Package 'loo'

April 11, 2018

Type Package

Title Efficient Leave-One-Out Cross-Validation and WAIC for Bayesian Models

Version 2.0.0

Date 2018-04-06

Maintainer Jonah Gabry <jsg2201@columbia.edu>

URL http://mc-stan.org, http://discourse.mc-stan.org

BugReports https://github.com/stan-dev/loo/issues

Description Efficient approximate leave-one-out cross-validation (LOO) for Bayesian models fit using Markov chain Monte Carlo. The approximation uses Pareto smoothed importance sampling (PSIS), a new procedure for regularizing importance weights. As a byproduct of the calculations, we also obtain approximate standard errors for estimated predictive errors and for the comparison of predictive errors between models. The package also provides methods for using stacking and other model weighting techniques to average Bayesian predictive distributions.

License GPL (>= 3)

LazyData TRUE

Depends R (>= 3.1.2)

Imports graphics, matrixStats (>= 0.52), parallel, stats

Suggests bayesplot (>= 1.5.0), knitr, rmarkdown, rstan, rstanarm, rstantools, testthat

VignetteBuilder knitr

Encoding UTF-8

RoxygenNote 6.0.1

NeedsCompilation no

Author Aki Vehtari [aut],
Andrew Gelman [aut],
Jonah Gabry [cre, aut],
Yuling Yao [aut],

Juho Piironen [ctb],
Ben Goodrich [ctb]

Repository CRAN

Date/Publication 2018-04-11 12:34:58 UTC

R topics documented:

`loo-package` *Efficient LOO-CV and WAIC for Bayesian models*

Description

This package implements the methods described in Vehtari, Gelman, and Gabry (2017a, 2017b) and Yao et al. (2018). To get started see the `loo` function for efficient approximate leave-one-out cross-validation (LOO-CV), the `psis` function for the Pareto smoothed importance sampling (PSIS) algorithm, or `loo_model_weights` for an implementation of Bayesian stacking of predictive distributions from multiple models.

Details

Leave-one-out cross-validation (LOO-CV) and the widely applicable information criterion (WAIC) are methods for estimating pointwise out-of-sample prediction accuracy from a fitted Bayesian model using the log-likelihood evaluated at the posterior simulations of the parameter values. LOO-CV and WAIC have various advantages over simpler estimates of predictive error such as AIC and DIC but are less used in practice because they involve additional computational steps. This package implements the fast and stable computations for approximate LOO-CV laid out in Vehtari, Gelman, and Gabry (2017a). From existing posterior simulation draws, we compute LOO-CV using Pareto smoothed importance sampling (PSIS; Vehtari, Gelman, and Gabry, 2017b), a new procedure for regularizing and diagnosing importance weights. As a byproduct of our calculations, we also obtain approximate standard errors for estimated predictive errors and for comparing of predictive errors between two models.

We recommend PSIS-LOO-CV instead of WAIC, because PSIS provides useful diagnostics and effective sample size and Monte Carlo standard error estimates.

References

Vehtari, A., Gelman, A., and Gabry, J. (2017a). Practical Bayesian model evaluation using leave-one-out cross-validation and WAIC. *Statistics and Computing*. 27(5), 1413–1432. doi:10.1007/s11222-016-9696-4. (published version, arXiv preprint).

Vehtari, A., Gelman, A., and Gabry, J. (2017b). Pareto smoothed importance sampling. arXiv preprint: `http://arxiv.org/abs/1507.02646/`

Yao, Y., Vehtari, A., Simpson, D., and Gelman, A. (2018) Using stacking to average Bayesian predictive distributions. *Bayesian Analysis*, advance publication, doi:10.1214/17-BA1091. (online).

Epifani, I., MacEachern, S. N., and Peruggia, M. (2008). Case-deletion importance sampling estimators: Central limit theorems and related results. *Electronic Journal of Statistics* **2**, 774-806.

Gelfand, A. E. (1996). Model determination using sampling-based methods. In *Markov Chain Monte Carlo in Practice*, ed. W. R. Gilks, S. Richardson, D. J. Spiegelhalter, 145-162. London: Chapman and Hall.

Gelfand, A. E., Dey, D. K., and Chang, H. (1992). Model determination using predictive distributions with implementation via sampling-based methods. In *Bayesian Statistics 4*, ed. J. M. Bernardo, J. O. Berger, A. P. Dawid, and A. F. M. Smith, 147-167. Oxford University Press.

Gelman, A., Hwang, J., and Vehtari, A. (2014). Understanding predictive information criteria for Bayesian models. *Statistics and Computing* **24**, 997-1016.

Ionides, E. L. (2008). Truncated importance sampling. *Journal of Computational and Graphical Statistics* **17**, 295-311.

Koopman, S. J., Shephard, N., and Creal, D. (2009). Testing the assumptions behind importance sampling. *Journal of Econometrics* **149**, 2-11.

Peruggia, M. (1997). On the variability of case-deletion importance sampling weights in the Bayesian linear model. *Journal of the American Statistical Association* **92**, 199-207.

Stan Development Team (2017). The Stan C++ Library, Version 2.17.0. `http://mc-stan.org`.

Stan Development Team (2018). RStan: the R interface to Stan, Version 2.17.3. `http://mc-stan.org`.

Watanabe, S. (2010). Asymptotic equivalence of Bayes cross validation and widely application information criterion in singular learning theory. *Journal of Machine Learning Research* **11**, 3571-3594.

Zhang, J., and Stephens, M. A. (2009). A new and efficient estimation method for the generalized Pareto distribution. *Technometrics* **51**, 316-325.

`compare` *Model comparison*

Description

Compare fitted models on LOO or WAIC.

Usage

```
compare(..., x = list())

## S3 method for class 'compare.loo'
print(x, ..., digits = 1)
```

Arguments

`...`	At least two objects returned by `loo` (or `waic`).
`x`	A list of at least two objects returned by `loo` (or `waic`). This argument can be used as an alternative to specifying the objects in `...`.
`digits`	For the print method only, the number of digits to use when printing.

Details

When comparing two fitted models, we can estimate the difference in their expected predictive accuracy by the difference in `elpd_loo` or `elpd_waic` (multiplied by -2, if desired, to be on the deviance scale). To compute the standard error of this difference we can use a paired estimate to take advantage of the fact that the same set of N data points was used to fit both models. These calculations should be most useful when N is large, because then non-normality of the distribution is not such an issue when estimating the uncertainty in these sums. These standard errors, for all their flaws, should give a better sense of uncertainty than what is obtained using the current standard approach of comparing differences of deviances to a Chi-squared distribution, a practice derived for Gaussian linear models or asymptotically, and which only applies to nested models in any case.

Value

A vector or matrix with class `'compare.loo'` that has its own print method. If exactly two objects are provided in `...` or `x`, then the difference in expected predictive accuracy and the standard error of the difference are returned (see Details). *The difference will be positive if the expected predictive accuracy for the second model is higher.* If more than two objects are provided then a matrix of summary information is returned.

References

Vehtari, A., Gelman, A., and Gabry, J. (2017a). Practical Bayesian model evaluation using leave-one-out cross-validation and WAIC. *Statistics and Computing*. 27(5), 1413–1432. doi:10.1007/s11222-016-9696-4. (published version, arXiv preprint).

Vehtari, A., Gelman, A., and Gabry, J. (2017b). Pareto smoothed importance sampling. arXiv preprint: `http://arxiv.org/abs/1507.02646/`

Examples

```
## Not run:
loo1 <- loo(log_lik1)
loo2 <- loo(log_lik2)
print(compare(loo1, loo2), digits = 3)
print(compare(x = list(loo1, loo2)))

waic1 <- waic(log_lik1)
waic2 <- waic(log_lik2)
compare(waic1, waic2)

## End(Not run)
```

`example_loglik_array` *Objects to use in examples and tests*

Description

Example pointwise log-likelihood objects to use in demonstrations and tests. See the **Value** and **Examples** sections below.

Usage

```
example_loglik_array()

example_loglik_matrix()
```

Value

`example_loglik_array` returns a 500 (draws) x 2 (chains) x 32 (observations) pointwise log-likelihood array.

`example_loglik_matrix` returns the same pointwise log-likelihood values as `example_loglik_array` but reshaped into a 1000 (draws*chains) x 32 (observations) matrix.

Examples

```
LLarr <- example_loglik_array()
(dim_arr <- dim(LLarr))
LLmat <- example_loglik_matrix()
(dim_mat <- dim(LLmat))

all.equal(dim_mat[1], dim_arr[1] * dim_arr[2])
all.equal(dim_mat[2], dim_arr[3])

all.equal(LLarr[, 1, ], LLmat[1:500, ])
all.equal(LLarr[, 2, ], LLmat[501:1000, ])
```

extract_log_lik *Extract pointwise log-likelihood from a Stan model*

Description

Convenience function for extracting the pointwise log-likelihood matrix or array from a fitted Stan model.

Usage

```
extract_log_lik(stanfit, parameter_name = "log_lik", merge_chains = TRUE)
```

Arguments

`stanfit`	A `stanfit` object (**rstan** package).
`parameter_name`	A character string naming the parameter (or generated quantity) in the Stan model corresponding to the log-likelihood.
`merge_chains`	If `TRUE` (the default), all Markov chains are merged together (i.e., stacked) and a matrix is returned. If `FALSE` they are kept separate and an array is returned.

Details

Stan does not automatically compute and store the log-likelihood. It is up to the user to incorporate it into the Stan program if it is to be extracted after fitting the model. In a Stan model, the pointwise log likelihood can be coded as a vector in the transformed parameters block (and then summed up in the model block) or it can be coded entirely in the generated quantities block. We recommend using the generated quantities block so that the computations are carried out only once per iteration rather than once per HMC leapfrog step.

For example, the following is the `generated quantities` block for computing and saving the log-likelihood for a linear regression model with `N` data points, outcome `y`, predictor matrix `X`, coefficients `beta`, and standard deviation `sigma`:

```
vector[N] log_lik;
for (n in 1:N) log_lik[n] = normal_lpdf(y[n] | X[n, ] * beta, sigma);
```

Value

If `merge_chains=TRUE`, an S by N matrix of (post-warmup) extracted draws, where S is the size of the posterior sample and N is the number of data points. If `merge_chains=FALSE`, an I by C by N array, where $I \times C = S$.

References

Stan Development Team (2017). The Stan C++ Library, Version 2.16.0. http://mc-stan.org/

Stan Development Team (2017). RStan: the R interface to Stan, Version 2.16.1. http://mc-stan.org/

E_loo *Compute weighted expectations*

Description

The E_loo function computes weighted expectations (means, variances, quantiles) using the importance weights obtained from the PSIS smoothing procedure. The expectations estimated by the E_loo function assume that the PSIS approximation is working well. **A small Pareto k estimate is necessary, but not sufficient, for E_loo to give reliable estimates.** Additional diagnostic checks for gauging the reliability of the estimates are in development and will be added in a future release.

Usage

```
E_loo(x, psis_object, ...)

## Default S3 method:
E_loo(x, psis_object, ..., type = c("mean", "variance",
  "quantile"), probs = NULL, log_ratios = NULL)

## S3 method for class 'matrix'
E_loo(x, psis_object, ..., type = c("mean", "variance",
  "quantile"), probs = NULL, log_ratios = NULL)
```

Arguments

x	A numeric vector or matrix.
psis_object	An object returned by psis.
...	Arguments passed to individual methods.
type	The type of expectation to compute. The options are "mean", "variance", and "quantile".
probs	For computing quantiles, a vector of probabilities.
log_ratios	Optionally, a vector or matrix (the same dimensions as x) of raw (not smoothed) log ratios. If working with log-likelihood values, the log ratios are the **negative** of those values. If log_ratios is specified we are able to compute Pareto k diagnostics specific to E_loo.

Value

A named list with the following components:

value The result of the computation.

For the matrix method, value is a vector with ncol(x) elements, with one exception: when type is "quantile" and multiple values are specified in probs the value component of the returned object is a length(probs) by ncol(x) matrix.

For the default/vector method the value component is scalar, with one exception: when type is "quantile" and multiple values are specified in probs the value component is a vector with length(probs) elements.

`pareto_k` Function-specific diagnostic.

If `log_ratios` is not specified when calling `E_loo`, `pareto_k` will be `NULL`. Otherwise, for the matrix method it will be a vector of length `ncol(x)` containing estimates of the shape parameter k of the generalized Pareto distribution. For the default/vector method, the estimate is a scalar.

Examples

```
# Use rstanarm package to quickly fit a model and get both a log-likelihood
# matrix and draws from the posterior predictive distribution
library("rstanarm")

# data from help("lm")
ctl <- c(4.17,5.58,5.18,6.11,4.50,4.61,5.17,4.53,5.33,5.14)
trt <- c(4.81,4.17,4.41,3.59,5.87,3.83,6.03,4.89,4.32,4.69)
d <- data.frame(
  weight = c(ctl, trt),
  group = gl(2, 10, 20, labels = c("Ctl","Trt"))
)
fit <- stan_glm(weight ~ group, data = d)
yrep <- posterior_predict(fit)
dim(yrep)

ll <- log_lik(fit)
dim(ll)

r_eff <- relative_eff(exp(-ll), chain_id = rep(1:4, each = 1000))
psis_object <- psis(log_ratios = -ll, r_eff = r_eff, cores = 2)

E_loo(yrep, psis_object, type = "mean")
E_loo(yrep, psis_object, type = "var")
E_loo(yrep, psis_object, type = "quantile", probs = 0.5) # median
E_loo(yrep, psis_object, type = "quantile", probs = c(0.1, 0.9))

# To get Pareto k diagnostic with E_loo we also need to provide the negative
# log-likelihood values using the log_ratios argument.
E_loo(yrep, psis_object, type = "mean", log_ratios = -ll)
```

`gpdfit` *Estimate parameters of the Generalized Pareto distribution*

Description

Estimate the parameters k and σ of the generalized Pareto distribution (assuming location parameter is 0), given a sample x. By default the Pareto fit uses a prior for k, which will stabilize estimates for very small sample sizes and low effective sample sizes in the case of MCMC samples. The weakly informative prior is a Gaussian prior centered at 0.5.

Usage

```
gpdfit(x, wip = TRUE, min_grid_pts = 30, sort_x = TRUE)
```

Arguments

`x`	A numeric vector. The sample from which to estimate the parameters.
`wip`	Logical indicating whether to adjust k based on a weakly informative Gaussian prior centered on 0.5. Defaults to `TRUE`.
`min_grid_pts`	The minimum number of grid points used in the fitting algorithm. The actual number used is `min_grid_pts + floor(sqrt(length(x)))`.
`sort_x`	If `TRUE` (the default), the first step in the fitting algorithm is to sort the elements of `x`. If `x` is already sorted in ascending order then `sort_x` can be set to `FALSE` to skip the initial sorting step.

Details

Here the parameter k is the negative of k in Zhang & Stephens (2009).

Value

A named list with components `k` and `sigma`.

References

Zhang, J., and Stephens, M. A. (2009). A new and efficient estimation method for the generalized Pareto distribution. *Technometrics* **51**, 316-325.

See Also

`psis`, `pareto-k-diagnostic`, `loo-package`

`kfold-helpers` *Helper functions for K-fold cross-validation*

Description

These functions can be used to generate indexes for use with K-fold cross-validation.

Usage

```
kfold_split_random(K = 10, N = NULL)

kfold_split_balanced(K = 10, x = NULL)

kfold_split_stratified(K = 10, x = NULL)
```

Arguments

K	The number of folds to use.
N	The number of observations in the data.
x	A discrete variable of length N. Will be coerced to factor. For kfold_split_balanced x should be a binary variable. For kfold_split_stratified x should be a grouping variable with at least K levels.

Details

kfold_split_random splits the data into K groups of equal size (or roughly equal size).

For a binary variable x that has many more 0s than 1s (or vice-versa) kfold_split_balanced first splits the data by value of x, does kfold_split_random within each of the two groups, and then recombines the indexes returned from the two calls to kfold_split_random. This helps ensure that the observations in the less common category of x are more evenly represented across the folds.

For a grouping variable x, kfold_split_stratified places all observations in x from the same group/level together the same fold. The selection of which groups/levels go into which fold (relevant when when there are more folds than groups) is randomized.

Value

An integer vector of length N where each element is an index in 1:K.

Examples

```
kfold_split_random(K = 5, N = 20)

x <- sample(c(0, 1), size = 200, replace = TRUE, prob = c(0.05, 0.95))
table(x)
ids <- kfold_split_balanced(K = 5, x = x)
table(ids[x == 0])
table(ids[x == 1])

grp <- gl(n = 50, k = 15, labels = state.name)
length(grp)
head(table(grp))

ids_10 <- kfold_split_stratified(K = 10, x = grp)
(tab_10 <- table(grp, ids_10))
print(colSums(tab_10))
all.equal(sum(colSums(tab_10)), length(grp))

ids_9 <- kfold_split_stratified(K = 9, x = grp)
tab_9 <- table(grp, ids_9)
print(colSums(tab_9))
all.equal(sum(colSums(tab_10)), length(grp))
```

loo — *Efficient approximate leave-one-out cross-validation (LOO)*

Description

The loo methods for arrays, matrices, and functions compute PSIS-LOO CV, efficient approximate leave-one-out (LOO) cross-validation for Bayesian models using Pareto smoothed importance sampling (PSIS). This is an implementation of the methods described in Vehtari, Gelman, and Gabry (2017a, 2017b).

The loo_i function enables testing log-likelihood functions for use with the loo.function method.

Usage

```
loo(x, ...)

## S3 method for class 'array'
loo(x, ..., r_eff = NULL, save_psis = FALSE,
  cores = getOption("mc.cores", 1))

## S3 method for class 'matrix'
loo(x, ..., r_eff = NULL, save_psis = FALSE,
  cores = getOption("mc.cores", 1))

## S3 method for class 'function'
loo(x, ..., data = NULL, draws = NULL, r_eff = NULL,
  save_psis = FALSE, cores = getOption("mc.cores", 1))

loo_i(i, llfun, ..., data = NULL, draws = NULL, r_eff = NULL)
```

Arguments

x	A log-likelihood array, matrix, or function. See the **Methods (by class)** section below for a detailed description of how to specify the inputs for each method.
r_eff	Vector of relative effective sample size estimates for the likelihood (exp(log_lik)) of each observation. This is related to the relative efficiency of estimating the normalizing term in self-normalizing importance sampling. If r_eff is not provided then the reported PSIS effective sample sizes and Monte Carlo error estimates will be over-optimistic. See the relative_eff helper function for computing r_eff.
save_psis	Should the "psis" object created internally by loo be saved in the returned object? The loo function calls psis internally but by default discards the (potentially large) "psis" object after using it to compute the LOO-CV summaries. Setting save_psis to TRUE will add a psis_object component to the list returned by loo. Currently this is only needed if you plan to use the E_loo function to compute weighted expectations after running loo.

`cores`	The number of cores to use for parallelization. This defaults to the option `mc.cores` which can be set for an entire R session by `options(mc.cores = NUMBER)`. The old option `loo.cores` is now deprecated but will be given precedence over `mc.cores` until `loo.cores` is removed in a future release. **As of version 2.0.0 the default is now 1 core if `mc.cores` is not set, but we recommend using as many (or close to as many) cores as possible.**
`data, draws, ...`	For the `loo` function method and the `loo_i` function, the data, posterior draws, and other arguments to pass to the log-likelihood function. See the **Methods (by class)** section below for details on how to specify these arguments.
`i`	For `loo_i`, an integer in `1:N`.
`llfun`	For `loo_i`, the same as `x` for the `loo.function` method. A log-likelihood function as described in the **Methods (by class)** section.

Details

The `loo` function is an S3 generic and methods are provided for computing LOO from 3-D pointwise log-likelihood arrays, pointwise log-likelihood matrices, and log-likelihood functions. The array and matrix methods are most convenient, but for models fit to very large datasets the `loo.function` method is more memory efficient and may be preferable.

Value

The `loo` methods return a named list with class `c("psis_loo", "loo")` and components:

`estimates` A matrix with two columns (`Estimate`, `SE`) and four rows (`elpd_loo`, `mcse_elpd_loo`, `p_loo`, `looic`). This contains point estimates and standard errors of the expected log pointwise predictive density (`elpd_loo`), the Monte Carlo standard error of `elpd_loo` (`mcse_elpd_loo`), the effective number of parameters (`p_loo`) and the LOO information criterion `looic` (which is just `-2 * elpd_loo`, i.e., converted to deviance scale).

`pointwise` A matrix with four columns (and number of rows equal to the number of observations) containing the pointwise contributions of each of the above measures (`elpd_loo`, `mcse_elpd_loo`, `p_loo`, `looic`).

`diagnostics` A named list containing two vectors:

- `pareto_k`: Estimates of the shape parameter k of the generalized Pareto fit to the importance ratios for each leave-one-out distribution. See the `pareto-k-diagnostic` page for details.
- `n_eff`: PSIS effective sample size estimates.

`psis_object` This component will be `NULL` unless the `save_psis` argument is set to `TRUE` when calling `loo`. In that case `psis_object` will be the object of class `"psis"` that is created when the `loo` function calls `psis` internally to do the PSIS procedure.

The `loo_i` function returns a named list with components `pointwise` and `diagnostics`. These components have the same structure as the `pointwise` and `diagnostics` components of the object returned by `loo` except they contain results for only a single observation.

Methods (by class)

- `array`: An I by C by N array, where I is the number of MCMC iterations per chain, C is the number of chains, and N is the number of data points.
- `matrix`: An S by N matrix, where S is the size of the posterior sample (with all chains merged) and N is the number of data points.
- `function`: A function `f` that takes arguments `data_i` and `draws` and returns a vector containing the log-likelihood for a single observation `i` evaluated at each posterior draw. The function should be written such that, for each observation `i` in `1:N`, evaluating `f(data_i = data[i,, drop=FALSE], draws = c` results in a vector of length `S` (size of posterior sample). The log-likelihood function can also have additional arguments but `data_i` and `draws` are required.

  If using the function method then the arguments `data` and `draws` must also be specified in the call to `loo`:

  - `data`: A data frame or matrix containing the data (e.g. observed outcome and predictors) needed to compute the pointwise log-likelihood. For each observation `i`, the `i`th row of `data` will be passed to the `data_i` argument of the log-likelihood function.
  - `draws`: An object containing the posterior draws for any parameters needed to compute the pointwise log-likelihood. Unlike `data`, which is indexed by observation, for each observation the entire object `draws` will be passed to the `draws` argument of the log-likelihood function.
  - The `...` can be used to pass additional arguments to your log-likelihood function. These arguments are used like the `draws` argument in that they are recycled for each observation.

Defining `loo` methods in a package

Package developers can define `loo` methods for fitted models objects. See the example `loo.stanfit` method in the **Examples** section below for an example of defining a method that calls `loo.array`. The `loo.stanreg` method in **rstanarm** is an example of defining a method that calls `loo.function`.

References

Vehtari, A., Gelman, A., and Gabry, J. (2017a). Practical Bayesian model evaluation using leave-one-out cross-validation and WAIC. *Statistics and Computing*. 27(5), 1413–1432. doi:10.1007/s11222-016-9696-4. (published version, arXiv preprint).

Vehtari, A., Gelman, A., and Gabry, J. (2017b). Pareto smoothed importance sampling. arXiv preprint: `http://arxiv.org/abs/1507.02646/`

See Also

- The **loo** package vignettes for demonstrations.
- `psis` for the underlying Pareto Smoothed Importance Sampling (PSIS) procedure used in the LOO-CV approximation.
- pareto-k-diagnostic for convenience functions for looking at diagnostics.
- `compare` for model comparison.

Examples

```
### Array and matrix methods (using example objects included with loo package)
# Array method
LLarr <- example_loglik_array()
rel_n_eff <- relative_eff(exp(LLarr))
loo(LLarr, r_eff = rel_n_eff, cores = 2)

# Matrix method
LLmat <- example_loglik_matrix()
rel_n_eff <- relative_eff(exp(LLmat), chain_id = rep(1:2, each = 500))
loo(LLmat, r_eff = rel_n_eff, cores = 2)


## Not run:
### Usage with stanfit objects
# see ?extract_log_lik
log_lik1 <- extract_log_lik(stanfit1, merge_chains = FALSE)
rel_n_eff <- relative_eff(exp(log_lik1))
loo(log_lik1, r_eff = rel_n_eff, cores = 2)

## End(Not run)

### Using log-likelihood function instead of array or matrix
set.seed(124)

# Simulate data and draw from posterior
N <- 50; K <- 10; S <- 100; a0 <- 3; b0 <- 2
p <- rbeta(1, a0, b0)
y <- rbinom(N, size = K, prob = p)
a <- a0 + sum(y); b <- b0 + N * K - sum(y)
fake_posterior <- as.matrix(rbeta(S, a, b))
dim(fake_posterior) # S x 1
fake_data <- data.frame(y,K)
dim(fake_data) # N x 2

llfun <- function(data_i, draws) {
  # each time called internally within loo the arguments will be equal to:
  # data_i: ith row of fake_data (fake_data[i,, drop=FALSE])
  # draws: entire fake_posterior matrix
  dbinom(data_i$y, size = data_i$K, prob = draws, log = TRUE)
}

# Use the loo_i function to check that llfun works on a single observation
# before running on all obs. For example, using the 3rd obs in the data:
loo_3 <- loo_i(i = 3, llfun = llfun, data = fake_data, draws = fake_posterior)
print(loo_3$pointwise[, "elpd_loo"])

# Use loo.function method
loo_with_fn <- loo(llfun, draws = fake_posterior, data = fake_data)

# If we look at the elpd_loo contribution from the 3rd obs it should be the
# same as what we got above with the loo_i function and i=3:
print(loo_with_fn$pointwise[3, "elpd_loo"])
print(loo_3$pointwise[, "elpd_loo"])

# Check that the loo.matrix method gives same answer as loo.function method
log_lik_matrix <- sapply(1:N, function(i) {
  llfun(data_i = fake_data[i,, drop=FALSE], draws = fake_posterior)
})
loo_with_mat <- loo(log_lik_matrix)
all.equal(loo_with_mat$estimates, loo_with_fn$estimates) # should be TRUE!


## Not run:
### For package developers: defining loo methods

# An example of a possible loo method for 'stanfit' objects (rstan package).
# A similar method is planned for a future release of rstan (or is already
# released, depending on when you are reading this). In order for users
# to be able to call loo(stanfit) instead of loo.stanfit(stanfit) the
# NAMESPACE needs to be handled appropriately (roxygen2 and devtools packages
# are good for that).
#
loo.stanfit <-
 function(x,
         pars = "log_lik",
         ...,
         save_psis = FALSE,
         cores = getOption("mc.cores", 1)) {
  stopifnot(length(pars) == 1L)
  LLarray <- loo::extract_log_lik(stanfit = x,
                                  parameter_name = pars,
                                  merge_chains = FALSE)
  r_eff <- loo::relative_eff(x = exp(LLarray), cores = cores)
  loo::loo.array(LLarray,
                 r_eff = r_eff,
                 cores = cores,
                 save_psis = save_psis)
}

## End(Not run)
```

loo-datasets *Datasets for loo examples and vignettes*

Description

Small datasets for use in **loo** examples and vignettes. The `Kline` and `milk` datasets are also included in the **rethinking** package (McElreath, 2016a), but we include them here as **rethinking** is not yet on CRAN.

Format

`Kline` Small dataset from Kline and Boyd (2010) on tool complexity and demography in Oceanic islands societies. This data is discussed in detail in McElreath (2016a,2016b). (Link to variable descriptions)

`milk` Small dataset from Hinde and Milligan (2011) on primate milk composition.This data is discussed in detail in McElreath (2016a,2016b). (Link to variable descriptions)

References

Hinde and Milligan. 2011. Evolutionary Anthropology 20:9-23.

Kline, M.A. and R. Boyd. 2010. Proc R Soc B 277:2559–2564.

McElreath, R. (2016a). rethinking: Statistical Rethinking book package. R package version 1.59.

McElreath, R. (2016b). *Statistical rethinking: A Bayesian course with examples in R and Stan*. Chapman & Hall/CRC.

Examples

```
str(Kline)
str(milk)
```

`loo_model_weights` *Model averaging/weighting via stacking or pseudo-BMA weighting*

Description

Model averaging via stacking of predictive distributions, pseudo-BMA weighting or pseudo-BMA+ weighting with the Bayesian bootstrap. See Yao et al. (2018) and Vehtari, Gelman, and Gabry (2017a,2017b) for background.

Usage

```
loo_model_weights(x, ...)

## Default S3 method:
loo_model_weights(x, ..., method = c("stacking",
  "pseudobma"), optim_method = "BFGS", optim_control = list(), BB = TRUE,
  BB_n = 1000, alpha = 1, r_eff_list = NULL,
  cores = getOption("mc.cores", 1))

stacking_weights(lpd_point, optim_method = "BFGS", optim_control = list())

pseudobma_weights(lpd_point, BB = TRUE, BB_n = 1000, alpha = 1)
```

Arguments

`x`	A list of pointwise log-likelihood matrices or "psis_loo" objects (objects returned by `loo`), one for each model. Each matrix/object should have dimensions S by N, where S is the size of the posterior sample (with all chains merged) and N is the number of data points. If `x` is a list of log-likelihood matrices then `loo` is called internally on each matrix. Currently the `loo_model_weights` function is not implemented to be used with results from K-fold CV, but you can still obtain weights using K-fold CV results by calling the `stacking_weights` function directly.
`...`	Unused, except for the generic to pass arguments to individual methods.
`method`	Either `"stacking"` or `"pseudobma"`, indicating which method to use for obtaining the weights. `"stacking"` refers to stacking of predictive distributions and `"pseudobma"` refers to pseudo-BMA+ weighting (or plain pseudo-BMA weighting if `BB` is `FALSE`).
`optim_method`	The optimization method to use if `method="stacking"`. It can be chosen from "Nelder-Mead", "BFGS", "CG", "L-BFGS-B", "SANN" and "Brent". The default method is "BFGS".
`optim_control`	If `method="stacking"`, a list of control parameters for optimization. See `constrOptim` for details.
`BB`	Logical used when `"method"="pseudobma"`. If `TRUE` (the default), the Bayesian bootstrap will be used to adjust the pseudo-BMA weighting, which is called pseudo-BMA+ weighting. It helps regularize the weight away from 0 and 1, so as to reduce the variance.
`BB_n`	For pseudo-BMA+ weighting only, the number of samples to use for the Bayesian bootstrap. The default is 1000.
`alpha`	Positive scalar shape parameter in the Dirichlet distribution used for the Bayesian bootstrap. The default is 1, which corresponds to a uniform distribution on the simplex space.
`r_eff_list`	Optionally, a list of relative effective sample size estimates for the likelihood `(exp(log_lik))` of each observation in each model. See `psis` and `relative_eff` helper function for computing `r_eff`. If `x` is a list of "psis_loo" objects then `r_eff_list` is ignored.
`cores`	The number of cores to use for parallelization. This defaults to the option `mc.cores` which can be set for an entire R session by `options(mc.cores = NUMBER)`. The old option `loo.cores` is now deprecated but will be given precedence over `mc.cores` until `loo.cores` is removed in a future release. **As of version 2.0.0 the default is now 1 core if `mc.cores` is not set, but we recommend using as many (or close to as many) cores as possible.**
`lpd_point`	A matrix of pointwise leave-one-out (or K-fold) log likelihoods evaluated for different models. It should be a N by K matrix where N is sample size and K is the number of models. Each column corresponds to one model. These values can be calculated approximately using `loo` or by running exact leave-one-out or K-fold cross-validation.

Details

loo_model_weights is a wrapper around the stacking_weights and pseudobma_weights functions that implements stacking, pseudo-BMA, and pseudo-BMA+ weighting for combining multiple predictive distributions. We can use approximate or exact leave-one-out cross-validation (LOO-CV) or K-fold CV to estimate the expected log predictive density (ELPD).

The stacking method (method="stacking") combines all models by maximizing the leave-one-out predictive density of the combination distribution. That is, it finds the optimal linear combining weights for maximizing the leave-one-out log score.

The pseudo-BMA method (method="pseudobma") finds the relative weights proportional to the ELPD of each model. However, when method="pseudobma", the default is to also use the Bayesian bootstrap (BB=TRUE), which corresponds to the pseudo-BMA+ method. The Bayesian bootstrap takes into account the uncertainty of finite data points and regularizes the weights away from the extremes of 0 and 1.

In general, we recommend stacking for averaging predictive distributions, while pseudo-BMA+ can serve as a computationally easier alternative.

Value

A numeric vector containing one weight for each model.

References

Vehtari, A., Gelman, A., and Gabry, J. (2017a). Practical Bayesian model evaluation using leave-one-out cross-validation and WAIC. *Statistics and Computing*. 27(5), 1413–1432. doi:10.1007/s11222-016-9696-4. (published version, arXiv preprint).

Vehtari, A., Gelman, A., and Gabry, J. (2017b). Pareto smoothed importance sampling. arXiv preprint: http://arxiv.org/abs/1507.02646/

Yao, Y., Vehtari, A., Simpson, D., and Gelman, A. (2018) Using stacking to average Bayesian predictive distributions. *Bayesian Analysis*, advance publication, doi:10.1214/17-BA1091. (online).

See Also

- The **loo** package vignettes for demonstrations.
- loo for details on leave-one-out ELPD estimation.
- constrOptim for the choice of optimization methods and control-parameters.
- relative_eff for computing r_eff.

Examples

```
## Not run:
### Demonstrating usage after fitting models with RStan
library(rstan)

# generate fake data from N(0,1).
N <- 100
y <- rnorm(N, 0, 1)
```

```
# Suppose we have three models: N(-1, sigma), N(0.5, sigma) and N(0.6,sigma).
stan_code <- "
  data {
    int N;
    vector[N] y;
    real mu_fixed;
  }
  parameters {
    real<lower=0> sigma;
  }
  model {
    sigma ~ exponential(1);
    y ~ normal(mu_fixed, sigma);
  }
  generated quantities {
    vector[N] log_lik;
    for (n in 1:N) log_lik[n] = normal_lpdf(y[n]| mu_fixed, sigma);
  }"

mod <- stan_model(model_code = stan_code)
fit1 <- sampling(mod, data=list(N=N, y=y, mu_fixed=-1))
fit2 <- sampling(mod, data=list(N=N, y=y, mu_fixed=0.5))
fit3 <- sampling(mod, data=list(N=N, y=y, mu_fixed=0.6))
model_list <- list(fit1, fit2, fit3)
log_lik_list <- lapply(model_list, extract_log_lik)

# optional but recommended
r_eff_list <- lapply(model_list, function(x) {
  ll_array <- extract_log_lik(x, merge_chains = FALSE)
  relative_eff(exp(ll_array))
})

# stacking method:
wts1 <- loo_model_weights(
  log_lik_list,
  method = "stacking",
  r_eff_list = r_eff_list,
  optim_control = list(reltol=1e-10)
)
print(wts1)

# can also pass a list of psis_loo objects to avoid recomputing loo
loo_list <- lapply(1:length(log_lik_list), function(j) {
  loo(log_lik_list[[j]], r_eff = r_eff_list[[j]])
})

wts2 <- loo_model_weights(
  loo_list,
  method = "stacking",
  optim_control = list(reltol=1e-10)
)
all.equal(wts1, wts2)
```

```
# pseudo-BMA+ method:
set.seed(1414)
loo_model_weights(loo_list, method = "pseudobma")

# pseudo-BMA method (set BB = FALSE):
loo_model_weights(loo_list, method = "pseudobma", BB = FALSE)

# calling stacking_weights or pseudobma_weights directly
lpd1 <- loo(log_lik_list[[1]], r_eff = r_eff_list[[1]])$pointwise[,1]
lpd2 <- loo(log_lik_list[[2]], r_eff = r_eff_list[[2]])$pointwise[,1]
lpd3 <- loo(log_lik_list[[3]], r_eff = r_eff_list[[3]])$pointwise[,1]
stacking_weights(cbind(lpd1, lpd2, lpd3))
pseudobma_weights(cbind(lpd1, lpd2, lpd3))
pseudobma_weights(cbind(lpd1, lpd2, lpd3), BB = FALSE)

## End(Not run)
```

`pareto-k-diagnostic` *Diagnostics for Pareto smoothed importance sampling (PSIS)*

Description

Print a diagnostic table summarizing the estimated Pareto shape parameters and PSIS effective sample sizes, find the indexes of observations for which the estimated Pareto shape parameter k is larger than some `threshold` value, or plot observation indexes vs. diagnostic estimates. The **Details** section below provides a brief overview of the diagnostics, but we recommend consulting Vehtari, Gelman, and Gabry (2017a, 2017b) for full details.

Usage

```
pareto_k_table(x)

pareto_k_ids(x, threshold = 0.5)

pareto_k_values(x)

psis_n_eff_values(x)

mcse_loo(x, threshold = 0.7)

## S3 method for class 'psis_loo'
plot(x, diagnostic = c("k", "n_eff"), ...,
  label_points = FALSE, main = "PSIS diagnostic plot")

## S3 method for class 'psis'
plot(x, diagnostic = c("k", "n_eff"), ...,
  label_points = FALSE, main = "PSIS diagnostic plot")
```

Arguments

`x`	An object created by `loo` or `psis`.
`threshold`	For `pareto_k_ids`, `threshold` is the minimum k value to flag (default is 0.5). For `mcse_loo`, if any k estimates are greater than `threshold` the MCSE estimate is returned as `NA` (default is 0.7).
`diagnostic`	For the `plot` method, which diagnostic should be plotted? The options are `"k"` for Pareto k estimates (the default) or `"n_eff"` for PSIS effective sample size estimates.
`label_points, ...`	For the `plot` method, if `label_points` is `TRUE` the observation numbers corresponding to any values of k greater than 0.5 will be displayed in the plot. Any arguments specified in `...` will be passed to `text` and can be used to control the appearance of the labels.
`main`	For the `plot` method, a title for the plot.

Details

The reliability and approximate convergence rate of the PSIS-based estimates can be assessed using the estimates for the shape parameter k of the generalized Pareto distribution:

- If $k < 0.5$ then the distribution of raw importance ratios has finite variance and the central limit theorem holds. However, as k approaches 0.5 the RMSE of plain importance sampling (IS) increases significantly while PSIS has lower RMSE.
- If $0.5 \leq k < 1$ then the variance of the raw importance ratios is infinite, but the mean exists. TIS and PSIS estimates have finite variance by accepting some bias. The convergence of the estimate is slower with increasing k. If k is between 0.5 and approximately 0.7 then we observe practically useful convergence rates and Monte Carlo error estimates with PSIS (the bias of TIS increases faster than the bias of PSIS). If $k > 0.7$ we observe impractical convergence rates and unreliable Monte Carlo error estimates.
- If $k \geq 1$ then neither the variance nor the mean of the raw importance ratios exists. The convergence rate is close to zero and bias can be large with practical sample sizes.

If the estimated tail shape parameter k exceeds 0.5, the user should be warned, although in practice we have observed good performance for values of k up to 0.7. (If k is greater than 0.5 then WAIC is also likely to fail, but WAIC lacks its own diagnostic.)

If using PSIS in the context of approximate LOO-CV, even if the PSIS estimate has a finite variance the user should consider sampling directly from $p(\theta^s|y_{-i})$ for any problematic observations i, use K-fold cross-validation, or use a more robust model. Importance sampling is likely to work less well if the marginal posterior $p(\theta^s|y)$ and LOO posterior $p(\theta^s|y_{-i})$ are much different, which is more likely to happen with a non-robust model and highly influential observations. A robust model may reduce the sensitivity to highly influential observations.

Effective sample size and error estimates: In the case that we obtain the samples from the proposal distribution via MCMC we can also compute estimates for the Monte Carlo error and the effective sample size for importance sampling, which are more accurate for PSIS than for IS and TIS (see Vehtari et al (2017b) for details). However, the PSIS effective sample size estimate will be **over-optimistic when the estimate of k is greater than 0.7.**

We can also compute estimates for the Monte Carlo error and the effective sample size for importance sampling. However, the PSIS effective sample size estimate will be **over-optimistic when the estimate of k is greater than 0.7**. In the case that we obtain the samples from the proposal distribution via MCMC, we need to take into account also the relative efficiency of MCMC sampling (see Vehtari et al (2017b) for details). Following the notation in Stan, the PSIS effective sample size is denoted here with n_{eff}, instead of S_{eff} used by Vehtari et al (2017b).

Value

`pareto_k_table` returns an object of class `"pareto_k_table"`, which is a matrix with columns `"Count"`, `"Proportion"`, and `"Min. n_eff"`, and has its own print method.

`pareto_k_ids` returns an integer vector indicating which observations have Pareto k estimates above `threshold`.

`pareto_k_values` returns a vector of the estimated Pareto k parameters.

`psis_n_eff_values` returns a vector of the estimated PSIS effective sample sizes.

`mcse_loo` returns the Monte carlo standard error (MCSE) estimate for PSIS-LOO. MCSE will be NA if any Pareto k values are above `threshold`.

The `plot` method is called for its side effect and does not return anything. If `x` is the result of a call to `loo` or `psis` then `plot(x, diagnostic)` produces a plot of the estimates of the Pareto shape parameters (`diagnostic = "k"`) or estimates of the PSIS effective sample sizes (`diagnostic = "n_eff"`).

References

Vehtari, A., Gelman, A., and Gabry, J. (2017a). Practical Bayesian model evaluation using leave-one-out cross-validation and WAIC. *Statistics and Computing*. 27(5), 1413–1432. doi:10.1007/s11222-016-9696-4. (published version, arXiv preprint).

Vehtari, A., Gelman, A., and Gabry, J. (2017b). Pareto smoothed importance sampling. arXiv preprint: `http://arxiv.org/abs/1507.02646/`

See Also

`psis` for the implementation of the PSIS algorithm.

`print.loo` *Print methods*

Description

Print methods

Usage

```
## S3 method for class 'loo'
print(x, digits = 1, ...)

## S3 method for class 'waic'
print(x, digits = 1, ...)

## S3 method for class 'psis_loo'
print(x, digits = 1, plot_k = FALSE, ...)

## S3 method for class 'psis'
print(x, digits = 1, plot_k = FALSE, ...)
```

Arguments

`x`	An object returned by `loo`, `psis`, or `waic`.
`digits`	An integer passed to `round`.
`...`	Arguments passed to `plot.psis_loo` if `plot_k` is `TRUE`.
`plot_k`	Logical. If `TRUE` the estimates of the Pareto shape parameter k are plotted. Ignored if `x` was generated by `waic`. To just plot k without printing use the `plot method`.

Value

`x`, invisibly.

See Also

`pareto-k-diagnostic`

---

psis *Pareto smoothed importance sampling (PSIS)*

Description

Implementation of Pareto smoothed importance sampling (PSIS), a method for stabilizing importance ratios. The version of PSIS implemented here corresponds to the algorithm presented in Vehtari, Gelman and Gabry (2017b). For PSIS diagnostics see the pareto-k-diagnostic page.

Usage

```
psis(log_ratios, ...)

## S3 method for class 'array'
psis(log_ratios, ..., r_eff = NULL,
  cores = getOption("mc.cores", 1))
```

```
## S3 method for class 'matrix'
psis(log_ratios, ..., r_eff = NULL,
  cores = getOption("mc.cores", 1))

## Default S3 method:
psis(log_ratios, ..., r_eff = NULL)

## S3 method for class 'psis'
weights(object, ..., log = TRUE, normalize = TRUE)
```

Arguments

`log_ratios`	An array, matrix, or vector of importance ratios on the log scale (for PSIS-LOO these are *negative* log-likelihood values). See the **Methods (by class)** section below for a detailed description of how to specify the inputs for each method.
`...`	Arguments passed on to the various methods.
`r_eff`	Vector of relative effective sample size estimates containing one element per observation. The values provided should be the relative effective sample sizes of `1/exp(log_ratios)` (i.e., `1/ratios`). This is related to the relative efficiency of estimating the normalizing term in self-normalizing importance sampling. If `r_eff` is not provided then the reported PSIS effective sample sizes and Monte Carlo error estimates will be over-optimistic. See the `relative_eff` helper function for computing `r_eff`.
`cores`	The number of cores to use for parallelization. This defaults to the option `mc.cores` which can be set for an entire R session by `options(mc.cores = NUMBER)`. The old option `loo.cores` is now deprecated but will be given precedence over `mc.cores` until `loo.cores` is removed in a future release. **As of version 2.0.0 the default is now 1 core if `mc.cores` is not set, but we recommend using as many (or close to as many) cores as possible.**
`object`	For the `weights` method, an object returned by `psis` (a list with class `"psis"`).
`log`	For the `weights` method, should the weights be returned on the log scale? Defaults to `TRUE`.
`normalize`	For the `weights` method, should the weights be normalized? Defaults to `TRUE`.

Value

The `psis` methods return an object of class `"psis"`, which is a named list with the following components:

`log_weights` Vector or matrix of smoothed (and truncated) but *unnormalized* log weights. To get normalized weights use the `weights` method provided for objects of class `"psis"`.

`diagnostics` A named list containing two vectors:

- `pareto_k`: Estimates of the shape parameter k of the generalized Pareto distribution. See the `pareto-k-diagnostic` page for details.
- `n_eff`: PSIS effective sample size estimates.

Objects of class "psis" also have the following attributes:

norm_const_log Vector of precomputed values of colLogSumExps(log_weights) that are used internally by the weights method to normalize the log weights.

tail_len Vector of tail lengths used for fitting the generalized Pareto distribution.

r_eff If specified, the user's r_eff argument.

dims Integer vector of length 2 containing S (posterior sample size) and N (number of observations).

The weights method returns an object with the same dimensions as the log_weights component of the "psis" object. The normalize and log arguments control whether the returned weights are normalized and whether or not to return them on the log scale.

Methods (by class)

- array: An I by C by N array, where I is the number of MCMC iterations per chain, C is the number of chains, and N is the number of data points.
- matrix: An S by N matrix, where S is the size of the posterior sample (with all chains merged) and N is the number of data points.
- default: A vector of length S (posterior sample size).

References

Vehtari, A., Gelman, A., and Gabry, J. (2017a). Practical Bayesian model evaluation using leave-one-out cross-validation and WAIC. *Statistics and Computing*. 27(5), 1413–1432. doi:10.1007/s11222-016-9696-4. (published version, arXiv preprint).

Vehtari, A., Gelman, A., and Gabry, J. (2017b). Pareto smoothed importance sampling. arXiv preprint: http://arxiv.org/abs/1507.02646/

See Also

- loo for approximate LOO-CV using PSIS.
- pareto-k-diagnostic for PSIS diagnostics.

Examples

```
log_ratios <- -1 * example_loglik_array()
r_eff <- relative_eff(exp(log_ratios))
psis_result <- psis(log_ratios, r_eff = r_eff)
str(psis_result)
plot(psis_result)

# extract smoothed weights
lw <- weights(psis_result) # default args are log=TRUE, normalize=TRUE
ulw <- weights(psis_result, normalize=FALSE) # unnormalized log-weights

w <- weights(psis_result, log=FALSE) # normalized weights (not log-weights)
uw <- weights(psis_result, log=FALSE, normalize = FALSE) # unnormalized weights
```

psislw *Pareto smoothed importance sampling (deprecated, old version)*

Description

As of version `2.0.0` this function is deprecated. Please use the `psis` function for the new PSIS algorithm.

Usage

```
psislw(lw, wcp = 0.2, wtrunc = 3/4, cores = getOption("mc.cores", 1),
  llfun = NULL, llargs = NULL, ...)
```

Arguments

`lw`	A matrix or vector of log weights. For computing LOO, `lw = -log_lik`, the *negative* of an S (simulations) by N (data points) pointwise log-likelihood matrix.
`wcp`	The proportion of importance weights to use for the generalized Pareto fit. The `100*wcp`% largest weights are used as the sample from which to estimate the parameters of the generalized Pareto distribution.
`wtrunc`	For truncating very large weights to S^`wtrunc`. Set to zero for no truncation.
`cores`	The number of cores to use for parallelization. This defaults to the option `mc.cores` which can be set for an entire R session by `options(mc.cores = NUMBER)`, the old option `loo.cores` is now deprecated but will be given precedence over `mc.cores` until it is removed. **As of version 2.0.0, the default is now 1 core if** `mc.cores` **is not set, but we recommend using as many (or close to as many) cores as possible.**
`llfun, llargs`	See `loo.function`.
`...`	Ignored when `psislw` is called directly. The `...` is only used internally when `psislw` is called by the `loo` function.

Value

A named list with components `lw_smooth` (modified log weights) and `pareto_k` (estimated generalized Pareto shape parameter(s) k).

References

Vehtari, A., Gelman, A., and Gabry, J. (2017a). Practical Bayesian model evaluation using leave-one-out cross-validation and WAIC. *Statistics and Computing*. 27(5), 1413–1432. doi:10.1007/s11222-016-9696-4. (published version, arXiv preprint).

Vehtari, A., Gelman, A., and Gabry, J. (2017b). Pareto smoothed importance sampling. arXiv preprint: `http://arxiv.org/abs/1507.02646/`

See Also

pareto-k-diagnostic for PSIS diagnostics.

---

relative_eff *Convenience function for computing relative efficiencies*

Description

relative_eff computes the the MCMC effective sample size divided by the total sample size.

Usage

```
relative_eff(x, ...)

## Default S3 method:
relative_eff(x, chain_id, ...)

## S3 method for class 'matrix'
relative_eff(x, chain_id, ..., cores = getOption("mc.cores",
  1))

## S3 method for class 'array'
relative_eff(x, ..., cores = getOption("mc.cores", 1))

## S3 method for class 'function'
relative_eff(x, chain_id, ...,
  cores = getOption("mc.cores", 1), data = NULL, draws = NULL)
```

Arguments

x	A vector, matrix, 3-D array, or function. See the **Methods (by class)** section below for details on the shape of x. For use with the loo function, the values in x (or generated by x if x is a function) should be **likelihood** values (i.e., exp(log_lik), not on the log scale). For generic use with psis, the values in x should be the reciprocal of the importance ratios (i.e., exp(-log_ratios)).
chain_id	A vector of length NROW(x) containing MCMC chain indexes for each each row of x (if a matrix) or each value in x (if a vector). No chain_id is needed if x is a 3-D array. If there are C chains then valid chain indexes are values in 1:C.
cores	The number of cores to use for parallelization.
data, draws, ...	Same as for the loo function method.

Value

A vector of relative effective sample sizes.

Methods (by class)

- `default`: A vector of length S (posterior sample size).
- `matrix`: An S by N matrix, where S is the size of the posterior sample (with all chains merged) and N is the number of data points.
- `array`: An I by C by N array, where I is the number of MCMC iterations per chain, C is the number of chains, and N is the number of data points.
- `function`: A function `f` that takes arguments `data_i` and `draws` and returns a vector containing the log-likelihood for a single observation `i` evaluated at each posterior draw. The function should be written such that, for each observation `i` in `1:N`, evaluating `f(data_i = data[i,, drop=FALSE], draws = c` results in a vector of length `S` (size of posterior sample). The log-likelihood function can also have additional arguments but `data_i` and `draws` are required.

  If using the function method then the arguments `data` and `draws` must also be specified in the call to `loo`:

  - `data`: A data frame or matrix containing the data (e.g. observed outcome and predictors) needed to compute the pointwise log-likelihood. For each observation `i`, the `i`th row of `data` will be passed to the `data_i` argument of the log-likelihood function.
  - `draws`: An object containing the posterior draws for any parameters needed to compute the pointwise log-likelihood. Unlike `data`, which is indexed by observation, for each observation the entire object `draws` will be passed to the `draws` argument of the log-likelihood function.
  - The `...` can be used to pass additional arguments to your log-likelihood function. These arguments are used like the `draws` argument in that they are recycled for each observation.

Examples

```
LLarr <- example_loglik_array()
LLmat <- example_loglik_matrix()
dim(LLarr)
dim(LLmat)

rel_n_eff_1 <- relative_eff(exp(LLarr))
rel_n_eff_2 <- relative_eff(exp(LLmat), chain_id = rep(1:2, each = 500))
all.equal(rel_n_eff_1, rel_n_eff_2)
```

`waic` *Widely applicable information criterion (WAIC)*

Description

The `waic` methods can be used to compute WAIC from the pointwise log-likelihood. However, we recommend LOO-CV using PSIS (as implemented by the `loo` function) because PSIS provides useful diagnostics and effective sample size and Monte Carlo estimates.

Usage

```
waic(x, ...)

## S3 method for class 'array'
waic(x, ...)

## S3 method for class 'matrix'
waic(x, ...)

## S3 method for class 'function'
waic(x, ..., data = NULL, draws = NULL)
```

Arguments

`x`	A log-likelihood array, matrix, or function. See the **Methods (by class)** section below for a detailed description of how to specify the inputs for each method.
`draws, data, ...`	For the function method only. See the **Methods (by class)** section below for details on these arguments.

Value

A named list (of class `c("waic", "loo")`) with components:

`estimates` A matrix with two columns (`"Estimate"`, `"SE"`) and three rows (`"elpd_waic"`, `"p_waic"`, `"waic"`). This contains point estimates and standard errors of the expected log pointwise predictive density (`elpd_waic`), the effective number of parameters (`p_waic`) and the LOO information criterion `waic` (which is just `-2 * elpd_waic`, i.e., converted to deviance scale).

`pointwise` A matrix with three columns (and number of rows equal to the number of observations) containing the pointwise contributions of each of the above measures (`elpd_waic`, `p_waic`, `waic`).

Methods (by class)

- `array`: An I by C by N array, where I is the number of MCMC iterations per chain, C is the number of chains, and N is the number of data points.
- `matrix`: An S by N matrix, where S is the size of the posterior sample (with all chains merged) and N is the number of data points.
- `function`: A function `f` that takes arguments `data_i` and `draws` and returns a vector containing the log-likelihood for a single observation `i` evaluated at each posterior draw. The function should be written such that, for each observation `i` in `1:N`, evaluating `f(data_i = data[i,, drop=FALSE], draws = c` results in a vector of length `S` (size of posterior sample). The log-likelihood function can also have additional arguments but `data_i` and `draws` are required.

  If using the function method then the arguments `data` and `draws` must also be specified in the call to `loo`:

  - `data`: A data frame or matrix containing the data (e.g. observed outcome and predictors) needed to compute the pointwise log-likelihood. For each observation `i`, the `i`th row of `data` will be passed to the `data_i` argument of the log-likelihood function.

  - `draws`: An object containing the posterior draws for any parameters needed to compute the pointwise log-likelihood. Unlike `data`, which is indexed by observation, for each observation the entire object `draws` will be passed to the `draws` argument of the log-likelihood function.
  - The `...` can be used to pass additional arguments to your log-likelihood function. These arguments are used like the `draws` argument in that they are recycled for each observation.

See Also

- `loo` for approximate LOO-CV.
- `compare` for comparing models on LOOIC or WAIC.

Examples

```
### Array and matrix methods
LLarr <- example_loglik_array()
dim(LLarr)

LLmat <- example_loglik_matrix()
dim(LLmat)

waic_arr <- waic(LLarr)
waic_mat <- waic(LLmat)
identical(waic_arr, waic_mat)


## Not run:
log_lik1 <- extract_log_lik(stanfit1)
log_lik2 <- extract_log_lik(stanfit2)
(waic1 <- waic(log_lik1))
(waic2 <- waic(log_lik2))
print(compare(waic1, waic2), digits = 2)

## End(Not run)
```

Index