Package 'probcal'

July 9, 2026

Title Calibration of Binary and Multiclass Probabilities

Version 0.2.0

Description Provides S3 calibrators, metrics, and diagnostics for binary and multiclass probability calibration. Binary methods include Platt scaling, temperature scaling, beta calibration, histogram binning, and isotonic regression. Multiclass methods include temperature scaling, vector scaling, Dirichlet calibration, and a one-vs-rest wrapper for the binary calibrators. A calibration-inference layer adds debiased calibration errors, bootstrap confidence intervals, and a kernel calibration hypothesis test for binary and multiclass predictions, including the strong (canonical) multiclass case. Methods follow Platt (1999, ISBN:9780262194488), Zadrozny and Elkan (2002) <doi:10.1145/775047.775151>, Guo et al. (2017) <https://proceedings.mlr.press/v70/guo17a.html>, Kull et al. (2017) <doi:10.1214/17-EJS1338SI>, Kull et al. (2019) <doi:10.48550/arXiv.1910.12656>, Widmann et al. (2019) <doi:10.48550/arXiv.1910.11385>, and Kumar et al. (2019) <doi:10.48550/arXiv.1909.10155>.

License MIT + file LICENSE

Depends R (>= 4.1.0)

Encoding UTF-8

Language en-US

Imports cli, ggplot2, stats, withr

Suggests betacal, dplyr, knitr, pkgdown, reticulate, rmarkdown, roxygen2, testthat (>= 3.0.0)

Config/testthat/edition 3

VignetteBuilder knitr

URL https://prdm0.github.io/probcal/, https://github.com/prdm0/probcal/

BugReports https://github.com/prdm0/probcal/issues/

Config/roxygen2/version 8.0.0

NeedsCompilation no

Author Pedro Rafael Diniz Marinho [aut, cre, cph] (ORCID:
<https://orcid.org/0000-0003-1591-8300>)

Maintainer Pedro Rafael Diniz Marinho <pedro.rafael.marinho@gmail.com>

Repository CRAN

Date/Publication 2026-07-09 09:20:02 UTC

Contents

`ace` *Average Calibration Error*

Description

`ace()` returns the empirical unweighted mean absolute calibration gap over non-empty equal-width bins. Unlike `ece()`, each non-empty bin contributes equally. For multiclass inputs the `"classwise"` form averages the binary ACE over the one-vs-rest columns and the `"confidence"` form uses the top-label confidence.

Usage

```
ace(p, y, bins = 10, type = c("classwise", "confidence"))
```

Arguments

p	Predicted probabilities. A numeric vector in [0, 1] for binary problems, or a numeric matrix with one column per class for multiclass problems. Matrix inputs must have finite entries in [0, 1], at least two columns, and rows summing to one within absolute tolerance 1e-6.
y	Outcome labels. A vector coded as 0 and 1 for binary problems, or a factor or vector of integer class codes in 1:K for multiclass problems.
bins	Number of bins on [0, 1]. Must be a single positive integer.
type	Multiclass aggregation, either "classwise" or "confidence". Ignored for binary inputs.

Details

Using the same bin notation and endpoint convention as ece(), let M be the number of non-empty bins. The binary empirical average calibration error is

$$\mathrm{ACE} = \frac{1}{M} \sum_{b:n_b>0} |\,\mathrm{acc}(b) - \mathrm{conf}(b)|.$$

Unlike ECE, ACE does not weight bins by their sample sizes. Sparse bins and dense bins therefore contribute equally once they are non-empty. This implementation uses equal-width bins on [0, 1]; it does not construct adaptive or equal-frequency bins. For a multiclass probability matrix, type = "classwise" returns the arithmetic mean of the one-vs-rest binary ACE values,

$$\mathrm{ACE}_{\mathrm{cw}} = \frac{1}{K} \sum_{k=1}^{K} \mathrm{ACE}(p_{\cdot k}, \mathbf{1}\{y_i = k\}).$$

type = "confidence" returns $\mathrm{ACE}(r, c)$ using top-label confidence and correctness.

Value

A single numeric value.

References

Niculescu-Mizil, A., & Caruana, R. (2005). Predicting good probabilities with supervised learning. Proceedings of the 22nd International Conference on Machine Learning.

Examples

```
predictions <- data.frame(
  p = c(0.10, 0.20, 0.80, 0.90),
  y = c(0, 0, 1, 1)
)

predictions |>
  dplyr::summarise(ace = ace(p, y, bins = 2))
```

`cal_beta` *Beta calibration*

Description

`cal_beta()` fits the beta calibration model `inv_logit(a * log(p) - b * log(1 - p) + c)`. Probabilities are clipped to to have lower bound `eps` and upper bound `1 - eps` before taking logarithms.

Usage

```
cal_beta(p, y, eps = 1e-15)
```

Arguments

`p`	Numeric vector of uncalibrated probabilities in `[0, 1]`.
`y`	Binary outcome vector coded as `0` and `1`.
`eps`	Clipping constant satisfying `0 < eps < 0.5`. Probabilities must first be valid values in `[0, 1]`; values below `eps` and above `1 - eps` are clipped before taking logarithms.

Details

Beta calibration treats the uncalibrated event probability p_i through two log-transformed features. Before the transformation, probabilities are clipped by

$$p_i^* = C_\epsilon(p_i) = \min\{\max(p_i, \epsilon), 1 - \epsilon\}.$$

The calibrated probability is

$$q_i = \text{logit}^{-1}\{a \log(p_i^*) - b \log(1 - p_i^*) + c\}.$$

The implementation fits an ordinary unpenalized binomial `glm()` with the original binary labels, without Platt target correction. Its linear predictor is

$$\eta_i = \gamma_0 + \gamma_1 \log(p_i^*) + \gamma_2 \log(1 - p_i^*).$$

Equivalently, the fitted coefficients minimize the binomial cross-entropy

$$-\sum_{i=1}^{n} \{y_i \log q_i + (1 - y_i) \log(1 - q_i)\}.$$

The beta-calibration parameters are the following reparameterization of the fitted `glm()` coefficients:

$$\hat{a} = \hat{\gamma}_1, \quad \hat{b} = -\hat{\gamma}_2, \quad \hat{c} = \hat{\gamma}_0.$$

Thus prediction first computes $p^*_{new} = C_\epsilon(p_{new})$ and then evaluates

$$\hat{q}(p_{new}) = \text{logit}^{-1}\{\hat{a}\log(p^*_{new}) - \hat{b}\log(1 - p^*_{new}) + \hat{c}\}.$$

The object element `coefficients` contains $(\hat{\gamma}_0, \hat{\gamma}_1, \hat{\gamma}_2)$ from `glm()`, while `a`, `b`, and `c` contain the reparameterized beta-calibration coefficients. Since $d\eta_i/dp_i = a/p_i + b/(1 - p_i)$, monotone increase on `(0, 1)` is guaranteed when $a \geq 0$ and $b \geq 0$. The implementation does not impose these constraints.

Value

A `cal_beta` object. Use `predict()` with new probabilities to obtain calibrated probabilities.

References

Kull, M., Silva Filho, T. M., & Flach, P. (2017). Beta calibration: A well-founded and easily implemented improvement on logistic calibration for binary classifiers. Electronic Journal of Statistics, 11(2), 5052-5080. doi:10.1214/17-EJS1338SI.

Examples

```
set.seed(3)
calibration <- data.frame(raw_p = stats::rbeta(120, 2, 2)) |>
  dplyr::mutate(y = rbinom(dplyr::n(), 1, raw_p))

fit <- cal_beta(calibration$raw_p, calibration$y)

calibration |>
  dplyr::mutate(calibrated = predict(fit, raw_p)) |>
  dplyr::summarise(
    raw_ece = ece(raw_p, y, bins = 10),
    calibrated_ece = ece(calibrated, y, bins = 10)
  )
```

---

cal_ci *Bootstrap confidence interval for a calibration metric*

Description

`cal_ci()` returns a percentile bootstrap confidence interval for a calibration-error metric by resampling prediction-label pairs with replacement and recomputing the metric. It works for the binned errors (`ece()`, `mce()`, `ace()`), the kernel error `mmce()`, and the squared kernel calibration error `skce()`.

Usage

```
cal_ci(
  p,
  y,
  metric = c("ece", "skce", "mmce", "mce", "ace"),
  conf_level = 0.95,
  n_boot = 999,
  bins = 10,
  ...
)
```

Arguments

p	Predicted probabilities. A numeric vector in [0, 1] for binary problems, or a numeric matrix with one column per class for multiclass problems. Matrix inputs must have finite entries in [0, 1], at least two columns, and rows summing to one within absolute tolerance 1e-6.
y	Outcome labels. A vector coded as 0 and 1 for binary problems, or a factor or vector of integer class codes in 1:K for multiclass problems.
metric	Which metric to bootstrap: one of "ece", "skce", "mmce", "mce", or "ace".
conf_level	Confidence level for the two-sided interval. A single number in (0, 1).
n_boot	Number of bootstrap resamples. A single positive integer.
bins	Number of bins for the binned metrics ("ece", "mce", "ace").
...	Additional arguments passed to the underlying metric (for example type for multiclass inputs, estimator for "skce", or debiased and strategy for "ece").

Details

For n_boot bootstrap resamples, the observations are sampled with replacement and the chosen metric is recomputed. The two-sided conf_level percentile interval uses the empirical quantiles of the bootstrap distribution at $\alpha/2$ and $1-\alpha/2$ with $\alpha = 1-$ conf_level. The point estimate is the metric on the full sample. Because the percentile interval of a biased metric need not bracket the point estimate, the reported interval is widened when necessary so that it always contains the estimate. The interval is clamped at zero from below for the nonnegative metrics (ece, mmce, mce, ace); the unbiased skce estimators can be negative, so their interval is not clamped.

Percentile bootstrap intervals make no distributional assumption, but Sun et al. (2024) show they can undercover in finite samples, most severely for models with small calibration error. Treat the interval as approximate, especially near zero.

Value

An object of class cal_ci, a list with estimate, lower, upper, conf_level, metric, and method, with a print() method.

References

Sun, Y., Chaudhari, P., Barnett, I. J., & Dobriban, E. (2024). A confidence interval for the l2 expected calibration error. arXiv:2408.08998.

See Also

skce(), cal_test(), ece(), mmce()

Examples

```
set.seed(42)
p <- stats::runif(300)
y <- rbinom(300, 1, p)

cal_ci(p, y, metric = "ece", bins = 10, n_boot = 199)
cal_ci(p, y, metric = "skce", n_boot = 199)
```

---

cal_cv *Cross-validated calibration*

Description

cal_cv() fits a calibrator with out-of-fold predictions. The function expects scores, probabilities, or logits that were already produced by a model. It does not train the underlying classifier.

Usage

```
cal_cv(
  x,
  y,
  method = c("platt", "temperature", "beta", "isotonic", "histogram", "vector",
    "dirichlet", "ovr"),
  folds = 5,
  seed = NULL,
  ...
)
```

Arguments

x	Numeric vector of uncalibrated values for binary calibration, or a numeric matrix with one column per class for multiclass calibration. Use logits for method = "temperature" and "vector", probabilities for "beta", "isotonic", "histogram", and "dirichlet", and scores or probabilities for "platt".
y	Binary outcome vector coded as 0 and 1, or a factor or vector of integer class codes in 1:K for multiclass calibration.
method	Calibration method.
folds	Number of stratified folds. Must be a single integer at least 2 and no larger than the smallest class count.
seed	Optional integer seed used only for fold assignment.
...	Additional arguments passed to the selected calibrator, such as bins for histogram binning or base_method for one-vs-rest calibration.

Details

Folds are stratified by the outcome. The returned object stores the out-of-fold calibrated probabilities and a final calibrator fitted on all observations for future prediction. Binary and multiclass problems are handled through the type of `x`. A numeric vector triggers binary calibration. A numeric matrix with one column per class triggers multiclass calibration, the out-of-fold predictions become a matrix, and the available methods are `"temperature"`, `"vector"`, `"dirichlet"`, and `"ovr"`. For `method = "ovr"`, pass the binary method through `base_method`.

Cross-validated calibration estimates how the calibration map behaves on observations not used to fit that map. Let $F_i \in \{1, \ldots, V\}$ denote the fold assigned to observation i. For each fold v, a calibrator $\hat{f}^{(-v)}$ is fitted using observations with $F_i \neq v$. The out-of-fold calibrated prediction for an observation in fold v is then

$$\hat{q}_i^{\text{oof}} = \hat{f}^{(-v)}(x_i), \quad F_i = v.$$

These out-of-fold predictions are stored in `oof_predictions` and are useful for estimating calibration metrics without evaluating a calibrator on the same observations used to fit it. In binary calibration, $\hat{q}_i^{\text{oof}}$ is a scalar event probability. In multiclass calibration, it is the row vector $(\hat{q}_{i1}^{\text{oof}}, \ldots, \hat{q}_{iK}^{\text{oof}})$ on the probability simplex. After the out-of-fold predictions are computed, a final calibrator $\hat{f}$ is fitted on all observations. The S3 `predict()` method for a `cal_cv` object uses this final calibrator for future data.

The folds are stratified by the observed labels. Setting `seed` affects only the fold assignment and restores the previous random-number state after the assignment is made. The function assumes that `x` already contains model outputs from another classifier; it does not refit that classifier inside each fold. Thus the predictions are out of fold for the calibration map only, unless `x` itself was produced out of fold by the underlying classifier.

`folds` must be at least `2` and no larger than the smallest class count. Within each class, observations are randomly permuted and assigned fold labels $1, \ldots, V, 1, \ldots$ in sequence. For multiclass inputs, column k corresponds to integer class code k; if `y` is a factor, column k corresponds to `levels(y)[k]`. For `method = "ovr"`, `base_method` is read from `...`; if it is not supplied, the default base method is `"platt"`.

Value

A `cal_cv` object. Use `predict()` to apply the final calibrator to new values. The object stores `fold_id`, `oof_predictions`, `fold_calibrators`, and `final_calibrator`. For binary calibration, `oof_predictions` is a numeric vector. For multiclass calibration, it is a numeric matrix with one row per observation and one column per class, with column names given by the class levels.

Examples

```
set.seed(7)
predictions <- data.frame(raw_p = stats::runif(120)) |>
  dplyr::mutate(y = rbinom(dplyr::n(), 1, raw_p))

fit <- cal_cv(
  predictions$raw_p,
  predictions$y,
  method = "histogram",
```

```
  folds = 3,
  bins = 5,
  seed = 1
)

predictions |>
  dplyr::mutate(calibrated = fit$oof_predictions) |>
  dplyr::summarise(ece = ece(calibrated, y, bins = 5))
```

`cal_dirichlet` *Dirichlet calibration*

Description

`cal_dirichlet()` is the multiclass generalization of beta calibration. It fits a linear map on the log of the predicted probabilities followed by a softmax, which is equivalent to a multinomial logistic regression with the log-probabilities as features. An off-diagonal and intercept regularization (ODIR) penalty shrinks the off-diagonal weights and the intercepts toward zero, which reduces overfitting risk when the number of classes is large.

Usage

```
cal_dirichlet(p, y, lambda = NULL, eps = 1e-12)
```

Arguments

`p`	Numeric matrix of uncalibrated probabilities with one row per observation and one column per class. Rows must sum to one within absolute tolerance `1e-6`.
`y`	A factor or a vector of integer class codes in `1:K`, where `K` is the number of columns of `p`.
`lambda`	Non-negative ODIR regularization strength. When `NULL` it is chosen by cross-validation.
`eps`	Clipping constant satisfying `0 < eps < 0.5`. Probabilities must first be valid values in `[0, 1]`; values below `eps` and above `1 - eps` are clipped before taking logarithms.

Details

The calibrated probabilities are computed row-wise as `softmax(log(p) %*% t(W) + b)`, where `W` is a `K` by `K` weight matrix and `b` is a length `K` intercept vector. Probabilities are clipped to to have lower bound `eps` and upper bound `1 - eps` before taking logarithms. When `lambda` is `NULL`, it is selected from a small deterministic grid by cross-validated log-likelihood.

Let p_{ik} be the uncalibrated probability assigned to class k for observation i. Each row of `p` must sum to one within absolute tolerance `1e-6`. Column k corresponds to integer class code k; if `y` is a factor, column k corresponds to `levels(y)[k]`. The entries are clipped elementwise by

$$p_{ik}^{*} = \min\{\max(p_{ik}, \epsilon), 1 - \epsilon\},$$

and transformed to $u_{ik} = \log(p^*_{ik})$. The clipped feature matrix is not renormalized; normalization occurs only after the linear map, through the final softmax. Dirichlet calibration fits a multinomial logistic regression on these log-probability features,

$$\eta_{ik} = b_k + \sum_{\ell=1}^{K} W_{k\ell} u_{i\ell},$$

followed by

$$q_{ik} = \frac{\exp(\eta_{ik})}{\sum_{m=1}^{K} \exp(\eta_{im})}.$$

With fixed λ, the fitted parameters minimize

$$-\frac{1}{n} \sum_i \log q_{iy_i} + \lambda \left(\sum_{k \neq \ell} W_{k\ell}^2 + \sum_k b_k^2 \right).$$

This is the off-diagonal and intercept regularization penalty. Diagonal weights are not penalized. For fixed `lambda`, optimization uses BFGS with analytic gradients, initial weight matrix $W = I_K$, initial bias $b = 0$, and `maxit = 500`. True-class probabilities entering logarithms are clipped to `[1e-15, 1 - 1e-15]`. The returned `weight` is a $K \times K$ matrix whose row k produces the logit for class k; `bias` is a length-K vector of intercepts. The object also stores `lambda`, `value`, and the optimizer `convergence` code.

If `lambda = NULL`, the implementation evaluates the grid `c(0, 1e-4, 1e-3, 1e-2, 1e-1)` with at most three deterministic stratified folds. Class indices are assigned to folds in their existing order. The selected value minimizes the unweighted average of the fold mean held-out negative log-likelihoods; ties choose the first grid value. If fewer than two observations are available in the smallest class during selection, the fallback value is `1e-3`. With `lambda = 0`, the multinomial softmax parameterization is not unique: adding the same linear function of the features to every class logit leaves all probabilities unchanged. The calibrated probabilities are the identified output.

Value

A `cal_dirichlet` object that also inherits from `cal_multiclass`. Use `predict()` with new probabilities to obtain calibrated probabilities.

References

Kull, M., Perello-Nieto, M., Kängsepp, M., Silva Filho, T., Song, H., & Flach, P. (2019). Beyond temperature scaling: Obtaining well-calibrated multi-class probabilities with Dirichlet calibration. Advances in Neural Information Processing Systems 32.

Examples

```
set.seed(23)
prob <- matrix(stats::runif(200 * 3), ncol = 3)
prob <- prob / rowSums(prob)
labels <- max.col(prob)
```

```
fit <- cal_dirichlet(prob, labels)
head(predict(fit, prob))
```

`cal_histogram` *Histogram binning calibration*

Description

`cal_histogram()` partitions `[0, 1]` into bins and replaces each probability with the empirical event frequency in its bin. Equal-width bins use fixed intervals. Equal-frequency bins use sample quantiles as break points.

Usage

```
cal_histogram(p, y, bins = 10, strategy = c("equal_width", "equal_freq"))
```

Arguments

`p`	Numeric vector of uncalibrated probabilities in `[0, 1]`.
`y`	Binary outcome vector coded as `0` and `1`.
`bins`	Number of bins. Must be a single positive integer.
`strategy`	Binning strategy. Use `"equal_width"` for fixed-width bins or `"equal_freq"` for quantile bins.

Details

Empty training bins inherit the empirical rate from the nearest non-empty bin. This makes prediction defined over the whole interval `[0, 1]`.

Histogram binning estimates a piecewise constant calibration map. Given distinct break points $0 = b_0 < b_1 < \cdots < b_J = 1$, the implementation uses left-closed bins. For $j < J$,

$$I_j = \{i : b_{j-1} \le p_i < b_j\},$$

and the last bin is

$$I_J = \{i : b_{J-1} \le p_i \le b_J\}.$$

The fitted value for a non-empty bin is the empirical event frequency,

$$\hat{q}_j = \frac{1}{n_j} \sum_{i \in I_j} y_i, \quad n_j = |I_j|.$$

A new probability receives the fitted value of the bin into which it falls. Values exactly on an internal break point are assigned to the bin that starts at that break point; the value `1` is assigned to the last bin.

With `strategy = "equal_width"`, the break points are equally spaced on `[0, 1]`, so $J = B$ when `bins = B`. With `strategy = "equal_freq"`, provisional break points are

$$b_j = Q_8(j/B), \quad j = 0, \ldots, B,$$

where Q_8 is the sample quantile computed by `stats::quantile(type = 8)`. The first and last break points are then forced to `0` and `1`. Duplicated break points are removed, so the actual number of bins J can be smaller than `bins`. Empty bins are assigned the value of the nearest non-empty bin by bin index; if an empty bin is equally close to two non-empty bins, the lower-index non-empty bin is used. If no non-empty bin is available, the global event rate is used as a fallback.

The returned object stores the requested `bins`, the realized `actual_bins`, `strategy`, `breaks`, per-bin fitted values in `bin_values`, training `counts`, `global_rate`, and the original call.

Value

A `cal_histogram` object. Use `predict()` with new probabilities to obtain calibrated probabilities.

References

Zadrozny, B., & Elkan, C. (2002). Transforming classifier scores into accurate multiclass probability estimates. Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. doi:10.1145/775047.775151.

Examples

```
set.seed(5)
calibration <- data.frame(raw_p = stats::runif(120)) |>
  dplyr::mutate(y = rbinom(dplyr::n(), 1, raw_p))

fit <- cal_histogram(calibration$raw_p, calibration$y, bins = 5)

calibration |>
  dplyr::mutate(calibrated = predict(fit, raw_p)) |>
  dplyr::summarise(
    raw_ece = ece(raw_p, y, bins = 5),
    calibrated_ece = ece(calibrated, y, bins = 5)
  )
```

---

cal_isotonic *Isotonic calibration*

Description

`cal_isotonic()` fits a monotone calibration curve with `stats::isoreg()`. New probabilities are calibrated by linear interpolation. Predictions below the training range use the leftmost fitted value; predictions above the range use the rightmost fitted value.

Usage

```
cal_isotonic(p, y)
```

Arguments

p	Numeric vector of uncalibrated probabilities in `[0, 1]`.
y	Binary outcome vector coded as `0` and `1`.

Details

Ties in the training probabilities are ordered with positive labels first before isotonic regression and then collapsed to a single fitted value per unique probability.

Isotonic calibration estimates a nondecreasing function g that maps raw probabilities to calibrated event probabilities. Let π be the ordering that sorts observations by increasing p_i and, for equal p_i, decreasing y_i. Thus positive labels precede negative labels within a tied probability value. The fitted values solve the projection problem

$$\min_{m_1 \leq \cdots \leq m_n} \sum_{i=1}^{n} (y_{\pi(i)} - m_i)^2.$$

The implementation uses `stats::isoreg()` for the constrained least-squares problem and clips the fitted values to `[0, 1]`. The label vector must contain at least one `0` and one `1`.

Prediction uses linear interpolation between the unique training probabilities and their fitted values. If a new probability is below the smallest training value, prediction returns the leftmost fitted value. If it is above the largest training value, prediction returns the rightmost fitted value. Training ties are collapsed to one fitted value per unique probability after the isotonic fit by averaging the fitted values within each tied group. If the training data contain a single unique probability, prediction is the resulting constant fitted value. The fitted object stores the unique probabilities in `x_thresholds`, the collapsed fitted values in `y_calibrated`, the `stats::isoreg()` object in `fit`, and the original call. Prediction uses `stats::approx(method = "linear")` with constant extrapolation at the two endpoints, so the package prediction rule is the interpolated monotone curve rather than the unmodified PAVA step function.

Value

A `cal_isotonic` object. Use `predict()` with new probabilities to obtain calibrated probabilities.

References

Zadrozny, B., & Elkan, C. (2002). Transforming classifier scores into accurate multiclass probability estimates. Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. doi:10.1145/775047.775151.

Examples

```
set.seed(4)
calibration <- data.frame(raw_p = sort(stats::runif(120))) |>
  dplyr::mutate(y = rbinom(dplyr::n(), 1, raw_p))
```

```
fit <- cal_isotonic(calibration$raw_p, calibration$y)

calibration |>
  dplyr::mutate(calibrated = predict(fit, raw_p)) |>
  dplyr::summarise(
    raw_ece = ece(raw_p, y, bins = 10),
    calibrated_ece = ece(calibrated, y, bins = 10)
  )
```

`cal_ovr` *One-vs-rest multiclass calibration*

Description

`cal_ovr()` extends any binary calibrator to a multiclass problem with the one-vs-rest reduction. For each class it fits a binary calibrator that separates that class from the others, applies the calibrators column by column, and renormalizes each row to sum to one. This is the default strategy that binning methods use for multiclass calibration.

Usage

```
cal_ovr(
  x,
  y,
  method = c("platt", "beta", "isotonic", "histogram", "temperature"),
  ...
)
```

Arguments

`x`	Numeric matrix of uncalibrated values with one row per observation and one column per class. For `method = "platt"`, entries may be arbitrary finite scores. For `"beta"`, `"isotonic"`, and `"histogram"`, entries must be probabilities in `[0, 1]`. For `"temperature"`, entries are logits.
`y`	A factor or a vector of integer class codes in `1:K`, where `K` is the number of columns of `x`.
`method`	Binary calibrator applied to each one-vs-rest problem.
`...`	Additional arguments passed to the binary calibrator, such as `bins` for `method = "histogram"`.

Details

The columns of `x` are the per-class uncalibrated values. Use scores or probabilities for `method = "platt"`, probabilities for `"beta"`, `"isotonic"`, and `"histogram"`, and binary one-vs-rest logits for `"temperature"`. Rows of `x` are not required to sum to one. Every class must appear at least once in `y`, because each one-vs-rest problem needs both labels.

For K classes, column k of `x` corresponds to integer class code k; if `y` is a factor, column k corresponds to `levels(y)[k]`. One-vs-rest calibration creates K binary labels,

$$y_i^{(k)} = \mathbf{1}\{y_i = k\}, \quad k = 1, \ldots, K.$$

A separate binary calibrator f_k is fitted to column k of `x` and the binary labels $y_i^{(k)}$. On new data, the classwise calibrated scores are

$$r_{ik} = f_k(x_{ik}).$$

Because the K binary calibrators are fitted independently, the row sums of r_{ik} need not equal one. Let $S_i = \sum_{\ell=1}^{K} r_{i\ell}$. If S_i is finite and positive, the final multiclass probabilities are renormalized by row,

$$q_{ik} = \frac{r_{ik}}{\sum_{\ell=1}^{K} r_{i\ell}}.$$

If S_i is zero or non-finite, the prediction for that row is replaced by the uniform distribution $q_{ik} = 1/K$. This fallback keeps the output on the probability simplex. The renormalization changes the individual r_{ik} values unless $S_i = 1$, so final columns should not be interpreted as the raw outputs of the independently calibrated binary problems. The renormalized probabilities are simplex-valued, but the one-vs-rest reduction does not by itself guarantee joint multiclass calibration.

Value

A `cal_ovr` object that also inherits from `cal_multiclass`. The object stores `calibrators`, `base_method`, `k`, `levels`, `input`, and the original call. Use `predict()` with a new score matrix to obtain a numeric matrix of calibrated probabilities whose rows sum to one.

References

Zadrozny, B., & Elkan, C. (2002). Transforming classifier scores into accurate multiclass probability estimates. Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. doi:10.1145/775047.775151.

Examples

```
set.seed(21)
raw <- matrix(stats::runif(150 * 3), ncol = 3)
raw <- raw / rowSums(raw)
labels <- max.col(raw)

fit <- cal_ovr(raw, labels, method = "isotonic")
calibrated <- predict(fit, raw)
head(calibrated)
```

`cal_platt` *Platt scaling*

Description

`cal_platt()` fits a logistic regression that maps an uncalibrated score to a calibrated probability. The binary targets are adjusted with Platt's target correction before fitting, which shrinks labels away from exact `0` and `1`.

Usage

```
cal_platt(x, y)
```

Arguments

`x`	Numeric vector of uncalibrated scores or raw probabilities.
`y`	Binary outcome vector coded as `0` and `1`.

Details

Let $(x_i, y_i), i = 1, \ldots, n$ be the calibration sample, where x_i is the supplied score and $y_i \in \{0, 1\}$ is the observed label. Write $n_+ = \sum_i y_i$ and $n_- = n - n_+$. Platt's correction replaces the binary labels by fractional targets. Positive labels use

$$t_+ = \frac{n_+ + 1}{n_+ + 2},$$

and negative labels use

$$t_- = \frac{1}{n_- + 2}.$$

Thus $t_i = t_+$ when $y_i = 1$ and $t_i = t_-$ when $y_i = 0$. The fitted logistic map is

$$q_i(\alpha, \beta) = \text{logit}^{-1}(\alpha + \beta x_i),$$

and (α, β) are estimated by minimizing the binomial cross-entropy with the corrected fractional targets,

$$\ell(\alpha, \beta) = -\sum_{i=1}^{n} \{t_i \log q_i(\alpha, \beta) + (1 - t_i) \log[1 - q_i(\alpha, \beta)]\}.$$

The implementation fits this model with `stats::glm()` using the formula `y_adj ~ x`. The label vector must contain at least one `0` and one `1`. The returned object stores `coefficients`, where `(Intercept)` is $\hat{\alpha}$ and `x` is $\hat{\beta}$, as well as the full `glm` object in `fit` and the corrected targets `target_pos` and `target_neg`. Prediction applies $\text{logit}^{-1}(\hat{\alpha} + \hat{\beta} x_{new})$ to new scores. The argument `x` may be a score on any real-valued scale or a raw probability, but the fitted map is always a logistic function of the supplied values. The slope is unconstrained; the fitted map is increasing in `x` only when $\hat{\beta} \geq 0$.

Value

A `cal_platt` object. Use `predict()` with new scores to obtain calibrated probabilities.

References

Platt, J. (1999). Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. In Advances in Large Margin Classifiers.

Examples

```
set.seed(1)
calibration <- data.frame(score = rnorm(120)) |>
  dplyr::mutate(
    truth = inv_logit(score),
    y = rbinom(dplyr::n(), 1, truth)
  )

fit <- cal_platt(calibration$score, calibration$y)

calibration |>
  dplyr::mutate(calibrated = predict(fit, score)) |>
  dplyr::summarise(ece = ece(calibrated, y, bins = 10))
```

`cal_temperature` *Temperature scaling*

Description

`cal_temperature()` estimates a single positive temperature parameter by minimizing the negative log-likelihood. Inputs must be logits, not probabilities. For binary probabilities, `logit()` gives the corresponding logit. For strictly positive multiclass probability rows, $z_{ik} = \log p_{ik}$ is a valid softmax logit representation, up to row-wise additive constants. If probabilities have zero entries, the user must choose and supply a transformed logit matrix, such as clipped log-probabilities. `cal_temperature()` does not accept or clip probability matrices.

Usage

```
cal_temperature(logits, y)
```

Arguments

`logits`	For binary calibration, a numeric vector of uncalibrated logits. For multiclass calibration, a numeric matrix of logits with one row per observation and one column per class.
`y`	Outcome labels. For binary calibration, a vector coded as `0` and `1`. For multiclass calibration, a factor or a vector of integer class codes in `1:K`, where `K` is the number of columns of `logits`.

Details

The function handles both binary and multiclass problems through the type of `logits`. A numeric vector triggers binary temperature scaling and the calibrated probability is `inv_logit(logits / T)`. A numeric matrix with one column per class triggers multiclass temperature scaling and the calibrated probabilities are `softmax(logits / T)`. Because dividing every logit by the same positive scalar preserves the row ordering and argmax, temperature scaling leaves the predicted class unchanged apart from existing ties and only sharpens or softens the probabilities.

In the binary case, let z_i be an uncalibrated logit. For a positive temperature T, the calibrated event probability is

$$q_i(T) = \text{logit}^{-1}(z_i/T).$$

The fitted temperature is found by a bounded one-dimensional optimization on $[10^{-3}, 10^3]$:

$$\hat{T} \in \arg\min_{10^{-3} \le T \le 10^3} -\sum_{i=1}^{n} \{y_i \log q_i(T) + (1 - y_i) \log[1 - q_i(T)]\}.$$

In the multiclass case, let z_{ik} be the logit for class k and observation i. The calibrated probabilities are

$$q_{ik}(T) = \frac{\exp(z_{ik}/T)}{\sum_{\ell=1}^{K} \exp(z_{i\ell}/T)},$$

and T is chosen by minimizing the average multiclass negative log-likelihood over the same interval,

$$L(T) = -\frac{1}{n} \sum_{i=1}^{n} \log q_{iy_i}(T).$$

For multiclass labels, column k of the logit matrix corresponds to class code k. If `y` is a factor, the stored order of `levels(y)` defines the column order. The numerical objective clips probabilities that enter logarithms to `[1e-15, 1 - 1e-15]`. The optimization uses `stats::optim()` with method `"Brent"` and initial value 1 on the bounded interval above. The returned object stores `temperature`, the optimizer `value`, and the optimizer `convergence` code; multiclass fits also store `k` and `levels`.

Values $T > 1$ soften the probability vector, while values $0 < T < 1$ make it more concentrated. Dividing all class logits by the same positive constant preserves their order, so the predicted class is unchanged apart from ties already present in the logits.

Value

A `cal_temperature` object. Use `predict()` with new logits to obtain calibrated probabilities. Multiclass objects also inherit from `cal_multiclass`.

References

Guo, C., Pleiss, G., Sun, Y., & Weinberger, K. Q. (2017). On calibration of modern neural networks. Proceedings of the 34th International Conference on Machine Learning.

Examples

```
set.seed(2)
calibration <- data.frame(logits = rnorm(120)) |>
  dplyr::mutate(
    raw_p = inv_logit(logits),
    y = rbinom(dplyr::n(), 1, raw_p)
  )

fit <- cal_temperature(calibration$logits, calibration$y)

calibration |>
  dplyr::mutate(calibrated = predict(fit, logits)) |>
  dplyr::summarise(
    raw_ece = ece(raw_p, y, bins = 10),
    calibrated_ece = ece(calibrated, y, bins = 10)
  )

# Multiclass temperature scaling with a logit matrix and integer labels.
set.seed(20)
logits <- matrix(rnorm(150 * 3), ncol = 3)
labels <- max.col(logits) # integer codes in 1:3
mc_fit <- cal_temperature(logits, labels)
head(predict(mc_fit, logits))
```

---

cal_test *Kernel calibration test*

Description

`cal_test()` tests the null hypothesis that a probabilistic classifier is calibrated, using the squared kernel calibration error of Widmann, Lindsten and Zachariah (2019) as the test statistic. Two tests are available: a bootstrap test for the quadratic unbiased estimator (`method = "bootstrap"`, the default) and an asymptotic normal test based on the linear-time unbiased estimator (`method = "asymptotic"`). The bootstrap test is the default because the quadratic estimator is more powerful than the linear one (Widmann et al. 2019, Section 7.2). Both build on the kernel machinery of `skce()` and `mmce()`.

Usage

```
cal_test(
  p,
  y,
  method = c("bootstrap", "asymptotic"),
  bandwidth = 0.2,
  n_boot = 999,
  type = c("canonical", "confidence"),
  ...
)
```

Arguments

p	Predicted probabilities. A numeric vector in `[0, 1]` for binary problems, or a numeric matrix with one column per class for multiclass problems. Matrix inputs must have finite entries in `[0, 1]`, at least two columns, and rows summing to one within absolute tolerance `1e-6`.
y	Outcome labels. A vector coded as `0` and `1` for binary problems, or a factor or vector of integer class codes in `1:K` for multiclass problems.
method	Test to perform: `"bootstrap"` (the default) for the wild-bootstrap test based on the more powerful quadratic estimator $\widehat{\mathrm{SKCE}}_{uq}$, or `"asymptotic"` for the faster $O(n)$ normal test based on the linear estimator $\widehat{\mathrm{SKCE}}_{ul}$.
bandwidth	Bandwidth of the Laplacian kernel. Either a single positive number (the default `0.2`) or the string `"median"`, which sets the bandwidth to the median of the positive pairwise distances (the median heuristic of Widmann et al. 2019, Section 7). The median heuristic is recommended for the canonical multiclass form, where the scale of the simplex distances depends on the number of classes; the fixed `0.2` is a reproducible default for the binary and confidence forms, whose confidences lie in `[0, 1]`.
n_boot	Number of bootstrap resamples for `method = "bootstrap"`. A single positive integer.
type	Multiclass calibration target tested, one of `"canonical"` (strong calibration of the full probability vector, the default) or `"confidence"` (top-label confidence). Ignored for binary vector inputs. The classwise average is not a valid test target; use `skce()` with `type = "classwise"` for a point estimate.
...	Unused; present for future extension.

Details

For binary inputs the test assesses binary calibration. For a multiclass probability matrix, `type` selects the target: `"canonical"` tests strong calibration of the full probability vector with the matrix-valued kernel, and `"confidence"` tests calibration of the top-label probability. The classwise (one-vs-rest) average is available as a point estimate from `skce()` but not as a test, because averaging per-class kernels does not yield a single positive-definite kernel with a valid null distribution for the disjoint-pair and U-statistic constructions.

The asymptotic test uses the linear estimator $\widehat{\mathrm{SKCE}}_{ul}$ over the disjoint pairs $(1, 2), (3, 4), \ldots$. Under H_0 it is asymptotically normal (Widmann et al. 2019, Lemma 3): with $\hat{\sigma}$ the sample standard deviation of the disjoint-pair terms $h_{2i-1,2i}$, the standardised statistic is $\sqrt{\lfloor n/2 \rfloor}\, \widehat{\mathrm{SKCE}}_{ul}/\hat{\sigma}$, and the one-sided p-value is $1 - \Phi(\cdot)$. This test is $O(n)$ and needs no resampling.

The bootstrap test targets the quadratic estimator $\widehat{\mathrm{SKCE}}_{uq}$, which is a degenerate U-statistic under strong calibration with no closed-form limit (Widmann et al. 2019, Theorem G.2). The null distribution is approximated by a wild bootstrap of the centred kernel terms h_{ij} with Rademacher multipliers (Arcones and Giné 1992), the analogue of the bootstrap for the maximum mean discrepancy two-sample test. The p-value is the fraction of bootstrap statistics that equal or exceed the observed estimator, with the customary add-one correction. This test is more powerful but costs $O(Bn^2)$ for `n_boot` resamples.

Two caveats are worth stating. Consistency-resampling tests for binned calibration error tend to over-reject calibrated models (Widmann et al. 2019, Section 7.2); the kernel tests here are the recommended replacement. Resampling procedures can also undercover or misstate uncertainty for models with small calibration error in finite samples (Sun et al. 2024); the asymptotic test is preferable when its normal approximation is adequate.

Value

An object of class `c("cal_test", "htest")` with components `statistic`, `p.value`, `method`, `data.name`, and `estimate` (the SKCE estimate used). It prints in the style of base R hypothesis tests.

References

Widmann, D., Lindsten, F., & Zachariah, D. (2019). Calibration tests in multi-class classification: A unifying framework. Advances in Neural Information Processing Systems 32. arXiv:1910.11385.

Sun, Y., Chaudhari, P., Barnett, I. J., & Dobriban, E. (2024). A confidence interval for the l2 expected calibration error. arXiv:2408.08998.

See Also

`skce()`, `cal_ci()`, `mmce()`

Examples

```
set.seed(40)
p <- stats::runif(300)
y <- rbinom(300, 1, p)

# Calibrated by construction: large p-value expected.
cal_test(p, y)

# Miscalibrated (overconfident) predictions: small p-value expected.
set.seed(41)
p2 <- stats::runif(300)
y2 <- rbinom(300, 1, pmin(pmax(p2 - 0.25, 0), 1))
cal_test(p2, y2)
```

---

cal_vector_scaling *Vector scaling*

Description

`cal_vector_scaling()` is the multiclass generalization of temperature scaling that gives each class its own scale and bias. It rescales a logit matrix column by column and applies the softmax. With a single shared scale and no bias it reduces to temperature scaling, so it is more flexible while remaining cheap to fit.

Usage

```
cal_vector_scaling(logits, y)
```

Arguments

`logits`	Numeric matrix of uncalibrated logits with one row per observation and one column per class.
`y`	A factor or a vector of integer class codes in `1:K`, where `K` is the number of columns of `logits`.

Details

The calibrated probabilities are `softmax(s * logits + b)`, where `s` is a length `K` vector of per-class scales applied column by column and `b` is a length `K` vector of per-class biases. Parameters are estimated by minimizing the average multiclass negative log-likelihood.

Let z_{ik} be the uncalibrated logit for observation i and class k. Vector scaling estimates class-specific scales s_k and intercepts b_k, then forms calibrated logits

$$\eta_{ik} = s_k z_{ik} + b_k.$$

The predicted probabilities are obtained with the softmax,

$$q_{ik} = \frac{\exp(\eta_{ik})}{\sum_{\ell=1}^{K} \exp(\eta_{i\ell})}.$$

Parameters are estimated by minimizing

$$L(s, b) = -\frac{1}{n} \sum_{i=1}^{n} \log q_{iy_i}.$$

For multiclass labels, column k of `logits` corresponds to class code k; if `y` is a factor, column k corresponds to `levels(y)[k]`. The implementation uses `stats::optim()` with method `"BFGS"`, analytic gradients, initial scales $s_k = 1$, initial biases $b_k = 0$, and `maxit = 500`. True-class probabilities entering logarithms are clipped to `[1e-15, 1 - 1e-15]`. The returned object stores `scale`, `bias`, the optimized average negative log-likelihood `value`, and the optimizer `convergence` code.

The scales are unconstrained in the fitted optimization, so a negative scale is possible when it improves the likelihood on the calibration data. Unlike temperature scaling, vector scaling can change the predicted class because scales and biases vary by class. As with any softmax model, adding the same constant to every class bias does not change the resulting probability vector, so the fitted bias vector is identifiable only up to a common additive constant.

Value

A `cal_vector_scaling` object that also inherits from `cal_multiclass`. Use `predict()` with new logits to obtain calibrated probabilities.

References

Guo, C., Pleiss, G., Sun, Y., & Weinberger, K. Q. (2017). On calibration of modern neural networks. Proceedings of the 34th International Conference on Machine Learning.

Examples

```
set.seed(22)
logits <- matrix(rnorm(200 * 3), ncol = 3)
labels <- max.col(logits)
fit <- cal_vector_scaling(logits, labels)
head(predict(fit, logits))
```

---

ece *Expected Calibration Error*

Description

`ece()` returns the empirical weighted average gap between mean confidence and empirical event frequency across equal-width probability bins. It is zero when confidence and accuracy match in every non-empty bin of the chosen partition.

Usage

```
ece(
  p,
  y,
  bins = 10,
  type = c("classwise", "confidence"),
  debiased = FALSE,
  strategy = c("width", "mass"),
  norm = c("l1", "l2")
)
```

Arguments

`p`	Predicted probabilities. A numeric vector in `[0, 1]` for binary problems, or a numeric matrix with one column per class for multiclass problems. Matrix inputs must have finite entries in `[0, 1]`, at least two columns, and rows summing to one within absolute tolerance `1e-6`.
`y`	Outcome labels. A vector coded as `0` and `1` for binary problems, or a factor or vector of integer class codes in `1:K` for multiclass problems.
`bins`	Number of bins on `[0, 1]`. Must be a single positive integer.
`type`	Multiclass aggregation, either `"classwise"` or `"confidence"`. Ignored for binary inputs.
`debiased`	Logical. If `FALSE` (the default) the plug-in estimate is returned. If `TRUE` the square root of the debiased squared calibration error estimator of Kumar, Liang and Ma (2019) is returned; this requires `norm = "l2"`.

`strategy` Binning strategy, either `"width"` for equal-width bins (the default, reproducing the historical behaviour) or `"mass"` for equal-frequency bins.

`norm` Scale of the calibration error: `"l1"` (the default) for the weighted mean absolute bin gap, or `"l2"` for the (root) weighted mean squared bin gap. The `debiased` correction is only defined for `"l2"`. The `"l1"` and `"l2"` values are on different scales and should not be compared directly; the `norm` and `debiased` choices are independent.

Details

For binary problems `p` is a probability vector. For multiclass problems `p` is a probability matrix with one column per class and `type` selects the multiclass definition. The `"classwise"` form averages the binary ECE over the one-vs-rest columns, also known as the static calibration error. The `"confidence"` form applies the binary ECE to the top-label confidence and whether the predicted class is correct, which is the definition used by Guo et al. (2017).

For binary calibration, the interval `[0, 1]` is split into B equal-width bins. The package uses left-closed bins, $I_b = \{i : (b-1)/B \le p_i < b/B\}$ for $b < B$, and $I_B = \{i : (B-1)/B \le p_i \le 1\}$ for the last bin. Let $n_b = |I_b|$ and $n = \sum_b n_b$. For each non-empty bin,

$$\mathrm{conf}(b) = \frac{1}{n_b} \sum_{i \in I_b} p_i,$$

and

$$\mathrm{acc}(b) = \frac{1}{n_b} \sum_{i \in I_b} y_i.$$

The returned empirical ECE is

$$\mathrm{ECE} = \sum_{b : n_b > 0} \frac{n_b}{n} |\,\mathrm{acc}(b) - \mathrm{conf}(b)|.$$

Empty bins have zero weight. The estimate depends on `bins`; changing the number of bins changes the empirical partition and can change the value. A value of zero means equality of sample bin means for this partition, not full population calibration.

For a probability matrix, `type = "classwise"` computes the binary ECE for each one-vs-rest column $p_{\cdot k}$ against $\mathbf{1}\{y_i = k\}$ and returns their arithmetic mean,

$$\mathrm{ECE}_{\mathrm{cw}} = \frac{1}{K} \sum_{k=1}^{K} \mathrm{ECE}(p_{\cdot k}, \mathbf{1}\{y_i = k\}).$$

`type = "confidence"` uses the top-label rule $\hat{y}_i = \min\{k : p_{ik} = \max_\ell p_{i\ell}\}$, the confidence $r_i = p_{i\hat{y}_i}$, and the correctness indicator $c_i = \mathbf{1}\{\hat{y}_i = y_i\}$, then applies the binary definition to (r_i, c_i): $\mathrm{ECE}_{\mathrm{conf}} = \mathrm{ECE}(r, c)$. For matrix inputs, column k corresponds to integer class code k; if `y` is a factor, column k corresponds to `levels(y)[k]`.

Here "calibrated" refers to the output of a fitted calibration map. It does not imply population calibration. Binary population calibration can be stated as $E(Y \mid Q) = Q$ for the predicted probability random variable Q. For top-label confidence R, the analogous condition is $E[\mathbf{1}\{\hat{Y} = Y\} \mid R] = R$.

Value

A single numeric value.

References

Guo, C., Pleiss, G., Sun, Y., & Weinberger, K. Q. (2017). On calibration of modern neural networks. Proceedings of the 34th International Conference on Machine Learning.

Kumar, A., Liang, P., & Ma, T. (2019). Verified uncertainty calibration. Advances in Neural Information Processing Systems 32. arXiv:1909.10155.

Roelofs, R., Cain, N., Shlens, J., & Mozer, M. C. (2022). Mitigating bias in calibration error estimation. Proceedings of the 25th International Conference on Artificial Intelligence and Statistics.

Examples

```
predictions <- data.frame(
  p = c(0.10, 0.20, 0.80, 0.90),
  y = c(0, 0, 1, 1)
)

predictions |>
  dplyr::summarise(ece = ece(p, y, bins = 2))

# Debiased squared-ECE estimate with equal-mass bins.
set.seed(33)
p <- stats::runif(500)
y <- rbinom(500, 1, p)
ece(p, y, bins = 15)
ece(p, y, bins = 15, debiased = TRUE, strategy = "mass", norm = "l2")

# Multiclass classwise ECE from a probability matrix.
set.seed(30)
prob <- matrix(stats::runif(150 * 3), ncol = 3)
prob <- prob / rowSums(prob)
labels <- max.col(prob)
ece(prob, labels, bins = 10, type = "classwise")
```

---

inv_logit *Inverse logit transformation*

Description

`inv_logit()` maps finite real values to probabilities. Mathematically the range is `(0, 1)`, although floating-point results can round to `0` or `1` for extreme finite inputs. It is used by temperature scaling and by the parametric calibrators fitted with logistic regression.

Usage

```
inv_logit(x)
```

Arguments

x	Numeric vector on the logit scale.

Details

The inverse logit, also called the logistic function, is

$$\text{logit}^{-1}(x) = \frac{1}{1 + \exp(-x)}.$$

It maps real-valued scores to probabilities, is monotone increasing, and satisfies $\text{logit}^{-1}(0) = 0.5$. The implementation uses `stats::plogis()`, which evaluates the same transformation with stable numerical handling for large positive or negative inputs. The implementation accepts finite numeric inputs only; infinite values are rejected even though the mathematical limits of the logistic function are defined. The returned vector has the same length as `x`.

Value

A numeric vector of probabilities with the same length as `x`.

Examples

```
scores <- data.frame(logit_score = c(-2, -1, 0, 1, 2)) |>
  dplyr::mutate(probability = inv_logit(logit_score))

scores
```

---

logit *Logit transformation*

Description

`logit()` maps probabilities from `(0, 1)` to the real line. Inputs must lie in `[0, 1]`; values outside this probability interval are rejected. Valid probabilities below `eps` and above `1 - eps` are clipped before the transformation, because the mathematical logit is infinite at the boundary.

Usage

```
logit(p, eps = .Machine$double.eps)
```

Arguments

p	Numeric vector of probabilities in `[0, 1]`.
eps	Positive clipping constant in `(0, 0.5)` used before applying the logit.

Details

For a probability $p \in (0, 1)$, the logit is

$$\text{logit}(p) = \log\left(\frac{p}{1-p}\right).$$

The transformation is monotone increasing and maps probabilities below 0.5 to negative values, 0.5 to zero, and probabilities above 0.5 to positive values. Because the expression is not finite at $p = 0$ or $p = 1$, the implementation first computes

$$p^* = \min\{\max(p, \epsilon), 1 - \epsilon\},$$

where ϵ is `eps`, and then returns $\text{logit}(p^*)$. The returned vector has the same length as `p`.

Value

A numeric vector on the logit scale with the same length as `p`.

Examples

```
probabilities <- data.frame(p = c(0.05, 0.25, 0.5, 0.75, 0.95)) |>
  dplyr::mutate(
    logit_p = logit(p),
    recovered = inv_logit(logit_p)
  )

probabilities
```

---

mce *Maximum Calibration Error*

Description

`mce()` returns the largest empirical absolute gap between mean confidence and empirical event frequency among non-empty equal-width bins. For multiclass inputs the `"classwise"` form returns the largest binary MCE across the one-vs-rest columns and the `"confidence"` form uses the top-label confidence.

Usage

```
mce(p, y, bins = 10, type = c("classwise", "confidence"))
```

Arguments

p	Predicted probabilities. A numeric vector in `[0, 1]` for binary problems, or a numeric matrix with one column per class for multiclass problems. Matrix inputs must have finite entries in `[0, 1]`, at least two columns, and rows summing to one within absolute tolerance `1e-6`.
y	Outcome labels. A vector coded as `0` and `1` for binary problems, or a factor or vector of integer class codes in `1:K` for multiclass problems.
bins	Number of bins on `[0, 1]`. Must be a single positive integer.
type	Multiclass aggregation, either `"classwise"` or `"confidence"`. Ignored for binary inputs.

Details

Using the same bin notation and endpoint convention as `ece()`, the binary empirical maximum calibration error is

$$\mathrm{MCE} = \max_{b:n_b>0} |\,\mathrm{acc}(b) - \mathrm{conf}(b)|.$$

Empty bins are ignored. For a multiclass probability matrix, `type = "classwise"` returns the maximum of the one-vs-rest binary MCE values across classes,

$$\mathrm{MCE}_{\mathrm{cw}} = \max_{1\le k\le K} \mathrm{MCE}(p_{\cdot k}, \mathbf{1}\{y_i = k\}).$$

`type = "confidence"` returns $\mathrm{MCE}(r, c)$ using the top-label confidence and correctness variables defined in `ece()`.

Value

A single numeric value.

References

Guo, C., Pleiss, G., Sun, Y., & Weinberger, K. Q. (2017). On calibration of modern neural networks. Proceedings of the 34th International Conference on Machine Learning.

Examples

```
predictions <- data.frame(
  p = c(0.10, 0.20, 0.80, 0.90),
  y = c(0, 0, 1, 1)
)

predictions |>
  dplyr::summarise(mce = mce(p, y, bins = 2))
```

mmce *Maximum Mean Calibration Error*

Description

`mmce()` is a binning-free empirical calibration statistic built from a kernel mean embedding of the calibration error. Unlike `ece()`, it does not partition the probability space into bins, so it avoids sensitivity to the number and placement of bins. It still depends on the kernel and bandwidth. The returned value is an empirical kernel statistic, not a population calibration parameter by itself.

Usage

```
mmce(p, y, bandwidth = 0.2)
```

Arguments

`p`	Predicted probabilities. A numeric vector in `[0, 1]` for binary problems, or a numeric matrix with one column per class for multiclass problems. Matrix inputs must have finite entries in `[0, 1]`, at least two columns, and rows summing to one within absolute tolerance `1e-6`.
`y`	Outcome labels. A vector coded as `0` and `1` for binary problems, or a factor or vector of integer class codes in `1:K` for multiclass problems.
`bandwidth`	Positive finite scalar bandwidth of the Laplacian kernel.

Details

For a binary input the residual compares the event indicator `y` with the predicted event probability `p`. For a multiclass probability matrix the confidence is the top-label probability and correctness indicates whether the predicted class is right. For multiclass inputs, `mmce()` implements only this top-label confidence form; there is no classwise `type` argument. The statistic uses a Laplacian kernel $k(a, b) = \exp(-|a - b|/\text{bandwidth})$. The computation builds an observation by observation kernel matrix, so both time and memory scale as $O(n^2)$.

Let r_i be the scalar probability assigned to observation i and c_i the corresponding binary target. In the binary case, $r_i = p_i$ and $c_i = y_i$. In the multiclass case, ties are broken by the first class, $\hat{y}_i = \min\{k : p_{ik} = \max_\ell p_{i\ell}\}$, $r_i = p_{i\hat{y}_i}$, and $c_i = \mathbf{1}\{\hat{y}_i = y_i\}$. The residual used by the statistic is

$$e_i = c_i - r_i.$$

With the Laplacian kernel

$$k(r_i, r_j) = \exp\left(-\frac{|r_i - r_j|}{h}\right),$$

where h is `bandwidth`, the returned value is the V-statistic plug-in estimate with diagonal terms,

$$\mathrm{MMCE} = \left\{ \frac{1}{n^2} \sum_{i=1}^{n} \sum_{j=1}^{n} e_i e_j k(r_i, r_j) \right\}^{1/2}.$$

The square-root argument is truncated at zero after numerical computation to avoid negative values caused only by floating-point error, so the returned value is nonnegative.

This statistic is the biased plug-in estimator of the squared kernel calibration error of Widmann, Lindsten and Zachariah (2019). In the binary and top-label confidence cases `mmce(p, y, bandwidth)` equals `sqrt(skce(p, y, estimator = "biased", bandwidth))`. The unbiased estimators offered by `skce()` remove the upward bias contributed by the diagonal terms.

Value

A single numeric value.

References

Kumar, A., Sarawagi, S., & Jain, U. (2018). Trainable calibration measures for neural networks from kernel mean embeddings. Proceedings of the 35th International Conference on Machine Learning.

Widmann, D., Lindsten, F., & Zachariah, D. (2019). Calibration tests in multi-class classification: A unifying framework. Advances in Neural Information Processing Systems 32. arXiv:1910.11385.

See Also

`skce()`, `cal_test()`, `cal_ci()`

Examples

```
set.seed(31)
p <- stats::runif(200)
y <- rbinom(200, 1, p)
mmce(p, y)
```

---

reliability_diagram *Reliability diagram*

Description

`reliability_diagram()` returns a `ggplot2` object comparing mean predicted confidence with the observed event frequency in equal-width probability bins. By default, points are sized by the number of observations in each non-empty bin and the subtitle reports the ECE computed with the same bins.

Usage

```
reliability_diagram(
  p,
  y,
  bins = 10,
  show_ece = TRUE,
  show_counts = TRUE,
  type = c("classwise", "confidence")
)
```

Arguments

p	Predicted probabilities. A numeric vector in `[0, 1]` for binary problems, or a numeric matrix with one column per class for multiclass problems. Matrix inputs must have finite entries in `[0, 1]`, at least two columns, and rows summing to one within absolute tolerance `1e-6`.
y	Outcome labels. A vector coded as `0` and `1` for binary problems, or a factor or vector of integer class codes in `1:K` for multiclass problems.
bins	Number of equal-width bins on `[0, 1]`. Must be a single positive integer.
show_ece	Logical. If `TRUE`, include the ECE in the plot subtitle.
show_counts	Logical. If `TRUE`, map point size to the number of observations in each bin.
type	Multiclass layout, either `"classwise"` or `"confidence"`. Ignored for binary inputs.

Details

For a probability matrix the function builds a multiclass diagram. The `"classwise"` form draws one panel per class from the one-vs-rest view. The `"confidence"` form draws a single panel from the top-label confidence and whether the predicted class is correct.

The diagram is a visual version of the binned summaries used by `ece()`. For binary inputs, the package uses the same left-closed equal-width bins as `ece()`, with the last bin closed on the right. For each non-empty bin b, the x-coordinate is the mean predicted probability,

$$\mathrm{conf}(b) = \frac{1}{n_b} \sum_{i \in I_b} p_i,$$

and the y-coordinate is the observed event frequency,

$$\mathrm{acc}(b) = \frac{1}{n_b} \sum_{i \in I_b} y_i.$$

Points near the diagonal line have similar average confidence and empirical frequency within the bin. Points below the diagonal indicate over-confident predictions in that bin, and points above the diagonal indicate under-confident predictions. Empty bins are omitted from the plotted data. The diagonal reference line is the set where the bin mean predicted probability equals the empirical event frequency.

For multiclass inputs, `type = "classwise"` builds these summaries separately for each one-vs-rest class and displays them in facets. `type = "confidence"` replaces p_i by the top-label probability and y_i by the indicator that the top-label prediction is correct. Ties in the top-label rule are broken by the first column, matching `max.col(..., ties.method = "first")`. When `show_ece = TRUE`, the subtitle reports `ece(p, y, bins = bins)` for binary inputs and `ece(p, y, bins = bins, type = type)` for multiclass inputs.

Value

A `ggplot` object.

References

Niculescu-Mizil, A., & Caruana, R. (2005). Predicting good probabilities with supervised learning. Proceedings of the 22nd International Conference on Machine Learning.

Examples

```
set.seed(6)
predictions <- data.frame(raw_p = stats::runif(120)) |>
  dplyr::mutate(y = rbinom(dplyr::n(), 1, raw_p))

reliability_diagram(predictions$raw_p, predictions$y, bins = 8)

# Multiclass reliability diagram with one panel per class.
set.seed(60)
prob <- matrix(stats::runif(150 * 3), ncol = 3)
prob <- prob / rowSums(prob)
labels <- max.col(prob)
reliability_diagram(prob, labels, bins = 8, type = "classwise")
```

---

skce *Squared Kernel Calibration Error*

Description

`skce()` returns an estimate of the squared kernel calibration error (SKCE) of Widmann, Lindsten and Zachariah (2019). The SKCE is a binning-free, kernel-based measure of calibration error built from a residual and a positive-definite kernel, generalising the maximum mean calibration error of `mmce()`. Three estimators are available: the biased plug-in V-statistic that keeps the diagonal terms, and two unbiased estimators that remove the diagonal-induced upward bias.

Usage

```
skce(
  p,
  y,
  estimator = c("uq", "ul", "biased"),
  bandwidth = 0.2,
```

```
  type = c("canonical", "confidence", "classwise")
)
```

Arguments

p	Predicted probabilities. A numeric vector in `[0, 1]` for binary problems, or a numeric matrix with one column per class for multiclass problems. Matrix inputs must have finite entries in `[0, 1]`, at least two columns, and rows summing to one within absolute tolerance `1e-6`.
y	Outcome labels. A vector coded as `0` and `1` for binary problems, or a factor or vector of integer class codes in `1:K` for multiclass problems.
estimator	Which estimator to return: `"uq"` for the unbiased $O(n^2)$ U-statistic (the default), `"ul"` for the unbiased linear-time estimator, or `"biased"` for the biased V-statistic that matches `mmce()^2`.
bandwidth	Bandwidth of the Laplacian kernel. Either a single positive number (the default `0.2`) or the string `"median"`, which sets the bandwidth to the median of the positive pairwise distances (the median heuristic of Widmann et al. 2019, Section 7). The median heuristic is recommended for the canonical multiclass form, where the scale of the simplex distances depends on the number of classes; the fixed `0.2` is a reproducible default for the binary and confidence forms, whose confidences lie in `[0, 1]`.
type	Multiclass calibration target, one of `"canonical"` (strong calibration of the full probability vector, the default for matrix inputs), `"confidence"` (top-label confidence), or `"classwise"` (mean over one-vs-rest columns). Ignored for binary vector inputs.

Details

For a binary input the residual compares the event indicator `y` with the predicted event probability `p`, and the kernel is the scalar Laplacian kernel on `p`. For a multiclass probability matrix, `type` selects the calibration target. The `"confidence"` form reduces to the top-label probability and whether the predicted class is correct, exactly as in `mmce()`. The `"canonical"` (strong) form uses the full probability vector with a matrix-valued kernel and characterises calibration of the entire forecast distribution. The `"classwise"` form averages the binary SKCE over the one-vs-rest columns.

In the binary or confidence case `skce(p, y, estimator = "biased")` equals `mmce(p, y, bandwidth)^2`; that is, `mmce()` is the square root of the biased SKCE estimator.

Let e_i be the residual for observation i and k the kernel. In the confidence and binary cases the residual is the scalar $e_i = c_i - \rho_i$, where ρ_i is the scalar confidence (the predicted event probability for binary inputs, or the top-label probability for the confidence reduction) and c_i the corresponding binary target. The kernel is the scalar Laplacian kernel $k(\rho_i, \rho_j) = \exp(-|\rho_i - \rho_j|/h)$ with bandwidth h, and the kernel term is $h_{ij} = e_i e_j k(\rho_i, \rho_j)$.

In the canonical multiclass case the residual is the vector $e_i = \mathbf{1}_{y_i} - p_i$ in $\mathbb{R}^K$, where $\mathbf{1}_{y_i}$ is the one-hot encoding of the label. The matrix-valued kernel is $k(s,t) = \tilde{k}(s,t) I_K$ with $\tilde{k}$ the Laplacian kernel $\tilde{k}(s,t) = \exp(-\|s-t\|_2/h)$ on the probability vectors, using the Euclidean norm on the simplex. The kernel term is then $h_{ij} = \tilde{k}(p_i, p_j)\,\langle e_i, e_j \rangle$.

Writing $H = [h_{ij}]$, the three estimators of Widmann et al. (2019, Table 1) are

$$\widehat{\mathrm{SKCE}}_b = \frac{1}{n^2}\sum_{i,j} h_{ij},$$

$$\widehat{\mathrm{SKCE}}_{uq} = \binom{n}{2}^{-1}\sum_{i<j} h_{ij},$$

$$\widehat{\mathrm{SKCE}}_{ul} = \lfloor n/2 \rfloor^{-1}\sum_{i=1}^{\lfloor n/2 \rfloor} h_{2i-1,\,2i}.$$

The biased estimator `"biased"` keeps the diagonal terms $i = j$ and is the V-statistic underlying `mmce()`; the diagonal contributes $n^{-2}\sum_i \|e_i\|^2 > 0$, so it is biased upward and need not vanish even under perfect finite-sample calibration. The unbiased estimator `"uq"` drops the diagonal and divides by $n(n-1)$; it costs $O(n^2)$. The unbiased estimator `"ul"` sums over the disjoint pairs $(1,2),(3,4),\ldots$ and costs $O(n)$. The unbiased estimators can be negative in finite samples because they are unbiased estimates of a nonnegative quantity; only the biased estimator is guaranteed nonnegative.

Value

A single numeric value, the squared kernel calibration error estimate. For `estimator = "biased"` the value is nonnegative; the unbiased estimators may be negative in finite samples.

References

Widmann, D., Lindsten, F., & Zachariah, D. (2019). Calibration tests in multi-class classification: A unifying framework. Advances in Neural Information Processing Systems 32. arXiv:1910.11385.

Kumar, A., Sarawagi, S., & Jain, U. (2018). Trainable calibration measures for neural networks from kernel mean embeddings. Proceedings of the 35th International Conference on Machine Learning.

See Also

`mmce()`, `cal_test()`, `cal_ci()`

Examples

```
set.seed(31)
p <- stats::runif(200)
y <- rbinom(200, 1, p)

# Unbiased U-statistic estimate of the squared kernel calibration error.
skce(p, y)

# The biased estimator equals mmce()^2.
all.equal(skce(p, y, estimator = "biased"), mmce(p, y)^2)

# Canonical (strong) multiclass calibration from a probability matrix.
set.seed(32)
```

```
prob <- matrix(stats::runif(150 * 3), ncol = 3)
prob <- prob / rowSums(prob)
labels <- max.col(prob)
skce(prob, labels, type = "canonical")
```

Index