Package ‘baggr’

January 4, 2022

Type Package
Title Bayesian Aggregate Treatment Effects
Version 0.6.18
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. This implements many models from Meager (2019) <doi:10.1257/app.20170299>.
License GPL (>= 3)
Encoding UTF-8
LazyData true
Biarch true
Depends R (>= 3.5.0), Rcpp (>= 0.12.17)
Imports rstan (>= 2.18.1), rstantools (>= 2.1.1), bayesplot, crayon, forestplot, ggplot2, gridExtra, utils, stats, testthat, methods
LinkingTo StanHeaders (>= 2.18.1), rstan (>= 2.18.1), BH (>= 1.66.0-1), Rcpp (>= 0.12.17), RcppParallel (>= 5.0.1), RcppEigen (>= 0.3.3.4.0)
SystemRequirements GNU make
NeedsCompilation yes
RoxygenNote 7.1.2
Suggests knitr, covr, rmarkdown
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)
R topics documented:

- baggr-package .......................................................... 3
- baggr ................................................................. 4
- baggr_compare .......................................................... 8
- baggr_plot ............................................................ 10
- baggr_theme_set ....................................................... 11
- binary_to_individual ............................................... 13
- chicks ................................................................. 14
- convert_inputs ......................................................... 14
- data_spike ............................................................ 16
- effect_draw ............................................................ 16
- effect_plot ............................................................ 17
- fixed_effects .......................................................... 18
- forest_plot ............................................................ 19
- get_n_samples ......................................................... 20
- get_order ............................................................. 21
- group_effects ........................................................ 21
- is.baggr_cv ........................................................... 22
- labbe ................................................................. 23
- loocv ................................................................. 24
- loo_compare .......................................................... 25
- microcredit ........................................................... 26
- microcredit_simplified ............................................. 26
- mint ................................................................. 27
- mutau_cor ............................................................. 28
- plot.baggr ............................................................. 28
- plot.baggr_compare ................................................... 29
- plot.baggr_cv ........................................................ 30
- plot_quantiles ......................................................... 31
- pooling ............................................................... 31
- pp_check.baggr ........................................................ 34
- predict.baggr ........................................................ 34
- predict_mutau ........................................................ 35
- predict_quantiles ...................................................... 35
- predict_rubin ........................................................ 36
- predict_unknown ....................................................... 36
- prepare_ma ........................................................... 37
- prepare_prior ........................................................ 39
- print.baggr ............................................................ 40
- print.baggr_compare .................................................. 40
- print.baggr_cv ........................................................ 41
- print.compare_baggr_cv ............................................ 41
- print_dist ............................................................. 42
Description

This is baggr (pronounced as bagger or badger), a Bayesian meta-analysis package for R that uses Stan to fit the models. 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 are 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").)

References

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

```r
baggr(
  data,
  model = NULL,
  pooling = c("partial", "none", "full"),
  effect = NULL,
  covariates = c(),
  prior_hypermean = NULL,
  prior_hypersd = NULL,
  prior_hypercor = NULL,
  prior_beta = NULL,
  prior_control = NULL,
  prior_control_sd = NULL,
  prior_sigma = NULL,
  prior = NULL,
  ppd = FALSE,
  pooling_control = c("none", "partial"),
  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; see Details section for how to format your data

model if NULL, detected automatically from input data otherwise choose from "rubin", "mutau", "rubin_full", "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 section 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_control**

Prior for the mean in the control arm (baseline), currently used in "logit" model only; if pooling_control = "partial", the prior is hyperprior for all baselines, if "none", then it is an independent prior for all baselines

**prior_control_sd**

Prior for the SD in the control arm (baseline), currently used in "logit" model only; this can only be used if pooling_control = "partial"

**prior_sigma**

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.) If ppd=TRUE, Stan model will sample from the prior distribution(s) and ignore data in inference. However, data argument might still be used to infer the correct model (if model=NULL) and to set the default priors, therefore you must specify it.

**pooling_control**

Pooling for group-specific control mean terms in models using individual-level data. Either "none" or "partial".

**test_data**

data for cross-validation; NULL for no validation, otherwise a data frame with the same columns as data argument. See "Cross-validation" section below.

**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 behav­
ior.
warn print an additional warning if Rhat exceeds 1.05
... extra options passed to Stan function, e.g. control = list(adapt_delta =

Details

Below we briefly discuss 1/ data preparation, 2/ choice of model, 3/ choice of priors. All three are
discussed in more depth in the package vignette, vignette("baggr").

Data. For aggregate data models you need a data frame with columns tau and se (Rubin model) or
tau, mu, se.tau, se.mu ("mu & tau" model). 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 (using sum-
mary data), "mutau" version which takes into account means of control groups (also us-
ing summary data), "rubin_full", which is the same model as "rubin" but works with
individual-level data
• for continuous variable quantiles: "quantiles" model (see Meager, 2019 in references)
• for mixture data: "sslab" (experimental)
• 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 "rubin_full" models, covariates that change according to individual unit.
Then, such a model is commonly referred to as a mixed model. It has to be fitted to individual-
level data. Note that meta-regression is a special case of a mixed model for individual-level
data.
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. Meaning of the prior parameters may slightly change from model to model. Details and examples are given in vignette(“baggr”). Setting `ppd=TRUE` can be used to obtain prior predictive distributions, which is useful for understanding the prior assumptions, especially useful in conjunction with `effect_plot`. You can also `baggr_compare` different priors by setting `baggr_compare(..., compare="prior")`.

Cross-validation. When `test_data` are specified, an extra parameter, the log predictive density, will be returned by the model. (The fitted model itself is the same regardless of whether there are `test_data`.) To understand this parameter, see documentation of `loocv`, a function that can be used to assess out of sample prediction of the model using all available data. If using individual-level data model, `test_data` should only include treatment arms of the groups of interest. (This is because in cross-validation we are not typically interested in the model’s ability to fit heterogeneity in control arms, but only heterogeneity in treatment arms.) For using aggregate level data, there is no such restriction.

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 non-default (and 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
microcredit_summary_data <- prepare_ma(ms, outcome = "consumption")
baggr(microcredit_summary_data, model = "mutau",
      iter = 500, #this is just for illustration -- don't set it this low normally!
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**

```r
baggr_compare(
  ..., 
  what = "pooling",
  compare = c("groups", "hyperpars", "effects"),
  transform = NULL,
  plot = FALSE
)
```

**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), "hyperpars" (to omit group-specific estimates) or (predicted) "effects". The "groups" option 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
plot  logical; calls `plot.baggr_compare` when running `baggr_compare`

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 then 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

```r
# 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',
    #this is just for illustration -- don't set it this low normally!
    iter = 500,
    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',
    #this is just for illustration -- don't set it this low normally!
    iter = 500,
    prior_hypermean = normal(0, 3),
    prior_hypersd = normal(0,2),
    prior_hypercor = lkj(2),
    what = "pooling",
    # You can automatically plot:
    plot = TRUE)
```

```r
# Compare existing models (you don't have to, but best to name them):
bg1 <- baggr(schools, pooling = "partial")
bg2 <- baggr(schools, pooling = "full")
baggr_compare("Partial pooling model" = bg1, "Full pooling" = bg2)

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

# Compare how posterior predictive effect varies with e.g. choice of prior
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", plot = TRUE)

# Models don't have to be identical. 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,
    plot = TRUE)
```

---

**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**

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

bg object of class baggr
hyper logical; show hypereffect as the last row of the plot?
style either "intervals" or "areas"
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; the visual style is based on bayesplot package

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

```r
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

---

**binary_to_individual**  
*Generate individual-level binary outcome data from an aggregate statistics*

### Description

This is a helper function that is typically used automatically by some of *baggr* functions, such as when running `model="logit"` in *baggr*, when summary-level data are supplied.

### Usage

```
binary_to_individual(data, group = "group", rename_group = TRUE)
```

### Arguments

- **data**  
  A data frame with columns `a`, `c` and `b/n1`, `d/n2`. (You can also use `ai`, `ci`, `n1i`, `n2i` instead.)
- **group**  
  Column name storing group
- **rename_group**  
  If `TRUE` (default), this will rename the grouping variable to "group", making it easier to work with *baggr*
  
  See vignette("baggr_binary") for an example of use and notation details.

### Value

A data frame with columns `group`, `outcome` and `treatment`.

### See Also

`prepare_ma` uses this function

### Examples

```r
df_yusuf <- read.table(text="
trial    a   n1i  c   n2i
Balcon   14  56  15  58
Clausen  18  66  19  64
Multicentre 15 100 12  95
Barber   10  52  12  47
Norris   21 226 24 228
Kahler    3  38  6  31
")
```

This is a helper function that is typically used automatically by some of *baggr* functions, such as when running `model="logit"` in *baggr*, when summary-level data are supplied.
```
Ledwich 2 20 3 20
", header=TRUE)
bti <- binary_to_individual(df_yusuf, group = "trial")
head(bti)
# to go back to summary-level data
prepare_ma(bti, effect = "logOR")
# the last operation is equivalent to simply doing
prepare_ma(df_yusuf, group="trial", effect="logOR")
```

---

**chicks**  
*Chickens: impact of electromagnetic field on calcium ion efflux in chicken brains*

**Description**

An experiment conducted by Blackman et al. (1988) and documented in the following GitHub repository by Vakar and Gelman. The dataset consists of a large number of experiments (tau, se.tau) repeated at varying wave frequencies. Sham experiments (mu, se.mu) are also included, allowing us to compare performance of models with and without control measurements.

**Usage**

chicks

**Format**

An object of class `data.frame` with 38 rows and 7 columns.

**References**


---

**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".
convert_inputs

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")
data_spike  

Spike & slab example dataset

Description

Spike & slab example dataset

Usage

data_spike

Format

An object of class data.frame with 1500 rows and 4 columns.

effect_draw  

Make predictive draws from baggr model

Description

This function takes the samples of hyperparameters from a baggr model (typically hypermean and hyper-SD, which you can see using treatment_effect) and draws values of new realisations of treatment effect, i.e. an additional draw from the "population of studies". This can be used for both prior and posterior draws, depending on baggr model.

Usage

effect_draw(
  x,
  n,
  transform = NULL,
  summary = FALSE,
  message = TRUE,
  interval = 0.95
)

Arguments

x  
A baggr class object.

n  
How many values to draw? The default is as long as the number of samples in the baggr object (see Details).

transform  
a transformation (an R function) to apply to the result of a draw.

summary  
logical; if TRUE returns summary statistics rather than samples from the distribution;

message  
logical; use to disable messages prompted by using with no pooling models

interval  
uncertainty interval width (numeric between 0 and 1), if summary=TRUE
Details

The predictive distribution can be used to "combine" heterogeneity between treatment effects and uncertainty in the mean treatment effect. This is useful both in understanding impact of heterogeneity (see Riley et al, 2011, for a simple introduction) and for study design e.g. as priors in analysis of future data (since the draws can be seen as an expected treatment effect in a hypothetical study).

The default number of samples is the same as what is returned by Stan model implemented in baggr, (depending on such options as iter, chains, thin). If n is larger than what is available in Stan model, we draw values with replacement. This is not recommended and warning is printed in these cases.

Under default settings in baggr, a posterior predictive distribution is obtained. But effect_draw can also be used for prior predictive distributions when setting ppd=T in baggr. The two outputs work exactly the same way.

Value

A vector (with n values) for models with one treatment effect parameter, a matrix (n rows and same number of columns as number of parameters) otherwise.

References


See Also

treatment_effect returns samples of hypermean and hyper-SD which are used by this function

effect_plot

Plot predictive draws from baggr model

description

This function plots values from effect_draw, the predictive distribution (under default settings, posterior predictive), for one or more baggr objects.

Usage

effect_plot(..., transform = NULL)

Arguments

... Object(s) of class baggr. If there is more than one, a comparison will be plotted and names of objects will be used as a plot legend (see examples).

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)
fixed_effects

Details
Under default settings in `baggr` posterior predictive is obtained. But `effect_plot` can also be used for prior predictive distributions when setting `ppd=T` in `baggr`. The two outputs work exactly the same, but labels will change to indicate this difference.

Value
A ggplot object.

See Also
`effect_draw` documents the process of drawing values; `baggr_compare` can be used as a shortcut for `effect_plot` with argument `compare = "effects"`

Examples

```r
# 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)`
**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**

```r
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")
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

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 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_order

Separate out ordering so we can test directly

Description
Separate out ordering so we can test directly

Usage
get_order(df_groups, hyper)

Arguments
- df_groups: data.frame of group effects used in plot.baggr_compare
- hyper: show parameter estimate? same as in plot.baggr_compare

Details
Given a set of effects measured by models, identifies the model which has the biggest range of estimates and ranks groups by those estimates, returning the order

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. If there are no covariates in the model, this effect is a single random variable. If there are covariates, the group effect is a sum of effect of covariates (fixed effects) and the study-specific random variable (random effects). 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,
  random_only = FALSE
)
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
**labbe**

L'Abbe plot for binary data

---

**Description**

This plot shows relationship between proportions of events in control and treatment groups in binary data.

**Usage**

```r
labbe(
  data,
  group = "group",
  plot_model = FALSE,
  labels = TRUE,
  shade_se = c("rr", "or", "none")
)
```

**Arguments**

- `data`: a data frame with binary data (must have columns `a`, `c/b/n1`, `d/n2`)
- `group`: a character string specifying group names (e.g. study names), used for labels;
- `plot_model`: if TRUE, then odds ratios and risk ratios baggr models are estimated (using default settings) and their mean estimates of effects are plotted as lines
- `labels`: if TRUE, names from the `group` column are displayed
- `shade_se`: if "none", nothing is plotted, if "or" or "rr", a shaded area corresponding to inverse of effect's (OR or RR) SE is added to each data point; the default is "rr"

**Value**

A `ggplot` object

**See Also**

- `vignette("baggr_binary")` for an illustrative example
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 excluding any one group affects the overall result, as well as how well the model predicts the omitted group. LOO-CV approaches are a good general practice for comparing Bayesian models, not only in meta-analysis.

This function automatically runs K baggr models, where K is number of groups (e.g. studies), leaving out one group at a time. For each run, it calculates expected log predictive density (ELPD) for that group (see Gelman et al 2013). (In the logistic model, where the proportion in control group is unknown, each of the groups is divided into data for controls, which is kept for estimation, and data for treated units, which is not used for estimation but only for calculating predictive density. This is akin to fixing the baseline risk and only trying to infer the odds ratio.)

The main output is the cross-validation information criterion, or -2 times the ELPD summed over K models. (We sum the terms as we are working with logarithms.) This is related to, and often approximated by, the Watanabe-Akaike Information Criterion. When comparing models, smaller values mean 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


See Also

loo_compare for comparison of many LOO CV results; you can print and plot output via plot.baggr_cv and print.baggr_cv

Examples

## Not run:
# 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

## End(Not run)

---

**loo_compare**

*Compare LOO CV models*

**Description**

Given multiple loocv outputs, calculate differences in their expected log predictive density.

**Usage**

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

**Arguments**

- `x`: An object of class baggr.cv or a list of such objects.
- `...`: Additional objects of class "baggr.cv"

**See Also**

loocv for fitting LOO CV objects and explanation of the procedure

**Examples**

## Not run:
# 2 models with more/less informative priors -- this will take a while to run
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)

## End(Not run)
microcredit_simplified

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

---

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 consumption 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 consumption outcome variable.

References


```
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))
```
mutau_cor  
Correlation between mu and tau in a baggr model

Description
Correlation between mu and tau in a baggr model

Usage
mutau_cor(bg, summary = FALSE, interval = 0.95)

Arguments
bg  
a baggr model where model = "mutau"
summary  
logical; if TRUE returns summary statistics as explained below.
interval  
uncertainty interval width (numeric between 0 and 1), if summarising

Value
a vector of values

plot.baggr  
Plotting method for baggr outputs

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 Wiecik
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

```r
## S3 method for class 'baggr_compare'
plot(
x, 
compare = x$compare, 
style = "areas", 
grid_models = FALSE, 
grid_parameters = TRUE, 
interval = 0.95, 
hyper = TRUE, 
transform = NULL, 
order = F, 
vline = FALSE, 
add_values = FALSE, 
values_digits = 2, 
values_size = 2, 
... 
)
```

Arguments

- **x**: baggr_compare model to plot
- **compare**: When plotting, choose between comparison of "groups" (default), "hyperpars" (to omit group-specific estimates) or (predicted) "effects". The "groups" option is not available when what = "prior".
- **style**: What kind of plot to display (if grid_models = TRUE), passed to the style argument in baggr_plot.
- **grid_models**: If FALSE (default), generate a single comparison plot; if TRUE, display each model (using individual baggr_plot’s) side-by-side.
- **grid_parameters**: If TRUE, uses ggplot-style facetting when plotting models with many parameters (especially "quantiles", "sslab"); if FALSE, returns separate plot for each parameter
- **interval**: Probability level used for display of posterior interval
- **hyper**: Whether to plot pooled treatment effect in addition to group treatment effects when compare = "groups"
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 sort by median treatment effect by group. If yes, medians from the model with largest range of estimates are used for sorting. If not, groups are shown alphabetically.

vline: logical; show vertical line through 0 in the plot?

add_values: logical; if TRUE, values will be printed next to the plot, in a style that’s similar to what is done for forest plots.

values_digits: number of significant digits to use when printing values,

values_size: size of font for the values, if add_values == TRUE

...: ignored for now, may be used in the future

plot.baggr_cv

### S3 method for class 'baggr_cv'
plot(x, y, ..., add_values = TRUE)

Arguments

x: output from loocv that has return_models = TRUE

y: Unused, ignore

...: Unused, ignore

add_values: logical; if TRUE, values of elpd are printed next to each study

Value

ggplot2 plot in similar style to baggr_compare default plots.
plot_quantiles

Description
Plot results for baggr quantile models. Displays results facetted per group. Results are ggplot2 plots and can be modified.

Usage
plot_quantiles(fit, ncol, hline = TRUE)

Arguments
- **fit**: an object of class `baggr`
- **ncol**: number of columns for the plot; defaults to half of number of groups
- **hline**: logical; plots a line through 0

Value
ggplot2 object

Examples
```r
## Not run:
bg <- baggr(microcredit_simplified, model = "quantiles",
            quantiles = c(0.25, 0.50, 0.75),
            iter = 1000, refresh = 0,
            outcome = "consumption")
# vanilla plot
plot_quantiles(bg)[[1]]
plot_quantiles(bg, hline = TRUE)[[2]] +
  ggplot2::coord_cartesian(ylim = c(-2, 5)) +
  ggplot2::ggtitle("Works like a ggplot2 plot!")
## End(Not run)
```

pooling

Pooling metrics and related statistics for baggr

Description
Compute statistics relating to pooling in a given `baggr` meta-analysis model returns statistics, for either the entire model or individual groups, such as pooling statistic by Gelman & Pardoe (2006), I-squared, H-squared, or study weights; heterogeneity is a shorthand for pooling(type = "total") weights is shorthand for pooling(metric = "weights")
Usage

```r
pooling(
  bg,
  metric = c("pooling", "isq", "hsq", "weights"),
  type = c("groups", "total"),
  summary = TRUE
)
```

```r
heterogeneity(
  bg,
  metric = c("pooling", "isq", "hsq", "weights"),
  summary = TRUE
)
```

```r
## S3 method for class 'baggr'
weights(object, ...)
```

Arguments

- **bg**: a baggr model
- **metric**: "pooling" for Gelman & Pardoe statistic $P$, "isq" for I-squared statistic $(1-P)$, Higgins & Thompson, 2002) "hsq" for $H$ squared statistic $(1/P)$, ibid.; "weights" for study weights; also see Details
- **type**: In pooling calculation is done for each of the "groups" (default) or for "total" hypereffect(s).
- **summary**: logical; if FALSE a whole vector of pooling values is returned, otherwise only the means and intervals
- **object**: baggr model for which to calculate group (study) weights
- **...**: Unused, please ignore.

Details

Pooling statistic (Gelman & Pardoe, 2006) describes the extent to which group-level estimates of treatment effect are "pooled" toward average treatment effect in the meta-analysis model. If pooling = "none" or "full" (which you specify when calling baggr), then the values are always 0 or 1, respectively. If pooling = "partial", the value is somewhere between 0 and 1. We can distinguish between pooling of individual groups and overall pooling in the model.

In many contexts, i.e. medical statistics, it is typical to report $1-P$, called $I^2$ (see Higgins and Thompson, 2002; sometimes another statistic, $H^2 = 1/P$, is used). Higher values of $I$-squared indicate higher heterogeneity; Von Hippel (2015) provides useful details for $I$-squared calculations (and some issues related to it, especially in frequentist models). See Gelman & Pardoe (2006) Section 1.1 for a short explanation of how $R^2$ statistic relates to the pooling metric.

Group pooling

This is the calculation done by pooling() if type = "groups" (default). In a partial pooling model (see baggr and above), group $k$ (e.g. study) has standard error of treatment effect estimate, $se_k$. The treatment effect (across $k$ groups) is variable across groups, with hyper-SD parameter $\sigma(\tau)$. 

The quantity of interest is ratio of variation in treatment effects to the total variation. By convention, we subtract it from 1, to obtain a *pooling metric* \( P \).

\[
p = 1 - \frac{(\sigma(\tau)^2)}{(\sigma(\tau)^2 + se_k^2)}
\]

- If \( p < 0.5 \), 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 indicate no pooling. Variation within studies dominates.

Note that, since \( \sigma^2 \) 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 \( \sigma_\tau \).

To obtain samples from the distribution of \( P \) (rather than summarised values), set `summary=FALSE`.

### Study weights

Contributions of each group (e.g. each study) to the mean meta-analysis estimate can be calculated by calculating for each study \( w_k \) the inverse of sum of group-specific SE squared and between-study variation. To obtain weights, this vector (across all studies) has to be normalised to 1, i.e. \( w_k / \text{sum}(w_k) \) for each \( k \).

SE is typically treated as a fixed quantity (and usually reported on the reported point estimate), but between-study variance is a model parameter, hence the weights themselves are also random variables.

### Overall pooling in the model

Typically researchers want to report a single measure from the model, relating to heterogeneity across groups. This is calculated by either `pooling(mymodel,type = "total")` or simply `heterogeneity(mymodel)`.

Formule for the calculations below are provided in main package vignette and almost analogous to the group calculation above, but using mean variance across all studies. In other words, pooling \( P \) is simply ratio of the expected within-study variance term to total variance.

To obtain such single estimate we need to substitute average variability of group-specific treatment effects and then calculate the same way we would calculate \( p \). By default we use the mean across \( k \) \( se_k^2 \) values. Typically, implementations of \( I^2 \) in statistical packages use a different calculation for this quantity, which may make \( I^2 \)'s not comparable when different studies have different SE’s.

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

### Value

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

### References


---

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

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

**Usage**  
```r
## 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**  
```r
## S3 method for class 'baggr'
predict(object, nsamples, newdata = NULL, allow_new_levels = TRUE, ...)
```
**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)

**Description**

Predict function for the mu & tau model

**Usage**

```r
predict_mutau(x, nsamples, newdata = NULL, allow_new_levels = TRUE)
```

**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

**Description**

Predict function for the quantiles model

**Usage**

```r
predict_quantiles(x, nsamples, newdata = NULL, allow_new_levels = TRUE)
```
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**

```r
predict_rubin(x, nsamples, newdata = NULL, allow_new_levels = TRUE)
```

**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**

```r
predict_unknown(x)
```

**Arguments**

- **x**: baggr model to generate predictions from
Prepare for Meta-Analyses

**Description**

Allows for one-way conversion from full to summary data or for calculation of effects for binary data. Input must be pre-formatted appropriately.

**Usage**

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

**Arguments**

- `data` either a data.frame of individual-level observations with columns for outcome (numeric), treatment (values 0 and 1) and group (numeric, character or factor); or, a data frame with binary data (must have columns a, b/n1, d/n2).
- `effect` what effect to calculate? a mean (and SE) of outcome in groups or (for binary data) logOR (odds ratio), logRR (risk ratio), RD (risk difference);
- `rare_event_correction` This correction is used when working with binary data (effect logOR or logRR) The value of correction is added to all cells in either some or all rows (groups), depending on `correction_type`. Using corrections may bias results but is the only alternative to avoid infinite values.
- `correction_type` If "single" then rare event correction is only applied to the particular rows that have 0 cells, if "all", then to all studies
- `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
prepare_ma

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
pooling  Internal use only, please ignore

Details

The conversions done by this function are not typically needed and may happen automatically when data is given 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 and generating summary tables.

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: a 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, that is: a = events in exposed; b = no events in exposed, c = events in unexposed, d = no events in unexposed).
- If you do not summarise data, individual level data will be returned, but some columns may be renamed or transformed (see the arguments above).

Author(s)

Witold Wiecek

See Also

convert_inputs for how any type of data is (internally) converted into a list of Stan inputs; vignette baggr_binary for more details about rare event corrections

Examples

```r
# Example of working with binary outcomes data
# Make up some individual-level data first:
df_rare <- data.frame(group = paste("Study", LETTERS[1:5]),
a = c(0, 2, 1, 3, 1), c = c(2, 2, 3, 3, 5),
n1i = c(120, 300, 110, 250, 95),
n2i = c(120, 300, 110, 250, 95))
df_rare_ind <- binary_to_individual(df_rare)
# Calculate ORs; default rare event correction will be applied
prepare_ma(df_rare_ind, effect = "logOR")
```
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 prints a concise summary of the main baggr 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 obtained from loocv to print
digits: number of digits to print
... Unused, ignore

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_dist  
*Output a distribution as a string*

**Description**

Used for printing nicely formatted outputs when reporting results etc.

**Usage**

```r
print_dist(dist)
```

**Arguments**

- `dist`  
  
  distribution name, one of `priors`

**Value**

Character string like `normal(0,10^2)`.

---

**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**

```r
multinormal(location, Sigma)

lkj(shape, order = NULL)

normal(location, scale)

lognormal(mu, sigma)

ciauchy(location, scale)

uniform(lower, upper)
```
priors

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
- **mu**: mean of ln(X) for lognormal
- **sigma**: SD of ln(X) for lognormal
- **lower**: Lower bound for Uniform
- **upper**: Upper bound for Uniform

Details

The prior choice in baggr is done via distinct arguments for each type of prior, e.g. `prior_hypermean`, or a named list of several passed to `prior`. See the examples below.

Notation for priors is "plain-text", in that you can write the distributions as `normal(5,10)`, `uniform(0,100)` etc.

Different parameters admit different priors (see baggr for explanations of what the different prior_arguments do):

- `prior_hypermean, prior_control, and prior_beta will take "normal", "uniform", "lognormal", and "cauchy" input for scalars. For a vector hypermean (see "mutau" model), 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, prior_control_sd, and prior_sigma will take "normal", "uniform", and "lognormal" but negative parts of the distribution are truncated
- `prior_hypercor allows "lkj" input (see Lewandowski et al.)`

Author(s)

Witold Wiecek, Rachael Meager

References


Examples

# (these are not the recommended priors -- for syntax illustration only)

# change the priors for 8 schools:
baggr(schools, model = "rubin", pooling = "partial",

---

43
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)))
microcredit_summary_data <- prepare_ma(microcredit, outcome = "consumption")
baggr(microcredit_summary_data, model = "mutau",
     pooling = "partial", prior = custom_priors)

random_effects

Extract random effects from a baggr model

Description
This is a shortcut for writing group_effects(random_only=TRUE,...)

Usage
random_effects(...)

Arguments
... arguments passed to group_effects

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 = TRUE)

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


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

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>target</td>
<td>list object (Stan input) to which prior will be added</td>
</tr>
<tr>
<td>name</td>
<td>prior name, like hypermean, hypersd, hypercor</td>
</tr>
<tr>
<td>prior</td>
<td>one of prior distributions allowed by baggr like normal</td>
</tr>
<tr>
<td>p</td>
<td>number of repeats of the prior, i.e. when P i.i.d. priors are set for P dimensional parameter as in &quot;mu &amp; tau&quot; type of model</td>
</tr>
</tbody>
</table>
single_comp_plot

Plot single comparison ggplot in baggr_compare style

Description

Plot single comparison ggplot in baggr_compare style

Usage

single_comp_plot(
  df,
  title = "",
  legend = "top",
  ylab = "",
  grid = F,
  points = FALSE,
  add_values = FALSE,
  values_digits = 2,
  values_size = 2.5
)

Arguments

df  data.frame with columns group, median, lci, uci, model (character or factor listing compared models) and, optionally, parameter (character or factor with name of parameter)
title ggttitle argument passed to ggplot
legend legend.position argument passed to ggplot
ylab Y axis label
grid logical; if TRUE, facets the plot by values in the parameter column
points you can optionally specify a (numeric) column that has values of points to be plotted next to intervals
add_values logical; if TRUE, values will be printed next to the plot, in a style that's similar to what is done for forest plots
values_digits number of significant digits to use when printing values,
values_size size of font for the values, if add_values == TRUE

Value

a ggplot2 object
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,
  message = TRUE
)

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
- `message`: logical; use to disable messages prompted by using with no pooling models

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

* datasets
  chicks, 14
  data_spike, 16
  microcredit, 26
  microcredit_simplified, 26
  schools, 45

  baggr, 3, 4, 8, 9, 13–19, 23, 24, 28, 31, 32, 38–40, 43, 47
  baggr(), 42
  baggr-package, 3, 4
  baggr_compare, 5, 7, 8, 18, 30
  baggr_plot, 7, 10, 20, 28, 29
  baggr_theme_get (baggr_theme_set), 11
  baggr_theme_replace (baggr_theme_set), 11
  baggr_theme_set, 11
  baggr_theme_update (baggr_theme_set), 11
  bayesplot::bayesplot_theme_get, 12
  bayesplot::bayesplot_theme_get(), 12
  bayesplot::MCMC-intervals, 11
  binary_to_individual, 13

  cauchy (priors), 42
  chicks, 14
  convert_inputs, 14, 38, 39

  data_spike, 16

  effect_draw, 7, 16, 17–19
  effect_plot, 7, 11, 17, 19, 20

  fixed_effects, 7, 18, 22
  forest_plot, 11, 19
  forestplot, 19, 20

  get_n_samples, 20
  get_order, 21
  ggplot2::theme_get(), 11
  group_effects, 7, 19, 21, 44

  here, 34
  heterogeneity (pooling), 31
  is.baggr_cv, 22
  labbe, 23
  lkj (priors), 42
  lognormal (priors), 42
  loo_compare, 25, 25
  loocv, 7, 24, 25, 30, 41
  microcredit, 26
  microcredit_simplified, 26
  mint, 27
  multinormal (priors), 42
  mutau_cor, 28
  normal, 45
  normal (priors), 42

  plot.baggr, 28
  plot.baggr_compare, 9, 21, 29
  plot.baggr_cv, 25, 30
  plot_quantiles, 31
  pooling, 31
  pp_check, 7
  pp_check (pp_check.baggr), 34
  pp_check.baggr, 34
  PPC, 34
  predict.baggr, 34
  predict_mutau, 35
  predict_quantiles, 35
  predict_rubin, 36
  predict_unknown, 36
  prepare_ma, 6, 13, 37
  prepare_prior, 39
  print.baggr, 40
  print.baggr_compare, 9, 40
  print.baggr_cv, 25, 41
  print.compare_baggr_cv, 41
  print_dist, 42
INDEX

priors, 42, 42
random_effects, 22, 44
round, 27
rubin_data, 44
schools, 45
set_prior_val, 45
single_comp_plot, 46
stop_not_implemented, 47
treatment_effect, 7, 16, 17, 19, 47
uniform (priors), 42
weights.baggr (pooling), 31