Package ‘loo’

July 15, 2020

Type Package

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

Version 2.3.1

Date 2020-07-14

Maintainer Jonah Gabry <jsg2201@columbia.edu>

URL https://mc-stan.org/loo, https://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, as described in Vehtari, Gelman, and Gabry (2017) <doi:10.1007/s11222-016-9696-4>.
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 checkmate, matrixStats (>= 0.52), parallel, stats

Suggests bayesplot (>= 1.7.0), brms (>= 2.10.0), ggplot2, graphics, knitr, rmarkdown, rstan, rstanarm (>= 2.19.0), rstantools, spdep, testthat (>= 2.1.0)

VignetteBuilder knitr

Encoding UTF-8

SystemRequirements pandoc (>= 1.12.3), pandoc-citeproc

RoxygenNote 7.1.1

NeedsCompilation no

Author Aki Vehtari [aut],
Jonah Gabry [cre, aut],
Mans Magnusson [aut],
Yuling Yao [aut],
Paul-Christian Bürkner [aut],
Topi Paananen [aut],
Andrew Gelman [aut],
Ben Goodrich [ctb],
Juho Piironen [ctb]

Repository CRAN

Date/Publication 2020-07-14 22:50:10 UTC

R topics documented:

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

Description

Stan Development Team

This package implements the methods described in Vehtari, Gelman, and Gabry (2017), Vehtari, Simpson, Gelman, Yao, and Gabry (2019), and Yao et al. (2018). To get started see the **loo** package vignettes, 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, Simpson, Gelman, Yao, and Gabry, 2019), a new procedure for stabilizing 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 (journal version, preprint arXiv:1507.04544).

Vehtari, A., Simpson, D., Gelman, A., Yao, Y., and Gabry, J. (2019). Pareto smoothed importance sampling. preprint arXiv: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).

Magnusson, M., Riis Andersen, M., Jonasson, J. and Vehtari, A. (2019). Leave-One-Out Cross-Validation for Large Data. In *International Conference on Machine Learning*

Magnusson, M., Riis Andersen, M., Jonasson, J. and Vehtari, A. (2019). Leave-One-Out Cross-Validation for Model Comparison in Large Data.

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. https://mc-stan.org.

Stan Development Team (2018). RStan: the R interface to Stan, Version 2.17.3. https://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.

`ap_psis` *Pareto smoothed importance sampling (PSIS) using approximate posteriors*

Description

Pareto smoothed importance sampling (PSIS) using approximate posteriors

Usage

```
ap_psis(log_ratios, log_p, log_g, ...)

## S3 method for class 'array'
ap_psis(log_ratios, log_p, log_g, ..., cores = getOption("mc.cores", 1))

## S3 method for class 'matrix'
ap_psis(log_ratios, log_p, log_g, ..., cores = getOption("mc.cores", 1))

## Default S3 method:
ap_psis(log_ratios, log_p, log_g, ...)
```

Arguments

`log_ratios`	The log-likelihood ratios (ie -log_liks)
`log_p`	The log-posterior (target) evaluated at S samples from the proposal distribution (g). A vector of length S.
`log_g`	The log-density (proposal) evaluated at S samples from the proposal distribution (g). A vector of length S.
`...`	Currently not in use.
`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. • Note for Windows 10 users: it is **strongly** recommended to avoid using the `.Rprofile` file to set `mc.cores` (using the `cores` argument or setting `mc.cores` interactively or in a script is fine).

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).

`compare` *Model comparison (deprecated, old version)*

Description

This function is deprecated. Please use the new `loo_compare()` function instead.

Usage

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

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 `...`.

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` (or multiplied by -2, if desired, to be on the deviance scale).

When that difference, `elpd_diff`*, is positive then the expected predictive accuracy for the second model is higher. A negative* `elpd_diff` *favors the first model.*

When using `compare()` with more than two models, the values in the `elpd_diff` and `se_diff` columns of the returned matrix are computed by making pairwise comparisons between each model and the model with the best ELPD (i.e., the model in the first row). Although the `elpd_diff` column is equal to the difference in `elpd_loo`, do not expect the `se_diff` column to be equal to the the difference in `se_elpd_loo`.

To compute the standard error of the difference in ELPD we 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. If more than two objects are provided then a matrix of summary information is returned (see **Details**).

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 (journal version, preprint arXiv:1507.04544).

Vehtari, A., Simpson, D., Gelman, A., Yao, Y., and Gabry, J. (2019). Pareto smoothed importance sampling. preprint arXiv: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. https://mc-stan.org/

Stan Development Team (2017). RStan: the R interface to Stan, Version 2.16.1. https://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="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, refresh = 0)
yrep <- posterior_predict(fit)
dim(yrep)

log_ratios <- -1 * log_lik(fit)
dim(log_ratios)

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

---

gpdfit *Estimate parameters of the Generalized Pareto distribution*

Description

Given a sample x, Estimate the parameters k and σ of the generalized Pareto distribution (GPD), assuming the location parameter is 0. By default the 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

---

kfold-generic *Generic function for K-fold cross-validation for developers*

Description

For developers of Bayesian modeling packages, **loo** includes a generic function `kfold()` so that methods may be defined for K-fold CV without name conflicts between packages. See, for example, the `kfold()` methods in the **rstanarm** and **brms** packages.

The **Value** section below describes the objects that `kfold()` methods should return in order to be compatible with `loo_compare()` and the **loo** package print methods.

Usage

```
kfold(x, ...)

is.kfold(x)
```

Arguments

`x`	A fitted model object.
`...`	Arguments to pass to specific methods.

Value

For developers defining a `kfold()` method for a class `"foo"`, the `kfold.foo()` function should return a list with class `c("kfold","loo")` with at least the following named elements:

- `"estimates"`: A 1x2 matrix containing the ELPD estimate and its standard error. The matrix must have row name "`elpd_kfold`" and column names `"Estimate"` and `"SE"`.
- `"pointwise"`: A `Nx1` matrix with column name `"elpd_kfold"` containing the pointwise contributions for each data point.

It is important for the object to have at least these classes and components so that it is compatible with other functions like `loo_compare()` and `print()` methods.

`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. See the **Details** section for explanations.

Usage

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

kfold_split_stratified(K = 10, x = NULL)

kfold_split_grouped(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` with at least `K` levels (unique values). Will be coerced to a factor.

Details

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

For a categorical variable `x` `kfold_split_stratified()` splits the observations into `K` groups ensuring that relative category frequencies are approximately preserved.

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

Value

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

Examples

```
ids <- kfold_split_random(K = 5, N = 20)
print(ids)
table(ids)


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

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

ids_10 <- kfold_split_grouped(K = 10, x = grp)
(tab_10 <- table(grp, ids_10))
colSums(tab_10)

ids_9 <- kfold_split_grouped(K = 9, x = grp)
(tab_9 <- table(grp, ids_9))
colSums(tab_9)
```

---

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 (2017) and Vehtari, Simpson, Gelman, Yao, and Gabry (2019).

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),
  is_method = c("psis", "tis", "sis")
)

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

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

loo_i(
  i,
  llfun,
  ...,
  data = NULL,
  draws = NULL,
  r_eff = NULL,
  is_method = "psis"
)

is.loo(x)

is.psis_loo(x)
```

Arguments

`x` A log-likelihood array, matrix, or function. The **Methods (by class)** section, below, has detailed descriptions 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 when using posterior draws obtained with MCMC. If MCMC draws are used and `r_eff` is not provided then the reported PSIS effective sample sizes and Monte Carlo error estimates will be over-optimistic. If the posterior draws are independent then `r_eff=1` and can be omitted. See the `relative_eff()` helper functions 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=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.

- Note for Windows 10 users: it is **strongly** recommended to avoid using the `.Rprofile` file to set `mc.cores` (using the `cores` argument or setting `mc.cores` interactively or in a script is fine).

`is_method` The importance sampling method to use. The following methods are implemented:

- `"psis"`: Pareto-Smoothed Importance Sampling (PSIS). Default method.
- `"tis"`: Truncated Importance Sampling (TIS) with truncation at `sqrt(S)`, where `S` is the number of posterior draws.
- `"sis"`: Standard Importance Sampling (SIS).

`data, draws, ...`

For the `loo.function()` method and the `loo_i()` function, these are 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 3-D pointwise log-likelihood arrays, pointwise log-likelihood matrices, and log-likelihood functions. The array and matrix methods are the 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 three rows (elpd_loo, p_loo, looic). This contains point estimates and standard errors of the expected log pointwise predictive density (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 five columns (and number of rows equal to the number of observations) containing the pointwise contributions of the measures (elpd_loo, mcse_elpd_loo, p_loo, looic, influence_pareto_k). in addition to the three measures in estimates, we also report pointwise values of the Monte Carlo standard error of elpd_loo (mcse_elpd_loo), and statistics describing the influence of each observation on the posterior distribution (influence_pareto_k). These are the 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.

diagnostics A named list containing two vectors:

- pareto_k: Importance sampling reliability diagnostics. By default, these are equal to the influence_pareto_k in pointwise. Some algorithms can improve importance sampling reliability and modify these diagnostics. 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 = draws)
  ```

  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 ith 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 if your log-likelihood function takes additional arguments. 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 the **rstanarm** package 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 (journal version, preprint arXiv:1507.04544).

Vehtari, A., Simpson, D., Gelman, A., Yao, Y., and Gabry, J. (2019). Pareto smoothed importance sampling. preprint arXiv: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.
- loo_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, r_eff = NA)
print(loo_3$pointwise[, "elpd_loo"])

# Use loo.function method (setting r_eff=NA since this posterior not obtained via MCMC)
loo_with_fn <- loo(llfun, draws = fake_posterior, data = fake_data, r_eff = NA)

# 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, r_eff = NA)
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 on CRAN.

Details

Currently the data sets included are:

- `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-glossary` *LOO package glossary*

Description

Note: VGG2017 refers to Vehtari, Gelman, and Gabry (2017). See **References**, below.

ELPD and `elpd_loo`

The ELPD is the theoretical expected log pointwise predictive density for a new dataset (Eq 1 in VGG2017), which can be estimated, e.g., using cross-validation. `elpd_loo` is the Bayesian LOO estimate of the expected log pointwise predictive density (Eq 4 in VGG2017) and is a sum of N individual pointwise log predictive densities. Probability densities can be smaller or larger than 1, and thus log predictive densities can be negative or positive. For simplicity the ELPD acronym is used also for expected log pointwise predictive probabilities for discrete models. Probabilities are always equal or less than 1, and thus log predictive probabilities are 0 or negative.

Standard error of `elpd_loo`

As `elpd_loo` is defined as the sum of N independent components (Eq 4 in VGG2017), we can compute the standard error by using the standard deviation of the N components and multiplying by `sqrt(N)` (Eq 23 in VGG2017). This standard error is a coarse description of our uncertainty about the predictive performance for unknown future data. When N is small or there is severe model misspecification, the current SE estimate is overoptimistic and the actual SE can even be twice as large. Even for moderate N, when the SE estimate is an accurate estimate for the scale, it ignores the skewness. When making model comparisons, the SE of the component-wise (pairwise) differences should be used instead (see the `se_diff` section below and Eq 24 in VGG2017).

Monte Carlo SE of elpd_loo

The Monte Carlo standard error is the estimate for the computational accuracy of MCMC and importance sampling used to compute `elpd_loo`. Usually this is negligible compared to the standard describing the uncertainty due to finite number of observations (Eq 23 in VGG2017).

`p_loo` (effective number of parameters)

`p_loo` is the difference between `elpd_loo` and the non-cross-validated log posterior predictive density. It describes how much more difficult it is to predict future data than the observed data. Asymptotically under certain regularity conditions, `p_loo` can be interpreted as the *effective number of parameters*. In well behaving cases `p_loo < N` and `p_loo < p`, where `p` is the total number of parameters in the model. `p_loo > N` or `p_loo > p` indicates that the model has very weak predictive capability and may indicate a severe model misspecification. See below for more on interpreting `p_loo` when there are warnings about high Pareto k diagnostic values.

Pareto k estimates

The Pareto k estimate is a diagnostic for Pareto smoothed importance sampling (PSIS), which is used to compute components of elpd_loo. In importance-sampling LOO (the full posterior distribution is used as the proposal distribution). The Pareto k diagnostic estimates how far an individual leave-one-out distribution is from the full distribution. If leaving out an observation changes the posterior too much then importance sampling is not able to give reliable estimate. If k<0.5, then the corresponding component of elpd_loo is estimated with high accuracy. If 0.5<k<0.7 the accuracy is lower, but still ok. If k>0.7, then importance sampling is not able to provide useful estimate for that component/observation. Pareto k is also useful as a measure of influence of an observation. Highly influential observations have high k values. Very high k values often indicate model misspecification, outliers or mistakes in data processing. See Section 6 of Gabry et al. (2019) for an example.

Interpreting p_loo when Pareto k is large: If k > 0.7 then we can also look at the p_loo estimate for some additional information about the problem:

- If p_loo << p (the total number of parameters in the model), then the model is likely to be misspecified. Posterior predictive checks (PPCs) are then likely to also detect the problem. Try using an overdispersed model, or add more structural information (nonlinearity, mixture model, etc.).
- If p_loo < p and the number of parameters p is relatively large compared to the number of observations (e.g., p>N/5), it is likely that the model is so flexible or the population prior so weak that it's difficult to predict the left out observation (even for the true model). This happens, for example, in the simulated 8 schools (in VGG2017), random effect models with a few observations per random effect, and Gaussian processes and spatial models with short correlation lengths.
- If p_loo > p, then the model is likely to be badly misspecified. If the number of parameters p<<N, then PPCs are also likely to detect the problem. See the case study at https://avehtari.github.io/modelselection/roaches.html for an example. If p is relatively large compared to the number of observations, say p>N/5 (more accurately we should count number of observations influencing each parameter as in hierarchical models some groups may have few observations and other groups many), it is possible that PPCs won't detect the problem.

elpd_diff

elpd_diff is the difference in elpd_loo for two models. If more than two models are compared, the difference is computed relative to the model with highest elpd_loo.

se_diff

The standard error of component-wise differences of elpd_loo (Eq 24 in VGG2017) between two models. This SE is *smaller* than the SE for individual models due to correlation (i.e., if some observations are easier and some more difficult to predict for all models).

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 (journal version, preprint arXiv:1507.04544).

Vehtari, A., Simpson, D., Gelman, A., Yao, Y., and Gabry, J. (2019). Pareto smoothed importance sampling. preprint arXiv:1507.02646

Gabry, J. , Simpson, D. , Vehtari, A. , Betancourt, M. and Gelman, A. (2019), Visualization in Bayesian workflow. *J. R. Stat. Soc. A*, 182: 389-402. doi:10.1111/rssa.12378 (journal version, preprint arXiv:1709.01449, code on GitHub)

`loo_approximate_posterior` *Efficient approximate leave-one-out cross-validation (LOO) for posterior approximations*

Description

Efficient approximate leave-one-out cross-validation (LOO) for posterior approximations

Usage

```
loo_approximate_posterior(x, log_p, log_g, ...)

## S3 method for class 'array'
loo_approximate_posterior(
  x,
  log_p,
  log_g,
  ...,
  save_psis = FALSE,
  cores = getOption("mc.cores", 1)
)

## S3 method for class 'matrix'
loo_approximate_posterior(
  x,
  log_p,
  log_g,
  ...,
  save_psis = FALSE,
  cores = getOption("mc.cores", 1)
)

## S3 method for class '`function`'
loo_approximate_posterior(
  x,
  ...,
  data = NULL,
  draws = NULL,
  log_p = NULL,
  log_g = NULL,
```

```
  save_psis = FALSE,
  cores = getOption("mc.cores", 1)
)
```

Arguments

x
: A log-likelihood array, matrix, or function. The **Methods (by class)** section, below, has detailed descriptions of how to specify the inputs for each method.

log_p
: The log-posterior (target) evaluated at S samples from the proposal distribution (g). A vector of length S.

log_g
: The log-density (proposal) evaluated at S samples from the proposal distribution (g). A vector of length S.

save_psis
: Should the `"psis"` object created internally by `loo_approximate_posterior()` be saved in the returned object? See `loo()` for details.

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.

  - Note for Windows 10 users: it is **strongly** recommended to avoid using the `.Rprofile` file to set `mc.cores` (using the `cores` argument or setting `mc.cores` interactively or in a script is fine).

data, draws, ...
: For the `loo_approximate_posterior.function()` method, these are 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.

Details

The `loo_approximate_posterior()` function is an S3 generic and methods are provided for 3-D pointwise log-likelihood arrays, pointwise log-likelihood matrices, and log-likelihood functions. The implementation works for posterior approximations where it is possible to compute the log density for the posterior approximation.

Value

The `loo_approximate_posterior()` methods return a named list with class `c("psis_loo_ap","psis_loo","loo")`. It has the same structure as the objects returned by `loo()` but with the additional slot:

`posterior_approximation` A list with two vectors, `log_p` and `log_g` of the same length containing the posterior density and the approximation density for the individual draws.

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 = draws)
  ```

  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 if your log-likelihood function takes additional arguments. These arguments are used like the `draws` argument in that they are recycled for each observation.

References

Magnusson, M., Riis Andersen, M., Jonasson, J. and Vehtari, A. (2019). Leave-One-Out Cross-Validation for Large Data. In *International Conference on Machine Learning*

Magnusson, M., Riis Andersen, M., Jonasson, J. and Vehtari, A. (2019). Leave-One-Out Cross-Validation for Model Comparison in Large Data.

See Also

`loo()`, `psis()`, `loo_compare()`

`loo_compare` *Model comparison*

Description

Compare fitted models based on ELPD.

By default the print method shows only the most important information. Use `print(...,simplify=FALSE)` to print a more detailed summary.

Usage

```
loo_compare(x, ...)

## Default S3 method:
loo_compare(x, ...)

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

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

Arguments

`x`	An object of class `"loo"` or a list of such objects.
`...`	Additional objects of class `"loo"`.
`digits`	For the print method only, the number of digits to use when printing.
`simplify`	For the print method only, should only the essential columns of the summary matrix be printed? The entire matrix is always returned, but by default only the most important columns are printed.

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` (or multiplied by -2, if desired, to be on the deviance scale).

When using `loo_compare()`, the returned matrix will have one row per model and several columns of estimates. The values in the `elpd_diff` and `se_diff` columns of the returned matrix are computed by making pairwise comparisons between each model and the model with the largest ELPD (the model in the first row). For this reason the `elpd_diff` column will always have the value `0` in the first row (i.e., the difference between the preferred model and itself) and negative values in subsequent rows for the remaining models.

To compute the standard error of the difference in ELPD — which should not be expected to equal the difference of the standard errors — we 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 matrix with class `"compare.loo"` that has its own print method. See the **Details** section.

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 (journal version, preprint arXiv:1507.04544).

Vehtari, A., Simpson, D., Gelman, A., Yao, Y., and Gabry, J. (2019). Pareto smoothed importance sampling. preprint arXiv:1507.02646

Examples

```
# very artificial example, just for demonstration!
LL <- example_loglik_array()
loo1 <- loo(LL, r_eff = NA)     # should be worst model when compared
loo2 <- loo(LL + 1, r_eff = NA) # should be second best model when compared
loo3 <- loo(LL + 2, r_eff = NA) # should be best model when compared

comp <- loo_compare(loo1, loo2, loo3)
print(comp, digits = 2)

# show more details with simplify=FALSE
# (will be the same for all models in this artificial example)
print(comp, simplify = FALSE, digits = 3)

# can use a list of objects
loo_compare(x = list(loo1, loo2, loo3))

## Not run:
# works for waic (and kfold) too
loo_compare(waic(LL), waic(LL - 10))

## End(Not run)
```

---

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), Vehtari, Gelman, and Gabry (2017), and Vehtari, Simpson, Gelman, Yao, and Gabry (2019) 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" (the default) 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 argument BB is FALSE).
optim_method	If method="stacking", a string passed to the method argument of stats::constrOptim() to specify the optimization algorithm. The default is optim_method="BFGS", but other options are available (see stats::optim()).
optim_control	If method="stacking", a list of control parameters for optimization passed to the control argument of stats::constrOptim().
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 BB_n=1000.
alpha	Positive scalar shape parameter in the Dirichlet distribution used for the Bayesian bootstrap. The default is alpha=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.
  - Note for Windows 10 users: it is **strongly** recommended to avoid using the .Rprofile file to set mc.cores (using the cores argument or setting mc.cores interactively or in a script is fine).

lpd_point
: If calling stacking_weights() or pseudobma_weights() directly, 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"), which is the default for loo_model_weights(), 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 (journal version, preprint arXiv:1507.04544).

Vehtari, A., Simpson, D., Gelman, A., Yao, Y., and Gabry, J. (2019). Pareto smoothed importance sampling. preprint arXiv: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, particularly Bayesian Stacking and Pseudo-BMA weights using the **loo** package.
- `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)
```

`loo_moment_match` *Moment matching for efficient approximate leave-one-out cross-validation (LOO)*

Description

Moment matching algorithm for updating a loo object when Pareto k estimates are large.

Usage

```
loo_moment_match(x, ...)

## Default S3 method:
loo_moment_match(
  x,
  loo,
  post_draws,
  log_lik_i,
  unconstrain_pars,
  log_prob_upars,
  log_lik_i_upars,
  max_iters = 30L,
  k_threshold = 0.7,
  split = TRUE,
  cov = TRUE,
  cores = getOption("mc.cores", 1),
  ...
)
```

Arguments

x	A fitted model object.
...	Further arguments passed to the custom functions documented above.
loo	A loo object to be modified.
post_draws	A function the takes x as the first argument and returns a matrix of posterior draws of the model parameters.
log_lik_i	A function that takes x and i and returns a matrix (one column per chain) or a vector (all chains stacked) of log-likelihood draws of the ith observation based on the model x. If the draws are obtained using MCMC, the matrix with MCMC chains separated is preferred.
unconstrain_pars	A function that takes arguments x, and pars and returns posterior draws on the unconstrained space based on the posterior draws on the constrained space passed via pars.
log_prob_upars	A function that takes arguments x and upars and returns a matrix of log-posterior density values of the unconstrained posterior draws passed via upars.
log_lik_i_upars	A function that takes arguments x, upars, and i and returns a vector of log-likelihood draws of the ith observation based on the unconstrained posterior draws passed via upars.
max_iters	Maximum number of moment matching iterations. Usually this does not need to be modified. If the maximum number of iterations is reached, there will be a warning, and increasing max_iters may improve accuracy.
k_threshold	Threshold value for Pareto k values above which the moment matching algorithm is used. The default value is 0.5.

`split`	Logical; Indicate whether to do the split transformation or not at the end of moment matching for each LOO fold.
`cov`	Logical; Indicate whether to match the covariance matrix of the samples or not. If `FALSE`, only the mean and marginal variances are matched.
`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. • Note for Windows 10 users: it is **strongly** recommended to avoid using the `.Rprofile` file to set `mc.cores` (using the `cores` argument or setting `mc.cores` interactively or in a script is fine).

Details

The `loo_moment_match()` function is an S3 generic and we provide a default method that takes as arguments user-specified functions `post_draws`, `log_lik_i`, `unconstrain_pars`, `log_prob_upars`, and `log_lik_i_upars`. All of these functions should take `...`. as an argument in addition to those specified for each function.

Value

The `loo_moment_match()` methods return an updated `loo` object. The structure of the updated `loo` object is similar, but the method also stores the original Pareto k diagnostic values in the diagnostics field.

Methods (by class)

- `default`: A default method that takes as arguments a user-specified model object `x`, a `loo` object and user-specified functions `post_draws`, `log_lik_i`, `unconstrain_pars`, `log_prob_upars`, and `log_lik_i_upars`.

References

Paananen, T., Piironen, J., Buerkner, P.-C., Vehtari, A. (2020). Implicitly Adaptive Importance Sampling. preprint arXiv:1906.08850

See Also

`loo()`, `loo_moment_match_split()`

Examples

```
# See the vignette for loo_moment_match()
```

`loo_moment_match_split`

Split moment matching for efficient approximate leave-one-out cross-validation (LOO)

Description

A function that computes the split moment matching importance sampling loo. Takes in the moment matching total transformation, transforms only half of the draws, and computes a single elpd using multiple importance sampling.

Usage

```
loo_moment_match_split(
  x,
  upars,
  cov,
  total_shift,
  total_scaling,
  total_mapping,
  i,
  log_prob_upars,
  log_lik_i_upars,
  r_eff_i,
  cores,
  is_method,
  ...
)
```

Arguments

`x`	A fitted model object.
`upars`	A matrix containing the model parameters in unconstrained space where they can have any real value.
`cov`	Logical; Indicate whether to match the covariance matrix of the samples or not. If `FALSE`, only the mean and marginal variances are matched.
`total_shift`	A vector representing the total shift made by the moment matching algorithm.
`total_scaling`	A vector representing the total scaling of marginal variance made by the moment matching algorithm.
`total_mapping`	A vector representing the total covariance transformation made by the moment matching algorithm.
`i`	Observation index.
`log_prob_upars`	A function that takes arguments `x` and `upars` and returns a matrix of log-posterior density values of the unconstrained posterior draws passed via `upars`.

`log_lik_i_upars`
: A function that takes arguments `x`, `upars`, and `i` and returns a vector of log-likeliood draws of the `i`th observation based on the unconstrained posterior draws passed via `upars`.

`r_eff_i`
: MCMC relative effective sample size of the `i`'th log likelihood draws.

`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.

  - Note for Windows 10 users: it is **strongly** recommended to avoid using the `.Rprofile` file to set `mc.cores` (using the `cores` argument or setting `mc.cores` interactively or in a script is fine).

`is_method`
: The importance sampling method to use. The following methods are implemented:

  - `"psis"`: Pareto-Smoothed Importance Sampling (PSIS). Default method.
  - `"tis"`: Truncated Importance Sampling (TIS) with truncation at `sqrt(S)`, where `S` is the number of posterior draws.
  - `"sis"`: Standard Importance Sampling (SIS).

`...`
: Further arguments passed to the custom functions documented above.

Value

A list containing the updated log-importance weights and log-likelihood values. Also returns the updated MCMC effective sample size and the integrand-specific log-importance weights.

References

Paananen, T., Piironen, J., Buerkner, P.-C., Vehtari, A. (2020). Implicitly Adaptive Importance Sampling. preprint arXiv:1906.08850

See Also

`loo()`, `loo_moment_match()`

`loo_subsample` *Efficient approximate leave-one-out cross-validation (LOO) using subsampling*

Description

Efficient approximate leave-one-out cross-validation (LOO) using subsampling

Usage

```
loo_subsample(x, ...)

## S3 method for class '`function`'
loo_subsample(
  x,
  ...,
  data = NULL,
  draws = NULL,
  observations = 400,
  log_p = NULL,
  log_g = NULL,
  r_eff = NULL,
  save_psis = FALSE,
  cores = getOption("mc.cores", 1),
  loo_approximation = "plpd",
  loo_approximation_draws = NULL,
  estimator = "diff_srs",
  llgrad = NULL,
  llhess = NULL
)
```

Arguments

`x`	A function. The **Methods (by class)** section, below, has detailed descriptions of how to specify the inputs.
`data, draws, ...`	For `loo_subsample.function()`, these are the data, posterior draws, and other arguments to pass to the log-likelihood function.
`observations`	The subsample observations to use. The argument can take four (4) types of arguments: • `NULL` to use all observations. The algorithm then just uses standard `loo()` or `loo_approximate_posterior()`. • A single integer to specify the number of observations to be subsampled. • A vector of integers to provide the indices used to subset the data. *These observations need to be subsampled with the same scheme as given by the* `estimator` *argument*. • A `psis_loo_ss` object to use the same observations that were used in a previous call to `loo_subsample()`.
`log_p, log_g`	Should be supplied only if approximate posterior draws are used. The default (`NULL`) indicates draws are from "true" posterior (i.e. using MCMC). If not `NULL` then they should be specified as described in `loo_approximate_posterior()`.
`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 when using posterior draws obtained with MCMC. If MCMC draws are used and `r_eff` is not

provided then the reported PSIS effective sample sizes and Monte Carlo error estimates will be over-optimistic. If the posterior draws are independent then `r_eff=1` and can be omitted. See the `relative_eff()` helper functions for computing `r_eff`.

`save_psis` Should the `"psis"` object created internally by `loo_subsample()` be saved in the returned object? See `loo()` for details.

`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.

- Note for Windows 10 users: it is **strongly** recommended to avoid using the `.Rprofile` file to set `mc.cores` (using the `cores` argument or setting `mc.cores` interactively or in a script is fine).

`loo_approximation`

What type of approximation of the loo_i's should be used? The default is `"plpd"` (the log predictive density using the posterior expectation). There are six different methods implemented to approximate loo_i's (see the references for more details):

- `"plpd"`: uses the lpd based on point estimates (i.e., $p(y_i|\hat{\theta})$).
- `"lpd"`: uses the lpds (i,e., $p(y_i|y)$).
- `"tis"`: uses truncated importance sampling to approximate PSIS-LOO.
- `"waic"`: uses waic (i.e., $p(y_i|y) - p_{waic}$).
- `"waic_grad_marginal"`: uses waic approximation using first order delta method and posterior marginal variances to approximate p_{waic} (ie. $p(y_i|\hat{\theta})$-p_waic_grad_marginal). Requires gradient of likelihood function.
- `"waic_grad"`: uses waic approximation using first order delta method and posterior covariance to approximate p_{waic} (ie. $p(y_i|\hat{\theta})$-p_waic_grad). Requires gradient of likelihood function.
- `"waic_hess"`: uses waic approximation using second order delta method and posterior covariance to approximate p_{waic} (ie. $p(y_i|\hat{\theta})$-p_waic_grad). Requires gradient and Hessian of likelihood function.

As point estimates of $\hat{\theta}$, the posterior expectations of the parameters are used.

`loo_approximation_draws`

The number of posterior draws used when integrating over the posterior. This is used if `loo_approximation` is set to `"lpd"`, `"waic"`, or `"tis"`.

`estimator` How should `elpd_loo`, `p_loo` and `looic` be estimated? The default is `"diff_srs"`.

- `"diff_srs"`: uses the difference estimator with simple random sampling (srs). `p_loo` is estimated using standard srs.
- `"hh"`: uses the Hansen-Hurwitz estimator with sampling proportional to size, where `abs` of loo_approximation is used as size.
- `"srs"`: uses simple random sampling and ordinary estimation.

`llgrad` The gradient of the log-likelihood. This is only used when `loo_approximation` is `"waic_grad"`, `"waic_grad_marginal"`, or `"waic_hess"`. The default is `NULL`.

`llhess` The hessian of the log-likelihood. This is only used with `loo_approximation = "waic_hess"`. The default is `NULL`.

Details

The `loo_subsample()` function is an S3 generic and a methods is currently provided for log-likelihood functions. The implementation works for both MCMC and for posterior approximations where it is possible to compute the log density for the approximation.

Value

`loo_subsample()` returns a named list with class `c("psis_loo_ss","psis_loo","loo")`. This has the same structure as objects returned by `loo()` but with the additional slot:

- `loo_subsampling`: A list with two vectors, `log_p` and `log_g`, of the same length containing the posterior density and the approximation density for the individual draws.

Methods (by class)

- `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 = draws)
  ```

  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 if your log-likelihood function takes additional arguments. These arguments are used like the `draws` argument in that they are recycled for each observation.

References

Magnusson, M., Riis Andersen, M., Jonasson, J. and Vehtari, A. (2019). Leave-One-Out Cross-Validation for Large Data. In *International Conference on Machine Learning*

Magnusson, M., Riis Andersen, M., Jonasson, J. and Vehtari, A. (2019). Leave-One-Out Cross-Validation for Model Comparison in Large Data.

See Also

`loo()`, `psis()`, `loo_compare()`

nobs.psis_loo_ss — *The number of observations in a `psis_loo_ss` object.*

Description

The number of observations in a `psis_loo_ss` object.

Usage

```
## S3 method for class 'psis_loo_ss'
nobs(object, ...)
```

Arguments

`object`	a `psis_loo_ss` object.
`...`	Currently unused.

obs_idx — *Get observation indices used in subsampling*

Description

Get observation indices used in subsampling

Usage

```
obs_idx(x, rep = TRUE)
```

Arguments

`x`	A `psis_loo_ss` object.
`rep`	If sampling with replacement is used, an observation can have multiple samples and these are then repeated in the returned object if `rep=TRUE` (e.g., a vector `c(1,1,2)` indicates that observation 1 has been subampled two times). If `rep=FALSE` only the unique indices are returned.

Value

An integer vector.

`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 (2017) and Vehtari, Simpson, Gelman, Yao, and Gabry (2019) for full details.

Usage

```
pareto_k_table(x)

pareto_k_ids(x, threshold = 0.5)

pareto_k_values(x)

pareto_k_influence_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`). See **Details** for the motivation behind these defaults.

`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 `graphics::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.

What if the estimated tail shape parameter k exceeds 0.5: 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 very different, which is more likely to happen with a non-robust model and highly influential observations. If the estimated tail shape parameter k exceeds 0.5, the user should be warned. (Note: If k is greater than 0.5 then WAIC is also likely to fail, but WAIC lacks its own diagnostic.) In practice, we have observed good performance for values of k up to 0.7. When using PSIS in the context of approximate LOO-CV, we recommend one of the following actions when $k > 0.7$:

- With some additional computations, it is possible to transform the MCMC draws from the posterior distribution to obtain more reliable importance sampling estimates. This results in a smaller shape parameter k. See `loo_moment_match()` for an example of this.
- Sampling directly from $p(\theta^s|y_{-i})$ for the problematic observations i, or using k-fold cross-validation will generally be more stable.
- Using a model that is more robust to anomalous observations will generally make approximate LOO-CV more stable.

Observation influence statistics: The estimated shape parameter k for each observation can be used as a measure of the observation's influence on posterior distribution of the model. These can be obtained with `pareto_k_influence_values()`.

Effective sample size and error estimates: In the case that we obtain the samples from the proposal distribution via MCMC the **loo** package also computes 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**.

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. These represent the reliability of sampling.

`pareto_k_influence_values()` returns a vector of the estimated Pareto k parameters. These represent influence of the observations on the model posterior distribution.

`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 (journal version, preprint arXiv:1507.04544).

Vehtari, A., Simpson, D., Gelman, A., Yao, Y., and Gabry, J. (2019). Pareto smoothed importance sampling. preprint arXiv: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 'importance_sampling_loo'
print(x, digits = 1, plot_k = FALSE, ...)

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

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

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

Arguments

`x`	An object returned by `loo()`, `psis()`, or `waic()`.
`digits`	An integer passed to `base::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 for 'loo' objects.

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, Simpson, Gelman, Yao, and Gabry (2019). 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)

is.psis(x)

is.sis(x)

is.tis(x)
```

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`. If using `psis` with draws of the `log_ratios` not obtained from MCMC then the warning message thrown when not specifying `r_eff` can be disabled by setting `r_eff` to `NA`.

`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.

- Note for Windows 10 users: it is **strongly** recommended to avoid using the `.Rprofile` file to set `mc.cores` (using the `cores` argument or setting `mc.cores` interactively or in a script is fine).

`x` For `is.psis()`, an object to check.

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).

`method` Method used for importance sampling, here `psis`.

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 (journal version, preprint arXiv:1507.04544).

Vehtari, A., Simpson, D., Gelman, A., Yao, Y., and Gabry, J. (2019). Pareto smoothed importance sampling. preprint arXiv: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\` 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 (journal version, preprint arXiv:1507.04544).

Vehtari, A., Simpson, D., Gelman, A., Yao, Y., and Gabry, J. (2019). Pareto smoothed importance sampling. preprint arXiv: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
)

## S3 method for class 'importance_sampling'
relative_eff(x, ...)
```

Arguments

x	A vector, matrix, 3-D array, or function. See the **Methods (by class)** section below for details on specifying x. • For use with the loo() function, the values in x (or generated by x, if 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 = draws)
  ```

  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 ith 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 if your log-likelihood function takes additional arguments. These arguments are used like the draws argument in that they are recycled for each observation.
- importance_sampling: If x is an object of class "psis", relative_eff() simply returns the r_eff attribute of x.

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)
```

---

sis *Standard importance sampling (SIS)*

Description

Implementation of standard importance sampling (SIS).

Usage

```
sis(log_ratios, ...)

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

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

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

Arguments

`log_ratios`	An array, matrix, or vector of importance ratios on the log scale (for Importance sampling 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. See the `relative_eff()` helper function for computing `r_eff`. If using `psis` with draws of the `log_ratios` not obtained from MCMC then the warning message thrown when not specifying `r_eff` can be disabled by setting `r_eff` to `NA`.

`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.

- Note for Windows 10 users: it is **strongly** recommended to avoid using the `.Rprofile` file to set `mc.cores` (using the `cores` argument or setting `mc.cores` interactively or in a script is fine).

Value

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

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

`diagnostics` A named list containing one vector:

- `pareto_k`: Not used in `sis`, all set to 0.
- `n_eff`: effective sample size estimates.

Objects of class `"sis"` 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.

`r_eff` If specified, the user's `r_eff` argument.

`tail_len` Not used for `sis`.

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

`method` Method used for importance sampling, here `sis`.

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 (journal version, preprint arXiv:1507.04544).

Vehtari, A., Simpson, D., Gelman, A., Yao, Y., and Gabry, J. (2019). Pareto smoothed importance sampling. preprint arXiv:1507.02646

See Also

- `psis()` for approximate LOO-CV using PSIS.
- `loo()` for approximate LOO-CV.
- pareto-k-diagnostic for PSIS diagnostics.

Examples

```
log_ratios <- -1 * example_loglik_array()
r_eff <- relative_eff(exp(-log_ratios))
sis_result <- sis(log_ratios, r_eff = r_eff)
str(sis_result)

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

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

---

tis *Truncated importance sampling (TIS)*

Description

Implementation of truncated (self-normalized) importance sampling (TIS), truncated at S^(1/2) as recommended by Ionides (2008).

Usage

```
tis(log_ratios, ...)

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

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

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

Arguments

`log_ratios` An array, matrix, or vector of importance ratios on the log scale (for Importance sampling 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. See the relative_eff() helper function for computing r_eff. If using psis with draws of the log_ratios not obtained from MCMC then the warning message thrown when not specifying r_eff can be disabled by setting r_eff to NA.

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.

- Note for Windows 10 users: it is **strongly** recommended to avoid using the .Rprofile file to set mc.cores (using the cores argument or setting mc.cores interactively or in a script is fine).

Value

The tis() methods return an object of class "tis", 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 tis.

diagnostics A named list containing one vector:

- pareto_k: Not used in tis, all set to 0.
- n_eff: Effective sample size estimates.

Objects of class "tis" 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.

r_eff If specified, the user's r_eff argument.

tail_len Not used for tis.

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

method Method used for importance sampling, here tis.

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

Ionides, Edward L. (2008). Truncated importance sampling. *Journal of Computational and Graphical Statistics* 17(2): 295–311.

See Also

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

Examples

```
log_ratios <- -1 * example_loglik_array()
r_eff <- relative_eff(exp(-log_ratios))
tis_result <- tis(log_ratios, r_eff = r_eff)
str(tis_result)

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

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

---

update.psis_loo_ss   *Update* psis_loo_ss *objects*

Description

Update psis_loo_ss objects

Usage

```
## S3 method for class 'psis_loo_ss'
update(
  object,
  ...,
  data = NULL,
  draws = NULL,
  observations = NULL,
  r_eff = NULL,
  cores = getOption("mc.cores", 1),
  loo_approximation = NULL,
  loo_approximation_draws = NULL,
  llgrad = NULL,
  llhess = NULL
)
```

Arguments

object
: A psis_loo_ss object to update.

...
: Currently not used.

data
: For loo_subsample.function(), these are the data, posterior draws, and other arguments to pass to the log-likelihood function.

draws
: For loo_subsample.function(), these are the data, posterior draws, and other arguments to pass to the log-likelihood function.

observations
: The subsample observations to use. The argument can take four (4) types of arguments:

- NULL to use all observations. The algorithm then just uses standard loo() or loo_approximate_posterior().
- A single integer to specify the number of observations to be subsampled.
- A vector of integers to provide the indices used to subset the data. *These observations need to be subsampled with the same scheme as given by the* estimator *argument.*
- A psis_loo_ss object to use the same observations that were used in a previous call to loo_subsample().

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 when using posterior draws obtained with MCMC. If MCMC draws are used and r_eff is not provided then the reported PSIS effective sample sizes and Monte Carlo error estimates will be over-optimistic. If the posterior draws are independent then r_eff=1 and can be omitted. See the relative_eff() helper functions 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.

- Note for Windows 10 users: it is **strongly** recommended to avoid using the .Rprofile file to set mc.cores (using the cores argument or setting mc.cores interactively or in a script is fine).

loo_approximation
: What type of approximation of the loo_i's should be used? The default is "plpd" (the log predictive density using the posterior expectation). There are six different methods implemented to approximate loo_i's (see the references for more details):

- "plpd": uses the lpd based on point estimates (i.e., $p(y_i|\hat{\theta})$).
- "lpd": uses the lpds (i,e., $p(y_i|y)$).
- "tis": uses truncated importance sampling to approximate PSIS-LOO.
- "waic": uses waic (i.e., $p(y_i|y) - p_{waic}$).

- "waic_grad_marginal": uses waic approximation using first order delta method and posterior marginal variances to approximate p_{waic} (ie. $p(y_i|\hat{\theta})$-p_waic_grad_marginal). Requires gradient of likelihood function.
- "waic_grad": uses waic approximation using first order delta method and posterior covariance to approximate p_{waic} (ie. $p(y_i|\hat{\theta})$-p_waic_grad). Requires gradient of likelihood function.
- "waic_hess": uses waic approximation using second order delta method and posterior covariance to approximate p_{waic} (ie. $p(y_i|\hat{\theta})$-p_waic_grad). Requires gradient and Hessian of likelihood function.

As point estimates of $\hat{\theta}$, the posterior expectations of the parameters are used.

loo_approximation_draws
: The number of posterior draws used when integrating over the posterior. This is used if loo_approximation is set to "lpd", "waic", or "tis".

llgrad
: The gradient of the log-likelihood. This is only used when loo_approximation is "waic_grad", "waic_grad_marginal", or "waic_hess". The default is NULL.

llhess
: The hessian of the log-likelihood. This is only used with loo_approximation = "waic_hess". The default is NULL.

Details

If observations is updated then if a vector of indices or a psis_loo_ss object is supplied the updated object will have exactly the observations indicated by the vector or psis_loo_ss object. If a single integer is supplied, new observations will be sampled to reach the supplied sample size.

Value

A psis_loo_ss object.

---

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 as well as 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)

is.waic(x)
```

Arguments

x — A log-likelihood array, matrix, or function. The **Methods (by class)** section, below, has detailed descriptions 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 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 = draws)
  ```

  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 ith 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 if your log-likelihood function takes additional arguments. These arguments are used like the draws argument in that they are recycled for each observation.

References

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.

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 (journal version, preprint arXiv:1507.04544).

Vehtari, A., Simpson, D., Gelman, A., Yao, Y., and Gabry, J. (2019). Pareto smoothed importance sampling. preprint arXiv:1507.02646

See Also

- The **loo** package vignettes and Vehtari, Gelman, and Gabry (2017) and Vehtari, Simpson, Gelman, Yao, and Gabry (2019) for more details on why we prefer `loo()` to `waic()`.
- `loo_compare()` for comparing models on approximate LOO-CV 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)
```

`weights.importance_sampling` *Extract importance sampling weights*

Description

Extract importance sampling weights

Usage

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

Arguments

`object`	An object returned by `psis()`, `tis()`, or `sis()`.
`...`	Ignored.
`log`	Should the weights be returned on the log scale? Defaults to `TRUE`.
`normalize`	Should the weights be normalized? Defaults to `TRUE`.

Value

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

Examples

```
# See the examples at help("psis")
```

Index