prob`
The posterior model probability of the calibrated model M_c given the observed outcomes `y`, i.e. $P(M_c|y)$.

`MLEs` Maximum likelihood estimates for δ and γ.

`optim_details` If `optim_details = TRUE`, the list returned by optim when minimizing the negative log likelihood, includes convergence information, number of iterations, and achieved negative log likelihood value and MLEs.

References

Guthrie, A. P., and Franck, C. T. (2024) Boldness-Recalibration for Binary Event Predictions, *The American Statistician* 1-17.

Kass, R. E., and Raftery, A. E. (1995) Bayes factors. *Journal of the American Statistical Association*

Kass, R. E., and Wassermann, L. (1995) A reference bayesian test for nested hypotheses and its relationship to the schwarz criterion. *Journal of the American Statistical Association*

Examples

```
# Simulate 100 predicted probabilities
x <- runif(100)
# Simulated 100 binary event outcomes using x
y <- rbinom(100, 1, x)  # By construction, x is well calibrated.

# Use bayesian model selection approach to check calibration of x given outcomes y
bayes_ms(x, y, optim_details=FALSE)

# To specify different prior model probability of calibration, use Pmc
# Prior model prob of 0.7:
bayes_ms(x, y, Pmc=0.7)
# Prior model prob of 0.2
bayes_ms(x, y, Pmc=0.2)

# Use optim_details = TRUE to see returned info from call to optim(),
# details useful for checking convergence
bayes_ms(x, y, optim_details=TRUE)  # no convergence problems in this example

# Pass additional arguments to optim() via ... (see optim() for details)
# Specify different start values via par in optim() call, start at delta = 5, gamma = 5:
bayes_ms(x, y, optim_details=TRUE, par=c(5,5))
# Specify different optimization algorithm via method, L-BFGS-B instead of Nelder-Mead:
bayes_ms(x, y, optim_details=TRUE, method = "L-BFGS-B")  # same result

# What if events are defined by text instead of 0 or 1?
y2 <- ifelse(y==0, "Loss", "Win")
bayes_ms(x, y2, event="Win", optim_details=FALSE)  # same result

# What if we're interested in the probability of loss instead of win?
x2 <- 1 - x
bayes_ms(x2, y2, event="Loss", optim_details=FALSE)

# Push probabilities away from bounds by 0.000001
x3 <- c(runif(50, 0, 0.0001), runif(50, .9999, 1))
y3 <- rbinom(100, 1, 0.5)
bayes_ms(x3, y3, epsilon=0.000001)
```

brcal *Boldness-Recalibration for Binary Events*

Description

Perform Bayesian boldness-recalibration as specified in Guthrie and Franck (2024). Boldness-recalibration maximizes the spread in predictions (x) subject to a constraint on the minimum tolerable posterior probability of calibration (t).

Usage

```
brcal(
  x,
  y,
  t = 0.95,
  Pmc = 0.5,
  tau = FALSE,
  event = 1,
  start_at_MLEs = TRUE,
  x0 = NULL,
  lb = c(1e-05, -Inf),
  ub = c(Inf, Inf),
  maxeval = 500,
  maxtime = NULL,
  xtol_rel_inner = 1e-06,
  xtol_rel_outer = 1e-06,
  print_level = 3,
  epsilon = .Machine$double.eps,
  opts = NULL,
  optim_options = NULL
)
```

Arguments

x	a numeric vector of predicted probabilities of an event. Must only contain values in [0,1].
y	a vector of outcomes corresponding to probabilities in x. Must only contain two unique values (one for "events" and one for "non-events"). By default, this function expects a vector of 0s (non-events) and 1s (events).
t	Minimum tolerable level of calibration in [0,1].
Pmc	The prior model probability for the calibrated model M_c.
tau	Logical. If TRUE, the optimization operates on $\tau = log(\delta)$ instead of δ. See details.
event	Value in y that represents an "event". Default value is 1.
start_at_MLEs	Logical. If TRUE, the optimizer will start at x_0 = the maximum likelihood estimates for δ and γ. Otherwise, the user must specify x_0.

x0	Vector with starting locations for δ and γ. This argument is ignored when start_at_MLEs = TRUE.
lb	Vector with lower bounds for δ and γ. Use `-Inf` to indicate no lower bound.
ub	Vector with upper bounds for δ and γ. Use `Inf` to indicate no upper bound.
maxeval	Value passed to `nloptr()` to stop optimization when the number of function evaluations exceeds `maxeval`.
maxtime	Value passed to `nloptr()` to stop optimization when evaluation time (in seconds) exceeds `maxtime`.
xtol_rel_inner	Value passed to `nloptr()` to stop the inner optimization routine when the parameter estimates for δ and γ change by less than `xtol_rel_inner`.
xtol_rel_outer	Value passed to `nloptr()` to stop the outer optimization routine when the parameter estimates for δ and γ change by less than `xtol_rel_inner`.
print_level	Value passed to `nloptr()` to control how much output is printed during optimization. Default is to print the most information allowable by `nloptr()`. Specify `0` to suppress all output.
epsilon	Amount by which probabilities are pushed away from 0 or 1 boundary for numerical stability. If a value in `x` < `epsilon`, it will be replaced with `epsilon`. If a value in `x` > `1-epsilon`, that value will be replaced with `1-epsilon`.
opts	List with options to be passed to `nloptr`.
optim_options	List with options to be passed to `optim`.

Details

The objective function in boldness-recalibration is

$$f(\delta, \gamma) = -sd(\mathbf{x}')$$

and the constraint is

$$g(\delta, \gamma) = -(P(M_c|\mathbf{y}, \mathbf{x}') - t) \leq 0.$$

As both the objective and constraint functions are non-linear with respect to δ and γ, we use nloptr for this optimization rather than optim. Note that we use `x` to denote a vector of predicted probabilities, `nloptr()` uses `x` to denote the parameters being optimized. Thus, starting values for δ and γ are passed via argument `x0` and all output refers to the objective and constraint as `f(x)` and `g(x)`.

By default, this function uses the Augmented Lagrangian Algorithm (AUGLAG) (Conn et. al. 1991, Birgin and Martinez 2008) as the outer optimization routine and Sequential Least-Squares Quadratic Programming (SLSQP) (Dieter 1988, Dieter 1994) as the inner optimization routine.

Value

A list with the following attributes:

nloptr	The list returned by `nloptr()` including convergence information, number of iterations, and more.
Pmc	The prior model probability for the calibrated model M_c specified in function call.

t	Desired level of calibration in [0,1] specified in function call.
BR_params	(100∗t)% Boldness-recalibration estimates for δ and γ.
sb	The Bayesian Information Criteria (BIC) for the calibrated model M_c.
probs	Vector of (100∗t)% boldness-recalibrated probabilities.

Adjusting call to `nloptr()`

For more control over the optimization routine conducted by `nloptr()`, the user may specify their own options via the `opts` argument. Note that any objective, constraint, or gradient functions specified by the user will be overwritten by those specified in this package. See the documentation for `nloptr()` and the NLopt website for full details (https://nlopt.readthedocs.io/en/latest/).

Adjusting call to `optim()`

While `optim()` is not used for the non-linear constrained optimization for finding he boldness-recalibration parameters, it is used in the constraint function as it involves the posterior model posterior. Because of this, we do allow users to pass additional arguments to `optim` to be used in this calculation. However, rather than use the `...`, users should pass these arguments to `optim_options` via a list.

Optimizing over τ

When `tau=TRUE`, the optimization routine operates relative to $\tau = log(\delta)$ instead of δ. Specification of start location `x0` and bounds `lb`, `ub` should still be specified in terms of δ. The `brcal` function will automatically convert from δ to τ. In the returned list, `BR_params` will always report in terms of δ. However, the results returned in `nloptr` will reflect whichever scale `nloptr()` optimized on.

References

Birgin, E. G., and Martínez, J. M. (2008) Improving ultimate convergence of an augmented Lagrangian method, *Optimization Methods and Software* vol. 23, no. 2, p. 177-195.

Conn, A. R., Gould, N. I. M., and Toint, P. L. (1991) A globally convergent augmented Lagrangian algorithm for optimization with general constraints and simple bounds, *SIAM Journal of Numerical Analysis* vol. 28, no. 2, p. 545-572.

Guthrie, A. P., and Franck, C. T. (2024) Boldness-Recalibration for Binary Event Predictions, *The American Statistician* 1-17.

Johnson, S. G., The NLopt nonlinear-optimization package, https://nlopt.readthedocs.io/en/latest/.

Kraft, D. (1988) A software package for sequential quadratic programming", *Technical Report* DFVLR-FB 88-28, Institut für Dynamik der Flugsysteme, Oberpfaffenhofen.

Kraft, D. (1994) Algorithm 733: TOMP-Fortran modules for optimal control calculations, *ACM Transactions on Mathematical Software*, vol. 20, no. 3, pp. 262-281.

Examples

```
# Simulate 50 predicted probabilities
x <- runif(50)
# Simulated 50 binary event outcomes using x
y <- rbinom(50, 1, x)  # By construction, x is well calibrated.

# Perform 90% boldness-recalibration by setting t=0.9
# To suppress all output from nloptr() for each iteration use print_level=0
# (For reduced output at each iteration used print_level=1 or 2)
# To specify different starting values, use x0 and set start_at_MLEs=FALSE
brcal(x, y, t=0.9, x0=c(1,1), start_at_MLEs=FALSE, print_level=0)

# Adjust stopping criteria set max number of evaluations to 50 (maxeval) OR
# stop after 0.5 second (maxtime)
# and set optimization bounds using lb and ub
brcal(x, y, maxeval = 50, maxtime = 0.5, lb=c(0.001, 0), ub=c(10, 10), print_level=0)

# Specify different options for nloptr & optim
brcal(x, y, opts=list(xtol_abs=0.01,
                      local_opts=list(algorithm="NLOPT_LD_MMA")),
                      optim_options=list(method = "L-BFGS-B", lower = c(0, -1),
                               upper = c(10, 25), control=list(factr=0.01)),
                               print_level=0)

# Push probabilities away from bounds by 0.000001 and
# Stop outer optimization when parameters change by less than .001
x3 <- c(runif(25, 0, 0.0001), runif(25, .9999, 1))
y3 <- rbinom(50, 1, 0.5)
brcal(x3, y3, epsilon=0.000001, xtol_rel_outer = .01, print_level=0)

# See vignette for more examples
```

---

foreclosure *Foreclosure Monitoring Predictions data*

Description

Foreclosure monitoring probability predictions and the true foreclosure status pertaining of 5,000 housing transactions in 2010 from Wayne County, Michigan. These data were a randomly selected subset from data from presented in Keefe et al. (2017).

Usage

```
foreclosure
```

Format

`foreclosure`:

A data frame with 5,000 rows and 3 columns:

y sale type, 1 = foreclosure, 0 = regular sale

x predicted probabilities of foreclosure

year year of observed foreclosure or regular sale

Source

Keefe, M.J., Franck, C.T., Woodall, W.H. (2017): Monitoring foreclosure rates with a spatially risk-adjusted bernoulli cusum chart for concurrent observations. *Journal of Applied Statistics* 44(2), 325–341 doi:10.1080/02664763.2016.1169257

`hockey` *Hockey Home Team Win Predictions data*

Description

Home team win probability predictions and outcomes pertaining to the 2020-21 National Hockey League (NHL) Season. Probability predictions x were obtained from FiveThirtyEight via downloadable spreadsheet on their website (see below for link). The win/loss game results were obtained by web-scraping from NHL.com using the NHL API.

Usage

```
hockey
```

Format

`hockey`:

A data frame with 868 rows and 4 columns:

y game result, 1 = home team win, 0 = home team loss

x predicted probabilities of a home team win from FiveThirtyEight

rand uniformly random generated predicted probability of a home team from range [0.26, 0.78]

winner game result (string), "home" = home team win, "away" = home team loss

Source

https://data.fivethirtyeight.com/

lineplot *Lineplot for LLO-adjusted Probability Predictions*

Description

Function to visualize how predicted probabilities change under MLE-recalibration and boldness-recalibration.

Usage

```
lineplot(
  x = NULL,
  y = NULL,
  t_levels = NULL,
  plot_original = TRUE,
  plot_MLE = TRUE,
  df = NULL,
  Pmc = 0.5,
  event = 1,
  return_df = FALSE,
  epsilon = .Machine$double.eps,
  title = "Line Plot",
  ylab = "Probability",
  xlab = "Posterior Model Probability",
  ylim = c(0, 1),
  breaks = seq(0, 1, by = 0.2),
  thin_to = NULL,
  thin_prop = NULL,
  thin_by = NULL,
  thin_percent = deprecated(),
  seed = 0,
  optim_options = NULL,
  nloptr_options = NULL,
  ggpoint_options = list(alpha = 0.35, size = 1.5, show.legend = FALSE),
  ggline_options = list(alpha = 0.25, linewidth = 0.5, show.legend = FALSE)
)
```

Arguments

x	a numeric vector of predicted probabilities of an event. Must only contain values in [0,1].
y	a vector of outcomes corresponding to probabilities in x. Must only contain two unique values (one for "events" and one for "non-events"). By default, this function expects a vector of 0s (non-events) and 1s (events).
t_levels	Vector of desired level(s) of calibration at which to plot contours.
plot_original	Logical. If TRUE, the original probabilities passed in x are plotted.

plot_MLE — Logical. If TRUE, the MLE-recalibrated probabilities are plotted.

df — Dataframe returned by previous call to lineplot() specially formatted for use in this function. Only used for faster plotting when making minor cosmetic changes to a previous call.

Pmc — The prior model probability for the calibrated model M_c.

event — Value in y that represents an "event". Default value is 1.

return_df — Logical. If TRUE, the dataframe used to build this plot will be returned.

epsilon — Amount by which probabilities are pushed away from 0 or 1 boundary for numerical stability. If a value in x < epsilon, it will be replaced with epsilon. If a value in x > 1-epsilon, that value will be replaced with 1-epsilon.

title — Plot title.

ylab — Label for x-axis.

xlab — Label for x-axis.

ylim — Vector with bounds for y-axis, must be in [0,1].

breaks — Locations along y-axis at which to draw horizontal guidelines, passed to scale_y_continous().

thin_to — When non-null, the observations in (x,y) are randomly sampled without replacement to form a set of size thin_to.

thin_prop — When non-null, the observations in (x,y) are randomly sampled without replacement to form a set that is thin_prop * 100% of the original size of (x,y).

thin_by — When non-null, the observations in (x,y) are thinned by selecting every thin_by observation.

thin_percent — This argument is deprecated, use thin_prop instead.

seed — Seed for random thinning. Set to NULL for no seed.

optim_options — List of additional arguments to be passed to optim().

nloptr_options — List with options to be passed to nloptr().

ggpoint_options — List with options to be passed to geom_point().

ggline_options — List with options to be passed to geom_line().

Details

This function leverages ggplot() and related functions from the ggplot2 package (REF).

The goal of this function is to visualize how predicted probabilities change under different recalibration parameters. By default this function only shows how the original probabilities change after MLE recalibration. Argument t_levels can be used to specify a vector of levels of boldness-recalibration to visualize in addition to MLE recalibration.

While the x-axis shows the posterior model probabilities of each set of probabilities, note the posterior model probabilities are not in ascending or descending order. Instead, they simply follow the ordering of how one might use the BRcal package: first looking at the original predictions, then maximizing calibration, then examining how far they can spread out predictions while maintaining calibration with boldness-recalibration.

Value

If `return_df = TRUE`, a list with the following attributes is returned:

`plot`	A `ggplot` object showing how the predicted probabilities under MLE recalibration and specified levels of boldness-recalibration.
`df`	Dataframe used to create `plot`, specially formatted for use in `lineplot()`.

Otherwise just the `ggplot` object of the plot is returned.

Reusing underlying dataframe via `return_df`

While this function does not typically come with a large burden on time under moderate sample sizes, there is still a call to `optim()` under the hood for MLE recalibration and a call to `nloptr()` for each level of boldness-recalibration that could cause a bottleneck on time. With this in mind, users can specify `return_df=TRUE` to return the underlying dataframe used to build the resulting lineplot. Then, users can pass this dataframe to `df` in subsequent calls of `lineplot` to circumvent these calls to `optim` and `nloptr` and make cosmetic changes to the plot.

When `return_df=TRUE`, both the plot and the dataframe are returned in a list. The dataframe contains 6 columns:

- `probs`: the values of each predicted probability under each set
- `outcome`: the corresponding outcome for each predicted probability
- `post`: the posterior model probability of the set as a whole
- `id`: the id of each individual probability used for mapping observations between sets
- `set`: the set with which the probability belongs to
- `label`: the label used for the x-axis in the lineplot

Essentially, each set of probabilities (original, MLE-, and each level of boldness-recalibration) and outcomes are "stacked" on top of each other. The `id` tells the plotting function how to connect (with line) the same observation as is changes from the original set to MLE- or boldness-recalibration.

Thinning

Another strategy to save time when plotting is to thin the amount of data plotted. When sample sizes are large, the plot can become overcrowded and slow to plot. We provide three options for thinning: `thin_to`, `thin_prop`, and `thin_by`. By default, all three of these settings are set to `NULL`, meaning no thinning is performed. Users can only specify one thinning strategy at a time. Care should be taken in selecting a thinning approach based on the nature of your data and problem. Note that MLE recalibration and boldness-recalibration will be done using the full set.

Also note that if a thinning strategy is used with `return_df=TRUE`, the returned data frame will **only contain the reduced set** (i.e. the data *after* thinning).

Passing additional arguments to `geom_point()` and `geom_line()`

To make cosmetic changes to the points and lines plotted, users can pass a list of any desired arguments of `geom_point()` and `geom_line()` to `ggpoint_options` and `ggline_options`, respectively. These will overwrite everything passed to `geom_point()` or `geom_line()` except any aesthetic arguments in `aes()`.

References

Guthrie, A. P., and Franck, C. T. (2024) Boldness-Recalibration for Binary Event Predictions, *The American Statistician* 1-17.

Wickham, H. (2016) ggplot2: Elegant Graphics for Data Analysis. Springer-Verlag New York.

Examples

```
set.seed(28)
# Simulate 100 predicted probabilities
x <- runif(100)
# Simulated 100 binary event outcomes using x
y <- rbinom(100, 1, x)  # By construction, x is well calibrated.

# Lineplot show change in probabilities from original to MLE-recalibration to
# specified Levels of Boldness-Recalibration via t_levels
# Return a list with dataframe used to construct plot with return_df=TRUE
lp1 <- lineplot(x, y, t_levels=c(0.98, 0.95), return_df=TRUE)
lp1$plot

# Reusing the previous dataframe to save calculation time
lineplot(df=lp1$df)

# Adjust geom_point cosmetics via ggpoint
# Increase point size and change to open circles
lineplot(df=lp1$df, ggpoint_options=list(size=3, shape=4))

# Adjust geom_line cosmetics via ggline
# Increase line size and change transparencys
lineplot(df=lp1$df, ggline_options=list(linewidth=2, alpha=0.1))

# Thinning down to 75 randomly selected observation
lineplot(df=lp1$df, thin_to=75)

# Thinning down to 53% of the data
lineplot(df=lp1$df, thin_prop=0.53)

# Thinning down to every 3rd observation
lineplot(df=lp1$df, thin_by=3)

# Setting a different seed for thinning
lineplot(df=lp1$df, thin_prop=0.53, seed=47)

# Setting NO seed for thinning (plot will be different every time)
lineplot(df=lp1$df, thin_to=75, seed=NULL)
```

LLO *Linear Log Odds (LLO) Recalibration Function*

Description

LLO-adjust predicted probabilities based on specified δ and γ.

Usage

```
LLO(x, delta, gamma)
```

Arguments

`x`	a numeric vector of predicted probabilities of an event. Must only contain values in [0,1].
`delta`	numeric, must be > 0, parameter δ in LLO recalibration function.
`gamma`	numeric, parameter γ in LLO recalibration function.

Details

The Linear Log Odds (LLO) recalibration function can be written as

$$c(x_i; \delta, \gamma) = \frac{\delta x_i^\gamma}{\delta x_i^\gamma + (1 - x_i)^\gamma}$$

where x_i is a predicted probability, $\delta > 0$ and $\gamma \in \mathbb{R}$. Then $c(x_i; \delta, \gamma)$ is the corresponding LLO-adjusted probability that has been shifted by δ and scaled by γ on the log odds scale. When $\delta = \gamma = 1$, there is no shifting or scaling imposed on `x`.

Value

Vector of LLO-adjusted probabilities via specified δ and γ.

References

Turner, B., Steyvers, M., Merkle, E., Budescu, D., and Wallsten, T. (2014) Forecast aggregation via recalibration, *Machine Learning* 95, 261–289.

Gonzalez, R., and Wu, G. (1999), On the shape of probability weighting function, *Cognitive Psychology* 38, 129–66.

Examples

```
# Vector of probability predictions from 0 to 1
x1 <- seq(0, 1, by=0.1)
x1

# LLO-adjusted predictions via delta = 2, gamma = 3
x1_llo23 <- LLO(x1, 2, 3)
x1_llo23

# LLO-adjusted predictions via delta = 1, gamma = 1
x1_llo11 <- LLO(x1, 1, 1)
x1_llo11  # no change
```

```
# Create vector of 100 probability predictions
x2 <- runif(100)

# LLO-adjust via delta = 2, gamma = 3
x2_llo23 <- LLO(x2, 2, 3)

plot(x2, x2_llo23)
```

`llo_lrt` *Likelihood Ratio Test for Calibration*

Description

Perform a likelihood ratio test for if calibration a set of probability predictions, `x`, are well-calibrated given a corresponding set of binary event outcomes, `y`. See Guthrie and Franck (2024).

Usage

```
llo_lrt(
  x,
  y,
  event = 1,
  optim_details = TRUE,
  epsilon = .Machine$double.eps,
  ...
)
```

Arguments

`x`	a numeric vector of predicted probabilities of an event. Must only contain values in [0,1].
`y`	a vector of outcomes corresponding to probabilities in `x`. Must only contain two unique values (one for "events" and one for "non-events"). By default, this function expects a vector of 0s (non-events) and 1s (events).
`event`	Value in `y` that represents an "event". Default value is 1.
`optim_details`	Logical. If `TRUE`, the list returned by optim when minimizing the negative log likelihood is also returned by this function.
`epsilon`	Amount by which probabilities are pushed away from 0 or 1 boundary for numerical stability. If a value in `x` < `epsilon`, it will be replaced with `epsilon`. If a value in `x` > `1-epsilon`, that value will be replaced with `1-epsilon`.
`...`	Additional arguments to be passed to optim.

Details

This likelihood ratio test is based on the following likelihood

$$\pi(\mathbf{x}, \mathbf{y}|\delta, \gamma) = \prod_{i=1}^{n} c(x_i; \delta, \gamma)^{y_i} \left[1 - c(x_i; \delta, \gamma)\right]^{1-y_i}$$

where $c(x_i; \delta, \gamma)$ is the Linear in Log Odds (LLO) function, $\delta > 0$ is the shift parameter on the logs odds scale, and $\gamma \in \mathbb{R}$ is the scale parameter on the log odds scale.

As $\delta = \gamma = 1$ corresponds to no shift or scaling of probabilities, i.e. `x` is well calibrated given corresponding outcomes `y`. Thus the hypotheses for this test are as follows:

$$H_0 : \delta = 1, \gamma = 1 \text{ (Probabilities are well calibrated)}$$

$$H_1 : \delta \neq 1 \text{ and/or } \gamma \neq 1 \text{ (Probabilities are poorly calibrated).}$$

The likelihood ratio test statistics for H_0 is

$$\lambda_{LR} = -2log \left[\frac{\pi(\delta = 1, \gamma = 1|\mathbf{x}, \mathbf{y})}{\pi(\delta = \hat{\delta}_{MLE}, \gamma = \hat{\gamma}_{MLE}|\mathbf{x}, \mathbf{y})} \right]$$

where $\lambda_{LR} \stackrel{H_0}{\sim} \chi^2_2$ asymptotically under the null hypothesis H_0, and $\hat{\delta}_{MLE}$ and $\hat{\gamma}_{MLE}$ are the maximum likelihood estimates for δ and γ.

Value

A list with the following attributes:

`test_stat`	The test statistic λ_{LR} from the likelihood ratio test.
`pval`	The p-value from the likelihood ratio test.
`MLEs`	Maximum likelihood estimates for δ and γ.
`optim_details`	If `optim_details = TRUE`, the list returned by optim when minimizing the negative log likelihood, includes convergence information, number of iterations, and achieved negative log likelihood value and MLEs.

References

Guthrie, A. P., and Franck, C. T. (2024) Boldness-Recalibration for Binary Event Predictions, *The American Statistician* 1-17.

Examples

```
# Simulate 100 predicted probabilities
x <- runif(100)
# Simulated 100 binary event outcomes using `x`
y <- rbinom(100, 1, x)  # By construction, `x` is well calibrated.

# Run the likelihood ratio test on `x` and `y`
llo_lrt(x, y, optim_details=FALSE)
```

```
# Use optim_details = TRUE to see returned info from call to optim(),
# details useful for checking convergence
llo_lrt(x, y, optim_details=TRUE)  # no convergence problems in this example

# Use different start value in `optim()` call, start at delta = 5, gamma = 5
llo_lrt(x, y, optim_details=TRUE, par=c(5,5))

# Use `L-BFGS-B` instead of `Nelder-Mead`
llo_lrt(x, y, optim_details=TRUE, method = "L-BFGS-B")  # same result

# What if events are defined by text instead of 0 or 1?
y2 <- ifelse(y==0, "Loss", "Win")
llo_lrt(x, y2, event="Win", optim_details=FALSE)  # same result

# What if we're interested in the probability of loss instead of win?
x2 <- 1 - x
llo_lrt(x2, y2, event="Loss", optim_details=FALSE)

# Push probabilities away from bounds by 0.000001
x3 <- c(runif(50, 0, 0.0001), runif(50, .9999, 1))
y3 <- rbinom(100, 1, 0.5)
llo_lrt(x3, y3, epsilon=0.000001)
```

`mle_recal` *Recalibration via Maximum Likelihood Estimates (MLEs)*

Description

MLE recalibrate (i.e. LLO-adjust via $\hat{\delta}_{MLE}$ and $\hat{\gamma}_{MLE}$ as specified in Guthrie and Franck (2024).

Usage

```
mle_recal(x, y, probs_only = FALSE, event = 1, optim_details = TRUE, ...)
```

Arguments

`x`	a numeric vector of predicted probabilities of an event. Must only contain values in [0,1].
`y`	a vector of outcomes corresponding to probabilities in `x`. Must only contain two unique values (one for "events" and one for "non-events"). By default, this function expects a vector of 0s (non-events) and 1s (events).
`probs_only`	Logical. If `TRUE`, `mle_recal()` returns only the vector of MLE recalibrated probabilities.
`event`	Value in `y` that represents an "event". Default value is 1.
`optim_details`	Logical. If `TRUE`, the list returned by optim when minimizing the negative log likelihood is also returned by this function.
`...`	Additional arguments to be passed to optim.

Details

Given a set of probability predictions x, the corresponding MLE recalibrated set is $c(x; \hat{\delta}_{MLE}, \hat{\gamma}_{MLE})$ (see LLO).

Value

If probs_only=TRUE, mle_recal()returns a vector of MLE recalibrated probabilities. Otherwise, mle_recal() returns a list with the following attributes:

probs	The vector of MLE recalibrated probabilities.
MLEs	Maximum likelihood estimates for δ and γ.
optim_details	If optim_details = TRUE, the list returned by optim when minimizing the negative log likelihood, includes convergence information, number of iterations, and achieved negative log likelihood value and MLEs. This arguement is ignored when probs_only=TRUE.

References

Guthrie, A. P., and Franck, C. T. (2024) Boldness-Recalibration for Binary Event Predictions, *The American Statistician* 1-17.

Examples

```
# Simulate 100 predicted probabilities
x <- runif(100)
# Simulated 100 binary event outcomes using `x`
y <- rbinom(100, 1, x)

# MLE recalibrate `x`
mle_recal(x, y, optim_details=FALSE)

# Just return the vector of MLE recalibrated probabilities
x_mle <- mle_recal(x, y, optim_details=FALSE, probs_only=TRUE)
x_mle

plot(x, x_mle)

# Use optim_details = TRUE to see returned info from call to optim(),
# details useful for checking convergence
mle_recal(x, y, optim_details=TRUE)  # no convergence problems in this example

# Use different start value in `optim()` call, start at delta = 5, gamma = 5
mle_recal(x, y, optim_details=TRUE, par=c(5,5))

# Use `L-BFGS-B` instead of `Nelder-Mead`
mle_recal(x, y, optim_details=TRUE, method = "L-BFGS-B")  # same result

# What if events are defined by text instead of 0 or 1?
y2 <- ifelse(y==0, "Loss", "Win")
mle_recal(x, y2, event="Win", optim_details=FALSE)  # same result
```

```
# What if we're interested in the probability of loss instead of win?
x2 <- 1 - x
mle_recal(x2, y2, event="Loss", optim_details=FALSE)
```

---

plot_params *Draw image plot of posterior model probability surface.*

Description

Function to visualize the posterior model probability of the given set of probabilities, x, after LLO-adjustment via a grid of uniformly spaced set of δ and γ values with optional contours.

Usage

```
plot_params(
  x = NULL,
  y = NULL,
  z = NULL,
  t_levels = NULL,
  Pmc = 0.5,
  event = 1,
  k = 100,
  dlim = c(1e-04, 5),
  glim = c(1e-04, 5),
  zlim = c(0, 1),
  return_z = FALSE,
  epsilon = .Machine$double.eps,
  contours_only = FALSE,
  main = "Posterior Model Probability of Calibration",
  xlab = "delta",
  ylab = "gamma",
  optim_options = NULL,
  imgplt_options = list(legend.lab = ""),
  contour_options = list(drawlabels = TRUE, labcex = 0.6, lwd = 1, col =
    ifelse(contours_only, "black", "white"))
)
```

Arguments

x	a numeric vector of predicted probabilities of an event. Must only contain values in [0,1].
y	a vector of outcomes corresponding to probabilities in x. Must only contain two unique values (one for "events" and one for "non-events"). By default, this function expects a vector of 0s (non-events) and 1s (events).
z	Matrix returned by previous call to `plot_params()` containing posterior model probabilities across k×k grid of δ and γ. Assumes z was constructed using the same k, `dlim`, and `glim` as the current function call.

t_levels	Vector of desired level(s) of calibration at which to plot contours.
Pmc	The prior model probability for the calibrated model M_c.
event	Value in y that represents an "event". Default value is 1.
k	The number of uniformly spaced δ and γ values used to construct the k×k grid.
dlim	Vector with bounds for δ, must be finite.
glim	Vector with bounds for γ, must be finite.
zlim	Vector with bounds for posterior probability of calibration, must be in [0,1].
return_z	Logical. If TRUE, the matrix of posterior model probabilities across the specified k×k grid of δ and γ will be returned.
epsilon	Amount by which probabilities are pushed away from 0 or 1 boundary for numerical stability. If a value in x < epsilon, it will be replaced with epsilon. If a value in x > 1-epsilon, that value will be replaced with 1-epsilon.
contours_only	Logical. If TRUE, only the contours at the specified t_levels will be plotted with no color for the posterior model probability across the k×k grid of δ and γ.
main	Plot title.
xlab	Label for x-axis.
ylab	Label for x-axis.
optim_options	List of additional arguments to be passed to optim().
imgplt_options	List of additional arguments to be passed to image.plot().
contour_options	List of additional arguments to be passed to contour().

Details

This function leverages the image.plot function from the fields package and the contour function from the graphics package.

The goal of this function is to visualize how the posterior model probability changes under different recalibration parameters, as this is used in boldness-recalibration. To do so, a k by k grid of uniformly spaced potential values for δ and γ are constructed. Then x is LLO-adjusted under each pair of δ and γ values. The posterior model probability of each LLO-adjusted set is calculated and this is the quantity we use to color each grid cell in the image plot to visualize change in calibration. See below for more details on setting the grid.

By default, only the posterior model probability surface is plotted. Argument t_levels can be used to optionally add contours at specified levels of the posterior model probability of calibration. The goal of this is to help visualize different values of t at which they may want to boldness-recalibrate. To only draw the contours without the colored posterior model probability surface, users can set contours_only=TRUE.

Value

If return_z = TRUE, a list with the following attributes is returned:

z	Matrix containing posterior model probabilities across k×k grid of uniformly spaced values of δ and γ in the specified ranges dlim and glim, respectively.

dlim	Vector with bounds for δ used to construct z.
glim	Vector with bounds for γ used to construct z.
k	The number of uniformly spaced δ and γ values used to construct z

Setting grid for δ and γ

Arguments dlim and glim are used to set the bounds of the δ, γ grid and the size is dictated by argument k. Some care is required for the selection of these arguments. The goal is to determine what range of δ and γ encompasses the region of non-zero posterior probabilities of calibration. However, it is not feasible to check the entire parameter space (as it is unbounded) and even at smaller regions it can be difficult to detect the region in which non-zero posterior probabilities are produced without as very dense grid (large k), as the region is often quite small relative to the entire parameter space. This is problematic, as computation time increases as k grows.

We suggest the following scheme setting k, dlim, and glim. First, fix k at some small number, less than 20 for sake of computation time. Then, center a grid with small range around the MLEs for δ and γ for the given x and y. Increase the size of k until your grid detects approximated the probability of calibration at the MLEs that you expect. Then, expand your grid until it the region with high probability of calibration is covered or contract your grid to "zoom in" on the region. Then, increase k to create a fine grid of values.

Additionally, we caution users from including $\gamma = 0$ in the grid. This setting recalibrates all values in x to a single value which is not desirable in practice. Unless the single value is near the base rate, the set will be poorly calibrated and minimally bold, which does not align with the goal of boldness-recalibration.

Reusing matrix z via return_z

The time bottleneck for this function occurs when calculating the posterior model probabilities across the grid of parameter values. Thus it can be useful to save the resulting matrix of values to be re-used to save time when making minor cosmetic changes to your plot. If these adjustments do not change the grid bounds or density, users can set return_z=TRUE to return the underlying matrix of posterior mode probabilities for plotting. Then, instead of specifying x and y users can just pass the returned matrix as z. Note this assumes you are NOT making any changes to k, dlim, or glim. Also, it is not recommended that you construct your own matrix to pass via z as this function relies on the structure as returned by a previous call of plot_params().

Thinning

Another approach to speed up the calculations of this function is to thin the data used. However, this is generally not recommended unless the sample size is very large as the calculations of the posterior model probability may change drastically under small sample sizes. This can lead to misleading results. Under large sample sizes where thinning is used, note this is only an approximate visual of the posterior model probability.

Grid cells that show up white / inaccuracies warning message

In some cases, grid cells in the plot may show up as white instead of one of the colors from red to blue shown on the legend. A white grid cell indicates that there is no calculated posterior model probability at that cell. There are two common reasons for this: (1) that grid cell location is not covered by the z matrix used (i.e. you've adjusted the bounds without recalculating z) or (2) the values of the parameters at these locations cause the values in x to be LLO-adjusted such that they virtually equal 0 or 1. This invokes the use of epsilon to push them away from these boundaries for stability. This typically happens when |gamma| is very large. However, in these extreme cases this can cause inaccuracies in this plot. For this reason, we either throw the warning message: "Probs too close to 0 or 1 under very large |gamma|" and allow the cell to be plotted as white to notify the user and avoid plotting artifacts.

Additionally, when gamma is very close to 0, we cannot directly calculate the MLEs for the grid shifted prediction and thus must use optim() to approximate them. In this case, we throw a warning to notify users there may be inaccuracies.

References

Guthrie, A. P., and Franck, C. T. (2024) Boldness-Recalibration for Binary Event Predictions, *The American Statistician* 1-17.

Nychka, D., Furrer, R., Paige, J., Sain, S. (2021). fields: Tools for spatial data. R package version 15.2, https://github.com/dnychka/fieldsRPackage.

Examples

```
# Simulate 50 predicted probabilities
set.seed(49)
x <- runif(50)
# Simulated 50 binary event outcomes using x
y <- rbinom(50, 1, x)  # By construction, x is well calibrated.

#' # Set grid density k=20
plot_params(x, y, k=20)

# Adjust bounds on delta and gamma
plot_params(x, y, k=20, dlim=c(0.001, 3), glim=c(0.01,2))

# Increase grid density via k & save z matrix for faster plotting
zmat_list <- plot_params(x, y, k=100, dlim=c(0.001, 3), glim=c(0.01,2), return_z=TRUE)

# Reuse z matrix
plot_params(z=zmat_list$z, k=100, dlim=c(0.001, 3), glim=c(0.01,2))

# Add contours at t=0.95, 0.9, and 0.8
plot_params(z=zmat_list$z, k=100, dlim=c(0.001, 3), glim=c(0.01,2), t_levels=c(0.95, 0.9, 0.8))

# Add points for 95% boldness-recalibration parameters
br95 <- brcal(x, y, t=0.95, print_level=0)
plot_params(z=zmat_list$z, k=100, dlim=c(0.001, 3), glim=c(0.01,2), t_levels=c(0.95, 0.9, 0.8))
points(br95$BR_params[1], br95$BR_params[2], pch=19, col="white")

# Change color and size of contours
plot_params(z=zmat_list$z, k=100, dlim=c(0.001, 3), glim=c(0.01,2), t_levels = c(0.99, 0.1),
contour_options=list(col="orchid", lwd=2))

# Plot contours only
```

```
plot_params(z=zmat_list$z, k=100, dlim=c(0.001, 3), glim=c(0.01,2), t_levels=c(0.95, 0.9, 0.8),
contours_only=TRUE)

# Pass arguments to image.plot()
plot_params(z=zmat_list$z, k=100, dlim=c(0.001, 3), glim=c(0.01,2),
            imgplt_options=list(horizontal = TRUE, nlevel=10,
            legend.lab="Posterior Model Prob"))

# See vignette for more examples
```

Index