Package 'baggr'

February 28, 2020

Type Package

Title Bayesian Aggregate Treatment Effects

Version 0.4.0

Maintainer Witold Wiecek <witold.wiecek@gmail.com>

Description Running and comparing meta-analyses of data with hierarchical Bayesian models in Stan, including convenience functions for formatting data, plotting and pooling measures specific to meta-analysis.

License GPL (>= 3)

Encoding UTF-8

LazyData true

ByteCompile true

Depends R (>= 3.5.0), Rcpp (>= 0.12.17), methods

Imports rstan (>= 2.18.1), rstantools (>= 1.5.0), bayesplot, crayon, forestplot, ggplot2, gridExtra, utils, stats, testthat

LinkingTo StanHeaders (>= 2.18.1), rstan (>= 2.18.1), BH (>= 1.66.0-1), Rcpp (>= 0.12.17), RcppEigen (>= 0.3.3.4.0)

SystemRequirements GNU make

NeedsCompilation yes

RoxygenNote 7.0.2

Suggests knitr, covr

VignetteBuilder knitr

URL https://github.com/wwiecek/baggr

BugReports https://github.com/wwiecek/baggr/issues

Language en-GB

Author Witold Wiecek [cre, aut],
Rachael Meager [aut],
Brice Green [ctb] (predict(), loo_compare, many visuals),
Trustees of Columbia University [cph] (tools/make_cc.R)

Repository CRAN

Date/Publication 2020-02-28 11:20:09 UTC

R topics documented:

`baggr-package` *baggr - a package for Bayesian meta-analysis*

Description

This is *baggr* (pronounced as *bagger* or *badger*), a Bayesian meta-analysis package for R using Stan. *Baggr* is intended to be user-friendly and transparent so that it's easier to understand the models you are building and criticise them.

Details

Baggr package provides a suite of models that work with both summary data and full data sets, to synthesise evidence collected from different groups, contexts or time periods. The baggr command automatically detects the data type and, by default, fits a partial pooling model (which you may know as random effects models) with weakly informative priors by calling Stan to carry out Bayesian inference. Modelling of variances or quantiles, standardisation and transformation of data is also possible.

Getting help

This is only a simple package help file. For documentation of the main function for conducting analyses see baggr. For description of models, data types and priors available in the package, try the built-in vignette (`vignette("baggr")`).

`baggr` *Bayesian aggregate treatment effects model*

Description

Bayesian inference on parameters of an average treatment effects model that's appropriate to the supplied individual- or group-level data, using Hamiltonian Monte Carlo in Stan. (For overall package help file see baggr-package)

Usage

```
baggr(
  data,
  model = NULL,
  pooling = "partial",
  effect = NULL,
  covariates = c(),
  prior_hypermean = NULL,
  prior_hypersd = NULL,
  prior_hypercor = NULL,
  prior_beta = NULL,
```

```
  prior = NULL,
  ppd = FALSE,
  test_data = NULL,
  quantiles = seq(0.05, 0.95, 0.1),
  outcome = "outcome",
  group = "group",
  treatment = "treatment",
  silent = FALSE,
  warn = TRUE,
  ...
)
```

Arguments

`data`	data frame with summary or individual level data to meta-analyse
`model`	if `NULL`, detected automatically from input data otherwise choose from `"rubin"`, `"mutau"`, `"individual"`, `"quantiles"` (see Details).
`pooling`	Type of pooling; choose from `"none"`, `"partial"` (default) and `"full"`. If you are not familiar with the terms, consult the vignette; "partial" can be understood as random effects and "full" as fixed effects
`effect`	Label for effect. Will default to "mean" in most cases, "log OR" in logistic model, quantiles in `quantiles` model etc. These labels are used in various print and plot outputs. Comparable models (e.g. in baggr_compare) should have same `effect`.
`covariates`	Character vector with column names in `data`. The corresponding columns are used as covariates (fixed effects) in the meta-regression model (in case of aggregate data). In the case of individual level data the model does not differentiate between group-level variables (same values of the covariate for all rows related to a given group) and individual-level covariates.
`prior_hypermean`	prior distribution for hypermean; you can use "plain text" notation like `prior_hypermean=normal(0,100` or `uniform(-10,10)`. See Details:Priors below for more possible specifications. If unspecified, the priors will be derived automatically based on data (and printed out in the console).
`prior_hypersd`	prior for hyper-standard deviation, used by Rubin and "mutau"`` models; same rules apply as for _hypermean‘;
`prior_hypercor`	prior for hypercorrelation matrix, used by the `"mutau"` model
`prior_beta`	prior for regression coefficients if `covariates` are specified; will default to experimental normal(0, 10^2) distribution
`prior`	alternative way to specify all priors as a named list with `hypermean`, `hypersd`, `hypercor`, `beta`, analogous to `prior_` arguments above, e.g. `prior = list(hypermean = normal(0,10),beta = uniform(-50,50))`
`ppd`	logical; use prior predictive distribution? (*p.p.d.*) Default is no. If `ppd=TRUE`, Stan model will sample from the prior distributions and ignore `data` in inference. However, `data` argument might still be used to infer the correct model and to set the default priors.

test_data	data for cross-validation; NULL for no validation, otherwise a data frame with the same columns as data argument
quantiles	if model = "quantiles", a vector indicating which quantiles of data to use (with values between 0 and 1)
outcome	character; column name in (individual-level) data with outcome variable values
group	character; column name in data with grouping factor; it's necessary for individual-level data, for summarised data it will be used as labels for groups when displaying results
treatment	character; column name in (individual-level) data with treatment factor;
silent	Whether to silence messages about prior settings and about other automatic behaviour.
warn	print an additional warning if Rhat exceeds 1.05
...	extra options passed to Stan function, e.g. control = list(adapt_delta = 0.99), number of iterations etc.

Details

Running baggr requires 1/ data preparation, 2/ choice of model, 3/ choice of priors. All three are discussed in depth in the package vignette (vignette("baggr")).

Data. For aggregate data models you need a data frame with columns tau and se or tau, mu, se.tau, se.mu. An additional column can be used to provide labels for each group (by default column group is used if available, but this can be customised – see the example below). For individual level data three columns are needed: outcome, treatment, group. These are identified by using the outcome, treatment and group arguments.

Many data preparation steps can be done through a helper function prepare_ma. It can convert individual to summary-level data, calculate odds/risk ratios (with/without corrections) in binary data, standardise variables and more. Using it will automatically format data inputs to work with baggr().

Models. Available models are:

- for the **continuous variable** means: "rubin" model for average treatment effect, "mutau" version which takes into account means of control groups, "full", which works with individual-level data
- for **continuous variable quantiles**: ‘"quantiles"‘ model (see Meager, 2019 in references)
- for **binary data**: "logit" model can be used on individual-level data; you can also analyse continuous statistics such as log odds ratios and logs risk ratios using the models listed above; see vignette("baggr_binary") for tutorial with examples

If no model is specified, the function tries to infer the appropriate model automatically. Additionally, the user must specify type of pooling. The default is always partial pooling.

Covariates. Both aggregate and individual-level data can include extra columns, given by covariates argument (specified as a character vector of column names) to be used in regression models. We also refer to impact of these covariates as *fixed effects*.

Two types of covariates may be present in your data:

- In "rubin" and "mutau" models, covariates that **change according to group unit**. In that case, the model accounting for the group covariates is a meta-regression model. It can be modelled on summary-level data.
- In "logit" and "full" models, covariates that **change according to individual unit**. Then, the model can be called a mixed model . It has to be fitted to individual-level data. Note that the first case can also be accounted for by using a mixed model.

Priors. It is optional to specify priors yourself, as the package will try propose an appropriate prior for the input data if you do not pass a prior argument. To set the priors yourself, use prior_ arguments. For specifying many priors at once (or re-using between models), a single prior = list(...) argument can be used instead. Appropriate examples are given in vignette("baggr").

Outputs. By default, some outputs are printed. There is also a plot method for *baggr* objects which you can access via baggr_plot (or simply plot()). Other standard functions for working with baggr object are

- treatment_effect for distribution of hyperparameters
- group_effects for distributions of group-specific parameters
- fixed_effects for coefficients in (meta-)regression
- effect_draw and effect_plot for posterior predictive distributions
- baggr_compare for comparing multiple baggr models
- loocv for cross-validation
- pp_check for posterior predictive checks

Value

baggr class structure: a list including Stan model fit alongside input data, pooling metrics, various model properties. If test data is used, mean value of -2*lpd is reported as mean_lpd

Author(s)

Witold Wiecek, Rachael Meager

Examples

```
df_pooled <- data.frame("tau" = c(1, -1, .5, -.5, .7, -.7, 1.3, -1.3),
                        "se" = rep(1, 8),
                        "state" = datasets::state.name[1:8])
baggr(df_pooled) #baggr automatically detects the input data
# same model, but with correct labels,
# different pooling & passing some options to Stan
baggr(df_pooled, group = "state", pooling = "full", iter = 500)
# model with different (very informative) priors
baggr(df_pooled, prior_hypersd = normal(0, 2))


# "mu & tau" model, using a built-in dataset
# prepare_ma() can summarise individual-level data
ms <- microcredit_simplified
ms$outcome <- microcredit_simplified$consumerdurables + 1
```

```
microcredit_summary_data <- prepare_ma(ms)
baggr(microcredit_summary_data, model = "mutau",
      pooling = "partial", prior_hypercor = lkj(1),
      prior_hypersd = normal(0,10),
      prior_hypermean = multinormal(c(0,0),matrix(c(10,3,3,10),2,2)))
```

`baggr_compare` *(Run and) compare multiple baggr models*

Description

Compare multiple baggr models by either providing multiple already existing models as (named) arguments or passing parameters necessary to run a baggr model.

Usage

```
baggr_compare(..., what = "pooling", compare = "groups", transform = NULL)
```

Arguments

`...`	Either some (at least 1) objects of class `baggr` (you should name your objects, see the example below) or the same arguments you'd pass to baggr. In the latter case you must specify `what` to compare.
`what`	One of `"pooling"` (comparison between no, partial and full pooling) or `"prior"` (comparison between prior and posterior predictive). If pre-existing baggr models are passed to `...`, this argument is ignored.
`compare`	When plotting, choose between comparison of `"groups"` (default) or (hyper-) `"effects"`. The former is not available when `what = "prior"`.
`transform`	a function (e.g. exp(), log()) to apply to the values of group (and hyper, if hyper=TRUE) effects before plotting; when working with effects that are on log scale, exponent transform is used automatically, you can plot on log scale by setting transform = identity

Details

If you pass parameters to the function you must specify what kind of comparison you want, either "pooling" which will run fully/partially/un-pooled models and compare them or "prior" which will generate estimates without the data and compare them to the model with the full data. For more details see baggr, specifically the PPD argument.

Value

an object of class `baggr_compare`

Author(s)

Witold Wiecek, Brice Green

See Also

plot.baggr_compare and print.baggr_compare for working with results of this function

Examples

```
# Most basic comparison between no, partial and full pooling
# (This will run the models)

# run model with just prior and then full data for comparison
# with the same arguments that are passed to baggr
prior_comparison <-
    baggr_compare(schools,
                  model = 'rubin',
                  prior_hypermean = normal(0, 3),
                  prior_hypersd = normal(0,2),
                  prior_hypercor = lkj(2),
                  what = "prior")

# print the aggregated treatment effects
prior_comparison

# plot the comparison of the two distributions
plot(prior_comparison)

# Now compare different types of pooling for the same model
pooling_comparison <-
   baggr_compare(schools,
                 model = 'rubin',
                 prior_hypermean = normal(0, 3),
                 prior_hypersd = normal(0,2),
                 prior_hypercor = lkj(2),
                 what = "pooling")

# plot this comparison
plot(pooling_comparison)

# Compare existing models:
bg1 <- baggr(schools, pooling = "partial")
bg2 <- baggr(schools, pooling = "full")
baggr_compare("Partial pooling model" = bg1, "Full pooling" = bg2,
              arrange = "grid")

#' ...or simply draw prior predictive dist (note ppd=T)
bg1 <- baggr(schools, ppd=T)
bg2 <- baggr(schools, prior_hypermean = normal(0, 5), ppd=T)
baggr_compare("Prior A, p.p.d."=bg1,
              "Prior B p.p.d."=bg2,
```

```
              compare = "effects")

# Compare posterior effects as a function of priors (note ppd=F)
bg1 <- baggr(schools, prior_hypersd = uniform(0, 20))
bg2 <- baggr(schools, prior_hypersd = normal(0, 5))
baggr_compare("Uniform prior on SD"=bg1,
              "Normal prior on SD"=bg2,
              compare = "effects")
# You can also compare different subsets of input data
bg1_small <- baggr(schools[1:6,], pooling = "partial")
baggr_compare("8 schools model" = bg1, "First 6 schools" = bg1_small)
```

`baggr_plot` *Plotting method in baggr package*

Description

Extracts study effects from the `baggr` model and sends them to one of `bayesplot` package plotting functions.

Usage

```
baggr_plot(
  bg,
  hyper = FALSE,
  style = "intervals",
  transform = NULL,
  prob = 0.5,
  prob_outer = 0.95,
  vline = TRUE,
  order = TRUE,
  ...
)
```

Arguments

`bg`	object of class `baggr`
`hyper`	logical; show hypereffect as the last row of the plot?
`style`	one of `areas`, `intervals`
`transform`	a function (e.g. `exp()`, `log()`) to apply to the values of group (and hyper, if `hyper=TRUE`) effects before plotting; when working with effects that are on log scale, exponent transform is used automatically, you can plot on log scale by setting `transform = identity`
`prob`	Probability mass for the inner interval in visualisation
`prob_outer`	Probability mass for the outer interval in visualisation

`vline`	logical; show vertical line through 0 in the plot?
`order`	logical; sort groups by magnitude of treatment effect?
`...`	extra arguments to pass to the `bayesplot` functions

Value

ggplot2 object

Author(s)

Witold Wiecek, Rachael Meager

See Also

bayesplot::MCMC-intervals for more information about *bayesplot* functionality; forest_plot for a typical meta-analysis alternative; effect_plot for plotting treatment effects for a new group

Examples

```
fit <- baggr(schools, pooling = "none")
plot(fit)
plot(fit, style = "areas", order = FALSE)
```

---

baggr_theme_set *Set, get, and replace themes for baggr plots*

Description

These functions get, set, and modify the ggplot2 themes of the baggr plots. `baggr_theme_get()` returns a ggplot2 theme function for adding themes to a plot. `baggr_theme_set()` assigns a new theme for all plots of baggr objects. `baggr_theme_update()` edits a specific theme element for the current theme while holding the theme's other aspects constant. `baggr_theme_replace()` is used for wholesale replacing aspects of a plot's theme (see `ggplot2::theme_get()`).

Usage

```
baggr_theme_set(new = bayesplot::theme_default())

baggr_theme_get()

baggr_theme_update(...)

baggr_theme_replace(...)
```

Arguments

`new`	New theme to use for all baggr plots
`...`	A named list of theme settings

Details

Under the hood, many of the visualizations rely on the bayesplot package, and thus these leverage the `bayesplot::bayesplot_theme_get()` functions. By default, these match the bayesplot's package theme to make it easier to form cohesive graphs across this package and others. The trickiest of these to use is `baggr_theme_replace`; 9 times out of 10 you want baggr_theme_update.

Value

The get method returns the current theme, but all of the others invisibly return the old theme.

See Also

bayesplot::bayesplot_theme_get

Examples

```
# make plot look like default ggplots

library(ggplot2)

fit <- baggr(schools)
baggr_theme_set(theme_grey())
baggr_plot(fit)

# use baggr_theme_get to return theme elements for current theme
qplot(mtcars$mpg) + baggr_theme_get()

# update specific aspect of theme you are interested in
baggr_theme_update(text = element_text(family = "mono"))

# undo that silliness
baggr_theme_update(text = element_text(family = "serif"))

# update and replace are similar, but replace overwrites the
# whole element, update just edits the aspect of the element
# that you give it
# this will error:
# baggr_theme_replace(text = element_text(family = "Times"))
# baggr_plot(fit)
# because it deleted everything else to do with text elements
```

`convert_inputs` *Convert inputs for baggr models*

Description

Converts data to Stan inputs, checks integrity of data and suggests default model if needed. Typically all of this is done automatically by baggr, **this function is only for debugging** or running models "by hand".

Usage

```
convert_inputs(
  data,
  model,
  quantiles,
  group = "group",
  outcome = "outcome",
  treatment = "treatment",
  covariates = c(),
  test_data = NULL,
  silent = FALSE
)
```

Arguments

`data`	'data.frame" with desired modelling input
`model`	valid model name used by baggr; see baggr for allowed models if `model = NULL`, this function will try to find appropriate model automatically
`quantiles`	vector of quantiles to use (only applicable if `model = "quantiles"`)
`group`	name of the column with grouping variable
`outcome`	name of column with outcome variable (designated as string)
`treatment`	name of column with treatment variable
`covariates`	Character vector with column names in `data`. The corresponding columns are used as covariates (fixed effects) in the meta-regression model.
`test_data`	same format as `data` argument, gets left aside for testing purposes (see baggr)
`silent`	Whether to print messages when evaluated

Details

Typically this function is only called within baggr and you do not need to use it yourself. It can be useful to understand inputs or to run models which you modified yourself.

Value

R structure that's appropriate for use by baggr Stan models; `group_label`, `model` and `n_groups` are included as attributes and are necessary for baggr to work correctly

Author(s)

Witold Wiecek

Examples

```
# simple meta-analysis example,
# this is the formatted input for Stan models in baggr():
convert_inputs(schools, "rubin")
```

`effect_draw` *Make posterior draws for treatment effect*

Description

This function takes the samples of hyperparameters of a `baggr` model (commonly hypermean tau and hyper-SD sigma_tau) and simulates values of new realisations of tau (a mean effect in some unobserved group).

Usage

```
effect_draw(x, n, transform = NULL)
```

Arguments

`x`	A `baggr` class object.
`n`	How many values to draw? The default is the same as number of samples in the model (default is 2,000).
`transform`	a transformation to apply to the result, should be an R function; (this is commonly used when calling `group_effects` from other plotting or printing functions)

Value

A vector of possible values of the treatment effect.

`effect_plot` *Plot posterior distribution for treatment effect*

Description

This function plots the effect_draw for one or more baggr objects.

Usage

```
effect_plot(..., transform = NULL)
```

Arguments

...	Object(s) of class baggr. If there is more than one, the names of objects will be used as a plot legend (see example).
transform	a transformation to apply to the result, should be an R function; (this is commonly used when calling group_effects from other plotting or printing functions)

Value

A ggplot.

See Also

baggr_compare can be used as a shortcut for effect_plot with argument compare = "effects"

Examples

```
# A single effects plot
bg1 <- baggr(schools, prior_hypersd = uniform(0, 20))
effect_plot(bg1)

# Compare how posterior depends on the prior choice
bg2 <- baggr(schools, prior_hypersd = normal(0, 5))
effect_plot("Uniform prior on SD"=bg1,
            "Normal prior on SD"=bg2)

# Compare the priors themselves (ppd=T)
bg1_ppd <- baggr(schools, prior_hypersd = uniform(0, 20), ppd=TRUE)
bg2_ppd <- baggr(schools, prior_hypersd = normal(0, 5), ppd=TRUE)
effect_plot("Uniform prior on SD"=bg1_ppd,
            "Normal prior on SD"=bg2_ppd)
```

---

fixed_effects *Effects of covariates on outcome in baggr models*

Description

Effects of covariates on outcome in baggr models

Usage

```
fixed_effects(bg, summary = FALSE, transform = NULL, interval = 0.95)
```

Arguments

bg	a baggr model
summary	logical; if TRUE returns summary statistic instead of all MCMC samples
transform	a transformation (R function) to apply to the result; (this is commonly used when calling from other plotting or printing functions)
interval	uncertainty interval width (numeric between 0 and 1), if summary=TRUE

Value

A list with 2 vectors (corresponding to MCMC samples) tau (mean effect) and sigma_tau (SD). If summary=TRUE, both vectors are summarised as mean and lower/upper bounds according to interval

See Also

treatment_effect for overall treatment effect across groups, group_effects for effects within each group, effect_draw and effect_plot for predicted treatment effect in new group

---

forest_plot *Draw a forest plot for a baggr model*

Description

The forest plot functionality in *baggr* is a simple interface for calling the forestplot function. By default the forest plot displays raw (unpooled) estimates for groups and the treatment effect estimate underneath. This behaviour can be modified to display pooled group estimates.

Usage

```
forest_plot(
  bg,
  show = c("inputs", "posterior", "both", "covariates"),
  print = show,
  prob = 0.95,
  digits = 3,
  ...
)
```

Arguments

bg	a baggr class object
show	if "inputs", then plotted points and lines correspond to raw inputs for each group; if "posterior" – to posterior distribution; you can also plot "both" inputs and posteriors; if "covariates", then fixed effect coefficients are plotted
print	which values to print next to the plot: values of "inputs" or "posterior" means? (if show="covariates", it must be "posterior")

prob	width of the intervals (lines) for the plot
digits	number of digits to display when printing out mean and SD in the plot
...	other arguments passed to forestplot

See Also

forestplot function and its associated vignette for examples; effect_plot and baggr_plot for non-forest plots of baggr results

Examples

```
bg <- baggr(schools, iter = 500)
forest_plot(bg)
forest_plot(bg, show = "posterior", print = "inputs", digits = 2)
```

---

get_n_samples *Extract number of samples from a baggr object*

Description

Extract number of samples from a baggr object

Usage

```
get_n_samples(x)
```

Arguments

x	baggr fit to get samples from

Details

Checks for number of iterations and number of Markov chains, returns maximum number of valid samples

`group_effects` *Extract baggr study effects*

Description

Given a baggr object, returns the raw MCMC draws of the posterior for each group's effect, or a summary of these draws. This is an internal function currently used as a helper for plotting and printing of results.

Usage

```
group_effects(bg, summary = FALSE, transform = NULL, interval = 0.95)
```

Arguments

`bg`	baggr object
`summary`	logical; if TRUE returns summary statistics as explained below.
`transform`	a transformation to apply to the result, should be an R function; (this is commonly used when calling `group_effects` from other plotting or printing functions)
`interval`	uncertainty interval width (numeric between 0 and 1), if summarising

Details

If summary = TRUE, the returned object contains, for each study or group, the following 5 values: the posterior medians, the lower and upper bounds of the uncertainty intervals using the central posterior credible interval of width specified in the argument `interval`, the posterior mean, and the posterior standard deviation.

Value

Either a matrix with MCMC samples (if summary = FALSE) or a summary of these samples (if summary = TRUE).

Examples

```
fit1 <- baggr(schools)
group_effects(fit1, summary = TRUE, interval = 0.5)
```

is.baggr_cv *Check if something is a baggr_cv object*

Description

Check if something is a baggr_cv object

Usage

```
is.baggr_cv(x)
```

Arguments

x	object to check

loocv *Leave one group out cross-validation for* baggr *models*

Description

Performs exact leave-one-group-out cross-validation on a baggr model.

Usage

```
loocv(data, return_models = FALSE, ...)
```

Arguments

data	Input data frame - same as for baggr function.
return_models	logical; if FALSE, summary statistics will be returned and the models discarded; if TRUE, a list of models will be returned alongside summaries
...	Additional arguments passed to baggr.

Details

The values returned by `loocv()` can be used to understand how any one group affects the overall result, as well as how well the model predicts the omitted group.

This function automatically runs K baggr models, leaving out one group at a time, and then calculates expected log predictive density (ELPD) for that group (see Gelman et al 2013). The main output is the cross-validation information criterion, or -2 times the ELPD averaged over 'K' models. This is related to, and often approximated by, the Watanabe-Akaike Information Criterion. A value closer to zero (i.e. a smaller number in magnitude) means a better fit. For more information on cross-validation see this overview article

For running more computation-intensive models, consider setting the `mc.cores` option before running loocv, e.g. `options(mc.cores = 4)` (by default baggr runs 4 MCMC chains in parallel). As a default, rstan runs "silently" (`refresh=0`). To see sampling progress, please set e.g. `loocv(data,refresh = 500)`.

Value

log predictive density value, an object of class baggr_cv; full model, prior values and *lpd* of each model are also returned. These can be examined by using `attributes()` function.

Author(s)

Witold Wiecek

References

Gelman, Andrew, Jessica Hwang, and Aki Vehtari. "Understanding Predictive Information Criteria for Bayesian Models." Statistics and Computing 24, no. 6 (November 2014): 997–1016. https://doi.org/10.1007/s11222-013-9416-2.

Examples

```
# even simple examples may take a while
cv <- loocv(schools, pooling = "partial")
print(cv)      # returns the lpd value
attributes(cv) # more information is included in the object
```

---

loo_compare *Compare fitted models on loo*

Description

Compare fitted models on loo

Usage

```
loo_compare(x, ...)
```

Arguments

x	An object of class baggr_cv or a list of such objects.
...	Additional objects of class "baggr_cv"

Examples

```
# 2 models with more/less informative priors
cv_1 <- loocv(schools, model = "rubin", pooling = "partial")
cv_2 <- loocv(schools, model = "rubin", pooling = "partial",
              prior_hypermean = normal(0, 5), prior_hypersd = cauchy(0,4))
loo_compare(cv_1, cv_2)
```

microcredit *7 studies on effect of microcredit supply*

Description

This dataframe contains the data used in Meager (2019) to estimate hierarchical models on the data from 7 randomized controlled trials of expanding access to microcredit.

Usage

```
microcredit
```

Format

A data frame with 40267 rows, 7 study identifiers and 7 outcomes

Details

The columns include the group indicator which gives the name of the lead author on each of the respective studies, the value of the 6 outcome variables of most interest (consumer durables spending, business expenditures, business profit, business revenues, temptation goods spending and consumption spending) all of which are standardised to USD PPP in 2009 dollars per two weeks (these are flow variables), and finally a treatment assignment status indicator.

The dataset has not otherwise been cleaned and therefore includes NAs and other issues common to real-world datasets.

For more information on how and why these variables were chosen and standardised, see Meager (2019) or consult the associated code repository which includes the standardisation scripts: link

References

Meager, Rachael (2019) Understanding the average impact of microcredit expansions: A Bayesian hierarchical analysis of seven randomized experiments. American Economic Journal: Applied Economics, 11(1), 57-91.

microcredit_simplified *Simplified version of the microcredit dataset.*

Description

This dataframe contains the data used in Meager (2019) to estimate hierarchical models on the data from 7 randomized controlled trials of expanding access to microcredit.

Usage

```
microcredit_simplified
```

Format

A data frame with 14224 rows, 7 study identifiers and 1 outcome

Details

The columns include the group indicator which gives the name of the lead author on each of the respective studies, the value of the household consumer durables spending standardised to USD PPP in 2009 dollars per two weeks (these are flow variables), and finally a treatment assignment status indicator.

The dataset has not otherwise been cleaned and therefore includes NAs and other issues common to real data.

For more information on how and why these variables were chosen and standardised, see Meager (2019) or consult the associated code repository: link

This dataset includes only complete cases and only the consumer durables outcome variable.

References

Meager, Rachael (2019) Understanding the average impact of microcredit expansions: A Bayesian hierarchical analysis of seven randomized experiments. American Economic Journal: Applied Economics, 11(1), 57-91.

`mint` *"Mean and interval" function, including other summaries, calculated for matrix (by column) or vector*

Description

This function is just a convenient shorthand for getting typical summary statistics.

Usage

```
mint(y, int = 0.95, digits = NULL, median = FALSE, sd = FALSE)
```

Arguments

`y`	matrix or a vector; for matrices, `mint` is done by-column
`int`	probability interval (default is 95 percent) to calculate
`digits`	number of significant digits to round values by.
`median`	return median value?
`sd`	return SD?

Examples

```
mint(rnorm(100, 12, 5))
```

plot.baggr *Generic plot for baggr package*

Description

Using generic `plot()` on `baggr` output invokes `baggr_plot` visual. See therein for customisation options. Note that plot output is `ggplot2` object.‘

Usage

```
## S3 method for class 'baggr'
plot(x, ...)
```

Arguments

`x`	object of class `baggr`
`...`	optional arguments, see `baggr_plot`

Value

ggplot2 object from `baggr_plot`

Author(s)

Witold Wiecek

plot.baggr_compare *Plot method for baggr_compare models*

Description

Allows plots that compare multiple baggr models that were passed for comparison purposes to baggr compare or run automatically by baggr_compare

Usage

```
## S3 method for class 'baggr_compare'
plot(
  x,
  style = "areas",
  arrange = "single",
  interval = 0.95,
  hyper = T,
  transform = NULL,
  order = F,
  ...
)
```

Arguments

x	baggr_compare model to plot
style	What kind of plot to display (if arrange = "grid"), passed to the style argument in baggr_plot.
arrange	If "single" (default), generate a single comparison plot; if "grid", display multiple plots side-by-side.
interval	probability level used for display of posterior interval
hyper	Whether to plot pooled treatment effect in addition to group treatment effects
transform	a function (e.g. exp(), log()) to apply to the values of group (and hyper, if hyper=TRUE) effects before plotting; when working with effects that are on log scale, exponent transform is used automatically, you can plot on log scale by setting transform = identity
order	Whether to order by median treatment effect by group. If not, this sorts group alphabetically. The pooled estimate is always listed first, when applicable.
...	ignored for now, may be used in the future

---

pooling *Pooling metrics for baggr*

Description

Compute statistics relating to heterogeneity (whole model) and pooling (for each group) given a baggr meta-analysis model. The statistics are the pooling metric by Gelman & Pardoe (2006) or its complement, the *I-squared* statistic.

Usage

```
pooling(bg, type = c("groups", "total"), summary = TRUE)

heterogeneity(bg, summary = TRUE)
```

Arguments

bg	output of a baggr() function
type	In pooling calculation is done for each of the "groups" (default) or for "total" hypereffect(s). See Details section for how calculation is done.
summary	logical; if FALSE a whole vector of pooling values is returned, otherwise only the means and intervals

Details

Pooling statistic describes the extent to which group-level estimates of treatment effect are "pooled" (or pulled!) toward average treatment effect in the meta-analysis model. If `pooling = "none"` or "full" in baggr, then the returned values are always 0 or 1, respectively. If `pooling = "partial"`, the value is somewhere between 0 and 1.

Formulae for the calculations below are provided in main package vignette. See vignette("baggr").

#' **Estimate of pooling in a group**: this is the calculation done by `pooling()` if `type = "groups"` (default).

In a partial pooling model (see baggr), group k (e.g. study) has a treatment effect estimate, with some SE around the real treatment effect (TE). Each TE itself is distributed with mean and variance.

The quantity of interest is ratio of variability in τ to total variability. By convention, we subtract it from 1, to obtain a *pooling metric p*.

$$p = 1 - (\sigma(\tau)^2/(\sigma_(\tau)^2 + se_k^2))$$

- If $p < 0.5$, that means the variation across studies is higher than variation within studies.
- Values close to 1 indicate nearly full pooling. Variation across studies dominates.
- Values close to 0 – no pooling. Variation within studies dominates.

Note that, since σ_τ^2 is a Bayesian parameter (rather than a single fixed value) p is also a parameter. It is typical for p to have very high dispersion, as in many cases we cannot precisely estimate σ_τ. To obtain the whole distribution of_p_ (rather than summarised values), set `summary=FALSE`.

Overall pooling (in the model)

Typically it is a single measure of heterogeneity that is of interest to researchers. This is calculated by setting `type = "total"` or simply writing `heterogeneity(mymodel)`

In many contexts, i.e. medical statistics, it is typical to report *I-P*, called I^2 (see Higgins *et al*, 2003). Higher values of *I-squared* indicate higher heterogeneity. Von Hippel (2015) provides useful details for *I-squared* calculations.

Same as for group-specific estimates, *P* is a Bayesian parameter and its dispersion can be high.

Relationship to R-squared statistic

See Gelman & Pardoe (2006) Section 1.1 for a short explanation of how R^2 statistic relates to the pooling metric.

Value

Matrix with mean and intervals for chosen pooling metric, each row corresponding to one meta-analysis group.

References

Gelman, Andrew, and Iain Pardoe. "Bayesian Measures of Explained Variance and Pooling in Multilevel (Hierarchical) Models." *Technometrics 48, no. 2 (May 2006): 241-51*. https://doi.org/10.1198/004017005000000517.

Higgins, Julian P T, Simon G Thompson, Jonathan J Deeks, and Douglas G Altman. "Measuring Inconsistency in Meta-Analyses." *British Medical Journal 327, no. 7414 (September 6, 2003): 557-60.*

Hippel, Paul T von. "The Heterogeneity Statistic I2 Can Be Biased in Small Meta-Analyses." *BMC Medical Research Methodology 15 (April 14, 2015).* https://doi.org/10.1186/s12874-015-0024-z.

`pp_check.baggr` *Posterior predictive checks for baggr model*

Description

Performs posterior predictive checks with the **bayesplot** package.

Usage

```
## S3 method for class 'baggr'
pp_check(x, type = "dens_overlay", nsamples = 40)
```

Arguments

`x`	Model to check
`type`	type of pp_check. For a list see **here**.
`nsamples`	number of samples to compare

Details

For a detailed explanation of each of the ppc functions, see the `PPC` documentation of the **bayesplot** package.

`predict.baggr` *Predict method for baggr objects*

Description

Predict method for baggr objects

Usage

```
## S3 method for class 'baggr'
predict(object, nsamples, newdata = NULL, allow_new_levels = T, ...)
```

Arguments

object	model to predict from
nsamples	Number of samples to draw from the posterior. Cannot exceed the number of samples in the fitted model.
newdata	optional, new data to predict observations from
allow_new_levels	whether to allow the model to make predictions about unobserved groups. Without additional group-level information the model will use the unconditional, pooled estimate.
...	other arguments to pass to predict function (currently not used)

---

predict_mutau *Predict function for the mu & tau model*

Description

Predict function for the mu & tau model

Usage

```
predict_mutau(x, nsamples, newdata = NULL, allow_new_levels = T)
```

Arguments

x	model to predict from
nsamples	number of samples to predict
newdata	new data to predict, defaults to NULL
allow_new_levels	allow the predictive of new, unobserved groups

---

predict_quantiles *Predict function for the quantiles model*

Description

Predict function for the quantiles model

Usage

```
predict_quantiles(x, nsamples, newdata = NULL, allow_new_levels = T)
```

Arguments

x	model to predict from
nsamples	number of samples to predict
newdata	new data to predict, defaults to NULL
allow_new_levels	allow the predictive of new, unobserved groups

---

predict_rubin *Predict function for the rubin model*

Description

Predict function for the rubin model

Usage

```
predict_rubin(x, nsamples, newdata = NULL, allow_new_levels = T)
```

Arguments

x	model to predict from
nsamples	number of samples to predict
newdata	new data to predict, defaults to NULL
allow_new_levels	allow the predictive of new, unobserved groups

---

predict_unknown *Predict method for model that is unknown or not implemented*

Description

Predict method for model that is unknown or not implemented

Usage

```
predict_unknown(x)
```

Arguments

x	baggr model to generate predictions from

prepare_ma *Convert from individual to summary data in meta-analyses*

Description

Allows one-way conversion from full to summary data. Input must be pre-formatted appropriately.

Usage

```
prepare_ma(
  data,
  effect = c("mean", "logOR", "logRR"),
  rare_event_correction = 0.25,
  log = FALSE,
  cfb = FALSE,
  summarise = TRUE,
  treatment = "treatment",
  baseline = NULL,
  group = "group",
  outcome = "outcome"
)
```

Arguments

data	data.frame of individual-level observations with columns for outcome (numeric), treatment (values 0 and 1) and group (numeric, character or factor); column names can be user-defined (see below)
effect	what effect to calculate? a mean (and SE) of outcome in groups or (for binary data) logOR (odds ratio), logRR (risk ratio);
rare_event_correction	If effect is logOR or logRR, this correction is used when working with binary data only. The value of correction is added to all arms in trials where some arms had 0 events. Using corrections may bias results but is the only alternative to avoid infinite values.
log	logical; log-transform the outcome variable?
cfb	logical; calculate change from baseline? If yes, the outcome variable is taken as a difference between values in outcome and baseline columns
summarise	logical; TRUE by default, but you can disable it to obtain converted (e.g. logged) data with columns renamed
treatment	name of column with treatment variable
baseline	name of column with baseline variable
group	name of the column with grouping variable
outcome	name of column with outcome variable

Details

The conversions done by this function are not typically needed and may happen automatically when data is fed to baggr. However, this function can be used to explicitly convert from full to reduced (summarised) data without analysing it in any model. It can be useful for examining your data.

If multiple operations are performed, they are taken in this order:

1. conversion to log scale,
2. calculating change from baseline,
3. summarising data (using appropriate `effect`)

Value

- If you `summarise` data.frame with columns for `group` `tau` and `se.tau` (for `effect = "mean"`, also baseline means, for `"logRR"` or `"logOR"` also `a`, `b`, `c`, `d`, which correspond to typical contingency table notation).
- If you do not summarise data, individual level data will be returned, but some columns may be renamed or transformed (see above).

Author(s)

Witold Wiecek

See Also

convert_inputs for how any type of data is (internally) converted into Stan inputs;

`prepare_prior` *Prepare prior values for Stan models in baggr*

Description

This is an internal function called by baggr. You can use it for debugging or to run modified models. It extracts and prepares priors passed by the user. Then, if any necessary priors are missing, it sets them automatically and notifies user about these automatic choices.

Usage

```
prepare_prior(
  prior,
  data,
  stan_data,
  model,
  pooling,
  covariates,
  quantiles = c(),
  silent = FALSE
)
```

Arguments

prior	prior argument passed from baggr call
data	data another argument in baggr
stan_data	list of inputs that will be used by sampler this is already pre-obtained through convert_inputs
model	same as in baggr
pooling	same as in baggr
covariates	same as in baggr
quantiles	same as in baggr
silent	same as in baggr

Value

A named list with prior values that can be appended to stan_data and passed to a Stan model.

---

print.baggr *S3 print method for objects of class* baggr *(model fits)*

Description

This print method for a very concise summary of main model features. More info is included in the summary of the model and its attributes.

Usage

```
## S3 method for class 'baggr'
print(x, exponent = FALSE, digits = 2, group, fixed = TRUE, ...)
```

Arguments

x	object of class baggr
exponent	if TRUE, results (for means) are converted to exp scale
digits	Number of significant digits to print.
group	logical; print group effects? If unspecified, they are printed only if less than 20 groups are present
fixed	logical: print fixed effects?
...	currently unused by this package: further arguments passed to or from other methods (print requirement)

print.baggr_compare *Print method for baggr_compare models*

Description

Print method for baggr_compare models

Usage

```
## S3 method for class 'baggr_compare'
print(x, digits, ...)
```

Arguments

x	baggr_compare model
digits	number of significant digits for effect estimates
...	other parameters passed to print

print.baggr_cv *Print baggr cv objects nicely*

Description

Print baggr cv objects nicely

Usage

```
## S3 method for class 'baggr_cv'
print(x, digits = 3, ...)
```

Arguments

x	baggr_cv object to print
digits	number of digits to print
...	additional arguments for s3 consistency

print.compare_baggr_cv *Print baggr_cv comparisons*

Description

Print baggr_cv comparisons

Usage

```
## S3 method for class 'compare_baggr_cv'
print(x, digits = 3, ...)
```

Arguments

x	baggr_cv comparison to print
digits	number of digits to print
...	additional arguments for s3 consistency

print.plot_list *Print list of baggr plots*

Description

Print list of baggr plots

Usage

```
## S3 method for class 'plot_list'
print(x)
```

Arguments

x	list of plots to print

Details

prints plots in a loop, internal use only

priors *Prior distributions in baggr*

Description

This page provides a list of all available distributions that can be used to specify priors in `baggr()`. These convenience functions are designed to allow the user to write the priors in the most "natural" way when implementing them in baggr. Apart from passing on the arguments, their only other role is to perform a rudimentary check if the distribution is specified correctly.

Usage

```
multinormal(location, Sigma)

lkj(shape, order = NULL)

normal(location, scale)

cauchy(location, scale)

uniform(lower, upper)
```

Arguments

`location`	Mean for normal and multivariate normal (in which case `location` is a vector), and median for Cauchy distributions
`Sigma`	Variance-covariance matrix for multivariate normal.
`shape`	Shape parameter for LKJ
`order`	Order of LKJ matrix (typically it does not need to be specified, as it is inferred directly in the model)
`scale`	SD for Normal, scale for Cauchy
`lower`	Lower bound for Uniform
`upper`	Upper bound for Uniform

Details

The prior choice in baggr is always done via 3 distinct arguments: `prior_hypermean`, `prior_hypersd`, and `prior_hypercor`.

These respectively refer to the priors on the average of the effects across the groups (hypermean), the standard deviation of the effects across the groups (hypersd), and the correlation in the distribution of parameters across groups when the model allows multivariate shrinkage (say on control group means and effects).

Notation for priors is "plain-text", in that you can write the distributions as `normal(5,10)`, `uniform(0,100)` etc. As with any other argument one has the option to simply input the prior directly, e.g. `prior_hypermean`

= `normal(0,1)` , or by creating a named list of custom priors and then inputting the list to the argument `priors`. See the examples below for more.

Different parameters admit different priors:

- `prior_hypermean` will take `"normal"`, `"uniform"` and `"cauchy"` input for a scalar mean. For a vector mean, it will take any of these arguments and apply them independently to each component of the vector, or it can also take a `"multinormal"` argument (see the example below).
- `prior_hypersd` will take `"normal"` and `"uniform"`
- `prior_hypercor` allows `"lkj"` input

Author(s)

Witold Wiecek, Rachael Meager

References

Lewandowski, Daniel, Dorota Kurowicka, and Harry Joe. "Generating Random Correlation Matrices Based on Vines and Extended Onion Method." *Journal of Multivariate Analysis* 100, no. 9 (October 1, 2009): 1989-2001. https://doi.org/10.1016/j.jmva.2009.04.008.

Examples

```
# change the priors for 8 schools:
baggr(schools, model = "rubin", pooling = "partial",
      prior_hypermean = normal(5,5),
      prior_hypersd = normal(0,20))


# passing priors as a list
custom_priors <- list(hypercor = lkj(1), hypersd = normal(0,10),
                      hypermean = multinormal(c(0,0),matrix(c(10,3,3,10),2,2)))
baggr(microcredit_summary_data, model = "mutau",
      pooling = "partial", prior = custom_priors)
```

`rubin_data` *Make model matrix for the rubin data*

Description

Make model matrix for the rubin data

Usage

```
rubin_data(x, newdata = NULL, allow_new_levels = T)
```

Arguments

x	model to get data from
newdata	new data to use with model
allow_new_levels	whether to allow for unobserved groups

---

schools *8 schools example*

Description

A classic example of aggregate level continuous data in Bayesian hierarchical modelling. This dataframe contains a column of estimated treatment effects of an SAT prep program implemented in 8 different schools in the US, and a column of estimated standard errors.

Usage

```
schools
```

Format

An object of class `data.frame` with 8 rows and 3 columns.

Details

See Gelman et al (1995), Chapter 5, for context and applied example.

References

Gelman, Andrew, John B. Carlin, Hal S. Stern, and Donald B. Rubin. Bayesian Data Analysis. Taylor & Francis, 1995.

---

set_prior_val *Add prior values to Stan input for baggr*

Description

Add prior values to Stan input for baggr

Usage

```
set_prior_val(target, name, prior, p = 1)
```

Arguments

target	list object (Stan input) to which prior will be added
name	prior name, like hypermean, hypersd, hypercor
prior	one of prior distributions allowed by baggr like normal
p	number of repeats of the prior, i.e. when P i.i.d. priors are set for P dimensional parameter as in "mu & tau" type of model

---

show_model *Show Stan code for baggr models or objects*

Description

Show Stan code for baggr models or objects

Usage

```
show_model(model)
```

Arguments

model	either a baggr object (fitted model) or one of "rubin", "mutau", "individual"

Value

Nothing is returned in R. Stan code will be opened externally (e.g. via notepad).

---

stop_not_implemented *Stop with informative error*

Description

Stop with informative error

Usage

```
stop_not_implemented()
```

`treatment_effect` *Average treatment effect in a baggr model*

Description

Average treatment effect in a baggr model

Usage

```
treatment_effect(bg, summary = FALSE, transform = NULL, interval = 0.95)
```

Arguments

`bg`	a baggr model
`summary`	logical; if TRUE returns summary statistics as explained below.
`transform`	a transformation to apply to the result, should be an R function; (this is commonly used when calling `treatment_effect` from other plotting or printing functions)
`interval`	uncertainty interval width (numeric between 0 and 1), if summarising

Value

A list with 2 vectors (corresponding to MCMC samples) `tau` (mean effect) and `sigma_tau` (SD). If `summary=TRUE`, both vectors are summarised as mean and lower/upper bounds according to `interval`

Index