Package: RprobitB (via r-universe)

October 29, 2026

Type Package

Title Bayesian Probit Choice Modeling

Version 2.0.1

Description Fits Bayesian probit models for binary, multinomial, ordered, and ranked choices in cross-sectional and panel data. Correlated or uncorrelated normal and log-normal random coefficients, finite mixtures, sparse finite mixtures, and Dirichlet process mixtures describe preference heterogeneity. Multiple Gibbs chains produce posterior draws for diagnostics and choice prediction. Choice data can be supplied as a data frame or simulated from the requested specification. For an overarching treatment of the methodology, see Oelschlaeger (2026) <https://pub.uni-bielefeld.de/record/3014719>. The latent-class model is described in Oelschlaeger and Bauer (2021) <https://trid.trb.org/view/1759753>.

URL https://loelschlaeger.de/RprobitB/, https://github.com/loelschlaeger/RprobitB

BugReports https://github.com/loelschlaeger/RprobitB/issues

License GPL-3

Encoding UTF-8

Language en-US

Imports bayesplot, bridgesampling, checkmate, choicedata (>= 0.2.1), cli, Formula, future.apply, ggplot2, loo, oeli (>= 0.7.9), posterior, progressr, Rdpack, Rcpp, stats

LinkingTo oeli (>= 0.7.9), Rcpp, RcppArmadillo, testthat

Suggests AER, gridExtra, knitr, MASS, mlogit, plotROC, rmarkdown, testthat (>= 3.0.0), xml2

RdMacros Rdpack

Depends R (>= 4.1.0)

Roxygen list(markdown = TRUE)

VignetteBuilder knitr

Config/testthat/edition 3

Config/roxygen2/version 8.1.0

Config/pak/sysreqs cmake make libicu-dev libuv1-dev libssl-dev

Repository https://loelschlaeger.r-universe.dev

Date/Publication 2026-10-29 15:37:26 UTC

RemoteUrl https://github.com/loelschlaeger/rprobitb

RemoteRef HEAD

RemoteSha bc0c36381943fe238f8ecd115d724bff5051fd1d

Contents

as_draws.RprobitB_fit . . . 3
bayes_factor . . . 3
class_updates . . . 5
coef.RprobitB_fit . . . 8
coefficient_updates . . . 9
confint.RprobitB_fit . . . 11
fit . . . 12
formula.RprobitB_fit . . . 21
interpret . . . 21
latent_class_diagnostics . . . 24
logLik.RprobitB_fit . . . 25
loo . . . 26
model.frame.RprobitB_fit . . . 27
nobs.RprobitB_fit . . . 28
plot.RprobitB_fit . . . 28
predict.RprobitB_fit . . . 30
print.RprobitB_fit . . . 32
print.summary.RprobitB_fit . . . 33
residuals.RprobitB_fit . . . 33
summary.RprobitB_fit . . . 34
update.RprobitB_fit . . . 36
utility_updates . . . 37
vcov.RprobitB_fit . . . 39
WAIC . . . 40

Index **41**

as_draws.RprobitB_fit *Convert a fitted model to posterior draws*

Description

Returns the canonical retained posterior draws.

Usage

```
## S3 method for class 'RprobitB_fit'
as_draws(x, ...)
```

Arguments

x	[RprobitB_fit] Fitted choice model.
...	Currently not used.

Value

A draws_array with dimensions iteration, chain, and variable.

Examples

```
set.seed(1)
model <- fit(
  choice ~ x | 0, dgp_parameters = list(beta = c(x = 1)), chains = 1
)
posterior::as_draws(model)
```

bayes_factor *Compare models with a Bayes factor*

Description

Estimates both marginal likelihoods with bridge sampling (Meng and Wong 1996; Frühwirth-Schnatter 2004) and compares the first model with the second model using bridgesampling::bf().

Usage

```
bayes_factor(
  model1,
  model2,
  log = FALSE,
  repetitions = 1L,
  ghk_draws = 500L,
  progress = interactive()
)
```

Arguments

model1, model2	[RprobitB_fit] Fitted models to compare.
log	[logical(1)] Return the logarithm of the Bayes factor?
repetitions	[integer(1)] Number of bridge-sampling repetitions.
ghk_draws	[integer(1)] Number of draws of the GHK simulator for multivariate normal probabilities of more than three dimensions, see `oeli::pmvnorm()`.
progress	[logical(1)] Show progress?

Value

A `bf_bridge` object from **bridgesampling**. Values greater than one (zero if `log = TRUE`) favor `model1`; smaller values favor `model2`.

References

Meng X, Wong WH (1996). "Simulating Ratios of Normalizing Constants via a Simple Identity: A Theoretical Exploration." *Statistica Sinica*, **6**(4), 831–860. https://www3.stat.sinica.edu.tw/statistica/j6n4/j6n43/j6n43.htm.

Frühwirth-Schnatter S (2004). "Estimating Marginal Likelihoods for Mixture and Markov Switching Models Using Bridge Sampling Techniques." *The Econometrics Journal*, **7**(1), 143–167. doi:10.1111/j.1368423X.2004.00125.x.

Gronau QF, Singmann H, Wagenmakers E (2020). "bridgesampling: An R Package for Estimating Normalizing Constants." *Journal of Statistical Software*, **92**(10), 1–29. doi:10.18637/jss.v092.i10.

Celeux G, Frühwirth-Schnatter S, Robert CP (2019). "Model Selection for Mixture Models – Perspectives and Strategies." In Frühwirth-Schnatter S, Celeux G, Robert CP (eds.), *Handbook of Mixture Analysis*, 117–154. Chapman and Hall/CRC. doi:10.1201/97804290559117.

Frühwirth-Schnatter S (2019). "Keeping the Balance – Bridge Sampling for Marginal Likelihood Estimation in Finite Mixture, Mixture of Experts and Markov Mixture Models." *Brazilian Journal of Probability and Statistics*, **33**(4), 706–733. doi:10.1214/19BJPS446.

Examples

```
### Simulate and fit the correctly specified model
set.seed(1)
correct_model <- fit(
  choice ~ x + z | 0,
  dgp_parameters = list(beta = c(x = 1, z = 0.5)),
  iterations = 500,
  chains = 1
)
simulated_data <- as.data.frame(correct_model$data)
```

```
### Fit the same data again, but omit the relevant regressor z
misspecified_model <- fit(
  choice ~ x | 0,
  data = simulated_data,
  iterations = 500,
  chains = 1
)

### A Bayes factor greater than one favors the correct first model
bayes_factor(correct_model, misspecified_model)
```

`class_updates` *Update latent classes*

Description

Low-level sampler kernels for class weights, allocations, sizes, the weight-based class update, and Dirichlet-process class updates.

Usage

```
sample_allocation(prob)

update_s(delta, m)

update_z(s, beta, b, Omega)

update_m(C, z, non_zero = FALSE)

update_classes_wb(
  s,
  b,
  Omega,
  epsmin = 0.01,
  epsmax = 0.7,
  deltamin = 0.1,
  deltashift = 0.5,
  identify_classes = FALSE,
  Cmax = 10L
)

update_classes_dp(
  beta,
  z,
  b,
  Omega,
  delta,
  mu_b_0,
```

```
  Sigma_b_0,
  n_Omega_0,
  V_Omega_0,
  identify_classes = FALSE,
  Cmax = 10L
)
```

Arguments

`prob`	[`numeric(C)`] Class probabilities.
`delta`	[`numeric(1)`] Dirichlet concentration parameter.
`m`	[`numeric(C)`] Class sizes.
`s`	[`numeric(C)`] Class weights.
`beta`	[`matrix(P, N)`] Individual coefficient draws in columns.
`b`	[`matrix(P, C)`] Class means in columns.
`Omega`	[`matrix(P * P, C)`] Vectorized class covariance matrices in columns.
`C`	[`integer(1)`] Number of classes.
`z`	[`numeric(N)`] Class allocations numbered from one to `C`.
`non_zero`	[`logical(1)`] Replace empty class sizes by one?
`epsmin`	[`numeric(1)`] Remove the smallest class when its weight is below this threshold.
`epsmax`	[`numeric(1)`] Split the largest class when its weight exceeds this threshold.
`deltamin`	[`numeric(1)`] Merge the two closest classes when the Euclidean distance between their means is below this threshold.
`deltashift`	[`numeric(1)`] Scale of the mean displacement along the leading covariance eigenvector after splitting a class.
`identify_classes`	[`logical(1)`] Order the current active classes by size?
`Cmax`	[`integer(1)`] Maximum number of classes.

`mu_b_0` [numeric(P)]
Prior mean for class means.

`Sigma_b_0` [matrix(P, P)]
Prior covariance for class means.

`n_Omega_0` [integer(1)]
Prior degrees of freedom for class covariances.

`V_Omega_0` [matrix(P, P)]
Prior scale matrix for class covariances.

Value

The functions return one sampler update:

- `sample_allocation()`: an integer class label.
- `update_s()`: a `C` by 1 numeric matrix of class weights.
- `update_z()`: an `N` by 1 numeric matrix of allocations.
- `update_m()`: a `C` by 1 numeric matrix of class sizes.
- `update_classes_wb()`: a list with `s`, `b`, `Omega`, and `update_type`, where the latter is zero for no update, one for removal, two for splitting, and three for merging.
- `update_classes_dp()`: a list with `z`, `b`, `Omega`, and `C`.

References

Neal RM (2000). "Markov Chain Sampling Methods for Dirichlet Process Mixture Models." *Journal of Computational and Graphical Statistics*, **9**(2), 249–265. doi:10.1080/10618600.2000.10474879.

Oelschläger L, Bauer D (2021). "Bayes Estimation of Latent Class Mixed Multinomial Probit Models." In *Proceedings of the 100th Annual Meeting of the Transportation Research Board*. https://trid.trb.org/view/1759753.

Examples

```
### a latent class state of six deciders with one random coefficient
set.seed(1)
beta <- matrix(c(-1, -1.2, -0.8, 1, 1.3, 0.9), nrow = 1)
b <- matrix(c(-1, 1), nrow = 1)
Omega <- matrix(c(0.2, 0.2), nrow = 1)

### the weights, the allocations, and the class sizes are updated in turn
s <- update_s(delta = 1, m = c(3, 3))
z <- update_z(s, beta, b, Omega)
m <- update_m(C = 2, z = z)
sample_allocation(c(0.5, 0.3, 0.2))

### the weight-based update splits a class that grew too large
update_classes_wb(s = c(0.9, 0.1), b = b, Omega = Omega)

### the Dirichlet process update draws the class count from the data
update_classes_dp(
```

```
  beta = beta, z = z, b = b, Omega = Omega, delta = 1,
  mu_b_0 = 0, Sigma_b_0 = diag(1), n_Omega_0 = 4, V_Omega_0 = diag(1)
)
```

`coef.RprobitB_fit` *Extract posterior coefficient summaries*

Description

Returns posterior means or medians for population parameters or individual random coefficients.

Usage

```
## S3 method for class 'RprobitB_fit'
coef(
  object,
  type = c("mean", "median"),
  level = c("population", "individual"),
  ...
)
```

Arguments

`object`	[`RprobitB_fit`] Fitted choice model.
`type`	[`character(1)`] The posterior summary to return: • `"mean"` averages the draws. • `"median"` is more robust for skewed posteriors.
`level`	[`character(1)`] Which parameters to return: • `"population"` returns the parameters shared by all deciders. • `"individual"` returns the random coefficients of every decider, which requires a mixed model fitted with `save_individual_draws = TRUE`.
`...`	Currently not used.

Value

For `level = "population"`, a named numeric vector with one value per global posterior variable that is not fixed by the normalization or the model structure. For `level = "individual"`, a numeric matrix with deciders in rows and random effects in columns. Log-normal coefficients remain on their latent normal scale.

Examples

```
set.seed(1)
model <- fit(
  choice ~ x | 0,
  random_effects = "x",
  dgp_parameters = list(beta = c(x = 1), Omega = matrix(0.5)),
  chains = 1,
  save_individual_draws = TRUE
)
coef(model)
head(coef(model, level = "individual"))
```

`coefficient_updates` *Update coefficient distributions*

Description

Low-level Gibbs sampler kernels for coefficients, their class means, and covariance matrices.

Usage

```
update_coefficient(mu_beta_0, Sigma_beta_0_inv, XSigX, XSigU)

update_b_c(bar_b_c, Omega_c, m_c, Sigma_b_0_inv, mu_b_0)

update_b(beta, Omega, z, m, Sigma_b_0_inv, mu_b_0)

update_Omega_c(S_c, m_c, n_Omega_0, V_Omega_0, correlated)

update_Omega(beta, b, z, m, n_Omega_0, V_Omega_0, correlated = NULL)
```

Arguments

`mu_beta_0`	[`numeric(P)`] Prior mean for a coefficient vector.
`Sigma_beta_0_inv`	[`matrix(P, P)`] Prior precision matrix for a coefficient vector.
`XSigX`	[`matrix(P, P)`] Sum of design cross-products weighted by inverse error covariance.
`XSigU`	[`numeric(P)`] Sum of design-utility products weighted by inverse error covariance.
`bar_b_c`	[`numeric(P)`] Average individual coefficient vector in one class.
`Omega_c`	[`matrix(P, P)`] Covariance matrix of one class.

`m_c` [integer(1)]
Size of one class.

`Sigma_b_0_inv` [matrix(P, P)]
Prior precision matrix for class means.

`mu_b_0` [numeric(P)]
Prior mean for class means.

`beta` [matrix(P, N)]
Individual coefficient draws in columns.

`Omega` [matrix(P * P, C)]
Vectorized class covariance matrices in columns.

`z` [numeric(N)]
Class allocations numbered from one to `C`.

`m` [numeric(C)]
Class sizes.

`S_c` [matrix(P, P)]
Scatter matrix for one class.

`n_Omega_0` [integer(1)]
Prior degrees of freedom for class covariances.

`V_Omega_0` [matrix(P, P)]
Prior scale matrix for class covariances.

`correlated` [logical(P) | NULL]
Which random effects are correlated. Covariances between the other effects are zero. In `update_Omega()`, the default (`NULL`) correlates all random effects.

`b` [matrix(P, C)]
Class means in columns.

Value

The functions return one sampler update:

- `update_b_c()`: a `P` by 1 numeric matrix containing a class mean.
- `update_b()`: a `P` by `C` matrix of class means.
- `update_Omega_c()`: a `P` by `P` class covariance matrix.
- `update_Omega()`: a `P * P` by `C` matrix of vectorized covariances.
- `update_coefficient()`: a `P` by 1 numeric coefficient matrix.

Examples

```
### four deciders with one random coefficient in two classes
set.seed(1)
beta <- matrix(c(-1, -1.2, 1, 1.3), nrow = 1)
Omega <- matrix(c(0.2, 0.2), nrow = 1)
z <- c(1, 1, 2, 2)
m <- c(2, 2)
```

```
### a coefficient from its conditional posterior
update_coefficient(c(0, 0), diag(2), diag(2), c(0, 0))

### the class means, for one class and for all classes at once
update_b_c(
  bar_b_c = c(0, 0), Omega_c = diag(2), m_c = 4,
  Sigma_b_0_inv = diag(2), mu_b_0 = c(0, 0)
)
update_b(beta, Omega, z, m, Sigma_b_0_inv = diag(1), mu_b_0 = 0)

### the class covariances
update_Omega_c(
  S_c = diag(2), m_c = 4, n_Omega_0 = 4, V_Omega_0 = diag(2),
  correlated = c(TRUE, TRUE)
)
update_Omega(
  beta, b = matrix(c(-1, 1), nrow = 1), z, m,
  n_Omega_0 = 4, V_Omega_0 = diag(1)
)
```

`confint.RprobitB_fit` *Compute posterior credible intervals*

Description

Computes equal-tailed posterior intervals for selected model variables.

Usage

```
## S3 method for class 'RprobitB_fit'
confint(object, parm = NULL, level = 0.95, ...)
```

Arguments

`object`	[`RprobitB_fit`] Fitted choice model.
`parm`	[`character()`	`NULL`] Variables to include. `NULL` includes every population-level posterior variable that is not fixed by the normalization or the model structure. Individual coefficients can be selected by their `individual[effect,decider]` names.
`level`	[`numeric(1)`] Probability of the equal-tailed credible intervals.
`...`	Currently not used.

Details

The returned intervals are credible intervals, not confidence intervals. They are reported through `stats::confint()` because it is the standard way to ask a fitted model for interval estimates.

Value

A numeric matrix with one row per selected variable and columns for the lower and upper credible limits.

Examples

```
set.seed(1)
model <- fit(
  choice ~ x | 0, dgp_parameters = list(beta = c(x = 1)), chains = 1
)
confint(model)
```

`fit` *Fit a Bayesian probit choice model*

Description

Fits a Bayesian probit choice model to a `data.frame` of choice data. If `data = NULL`, probit choice data are simulated with the **choicedata** package before estimation.

Usage

```
fit(
  formula,
  data = NULL,
  random_effects = character(),
  latent_class_effects = character(),
  alternatives = NULL,
  base = NULL,
  choice_type = c("unordered", "ordered", "ranked"),
  format = c("wide", "long"),
  column_decider = "deciderID",
  column_occasion = NULL,
  column_alternative = NULL,
  delimiter = "_",
  scale = NULL,
  prior = NULL,
  classes = 1L,
  class_update = c("fixed", "sparse", "dirichlet_process", "weight_based"),
  max_classes = 10L,
  weight_based_control = NULL,
  iterations = 1000L,
  warmup = iterations%/%2L,
  thin = 1L,
  chains = 4L,
  save_individual_draws = FALSE,
  n_deciders = 100L,
```

```
  n_occasions = 1L,
  n_alternatives = NULL,
  covariates = NULL,
  dgp_parameters = NULL,
  progress = interactive()
)
```

Arguments

`formula` [formula]
A symbolic description of the choice model, see the details on specifying the model formula.

`data` [data.frame | NULL]
Empirical choice data. NULL simulates data before fitting. In long format, a choice occasion may list only its available alternatives, see the details on individual choice sets.

`random_effects` [character()]
Named vector defining random effects, see the details on specifying random effects.

`latent_class_effects`
[character()]
Names of covariates whose effects differ between latent classes, see the details on specifying latent class effects.

`alternatives` [character() | NULL]
Alternative labels. Required if choice_type = "ordered" and data is supplied, then in increasing order of the response levels. Otherwise, NULL takes them from data or, if data = NULL, uses capital letters.

`base` [character(1) | NULL]
The alternative whose utility is subtracted from all others, see the details on the normalization. Coefficients that vary across alternatives are then expressed relative to it, and none is estimated for it. NULL uses the first alternative in alphabetical order. Not used if choice_type = "ordered", which has a single utility per occasion.

`choice_type` [character(1)]
What the response records, and how the model explains it:

- "unordered": the chosen alternative. Every alternative has its own utility, and the alternative with the greatest utility is chosen.
- "ordered": a level of the ordered scale given by alternatives. One utility per choice occasion is compared with increasing thresholds, and the level is the interval it falls into.
- "ranked": a complete ranking of the alternatives. Every alternative has its own utility, and the ranking orders them by utility.

`format` [character(1)]
The layout of data:

- "wide" has one row per choice occasion, where covariate columns that vary across alternatives end in the alternative name.

- "long" has one row per choice occasion and alternative, named in column_alternative.

column_decider [character(1) | NULL]
Column name with decider identifiers. NULL treats every row of wide data as its own decider and adds the identifiers as column deciderID.

column_occasion
[character(1) | NULL]
Column name with occasion identifiers. Set to NULL in cross-sectional data.

column_alternative
[character(1) | NULL]
Column name with alternative identifiers when format = "long".

delimiter [character(1)]
Delimiter separating alternative identifiers from covariate names when format = "wide".

scale [NULL | named numeric(1)]
Utility scale normalization, see the details on the normalization. NULL fixes the error variance of the first utility difference, or of the single utility if choice_type = "ordered", to one. Otherwise one named value:

- c(<effect> = <value>) fixes a non-random coefficient without latent classes, e.g. scale = c(price = -1).
- c("Sigma_<alternative>,<alternative>" = <value>) fixes the error variance of the utility difference between a non-base alternative and the base alternative to a positive value, e.g. scale = c("Sigma_B,B" = 1).
- c(Sigma = <value>) fixes the error variance if choice_type = "ordered".

prior [named list() | NULL]
Parameters of the prior distributions that replace the defaults, see the details on the prior distribution for the component names and default values.

classes [integer(1)]
Number of latent classes between which the latent_class_effects differ. For class_update = "dirichlet_process" or "weight_based", this is the number of classes the sampler starts from.

class_update [character(1)]
Mixture specification:

- "fixed" fits a finite mixture with classes classes.
- "sparse" fits an overfitted finite mixture with classes classes and infers the occupied number through a sparse Dirichlet weight prior.
- "dirichlet_process" samples the occupied class count with a Dirichlet process allocation sampler.
- "weight_based" adapts the number of classes during warmup with a weight-threshold split, removal, and merge heuristic.

max_classes [integer(1)]
Largest number of latent classes the sampler may reach when class_update = "dirichlet_process" or "weight_based" changes their number. fit() warns if the retained draws reach it, then refit with a larger value.

`weight_based_control`
: [named list() | NULL]
Tuning constants of `class_update = "weight_based"`, see the details on the number of latent classes. `NULL` uses the defaults:

- `buffer = 50`: minimum number of iterations between two updates.
- `epsmin = 0.01`: remove the smallest class if its weight falls below this value.
- `epsmax = 0.7`: split the largest class if its weight exceeds this value.
- `deltamin = 0.1`: merge the closest pair of classes if the distance of their means falls below this value.
- `deltashift = 0.5`: displacement of the two means after a split, in within-class standard deviations.

`iterations`
: [integer(1)]
Total MCMC iterations per chain, including warmup.

`warmup`
: [integer(1)]
Initial iterations discarded from each chain.

`thin`
: [integer(1)]
Interval between retained post-warmup draws.

`chains`
: [integer(1)]
Number of independent MCMC chains.

`save_individual_draws`
: [logical(1)]
Retain the posterior draws of the individual random coefficients? They are required for `coef(level = "individual")` and `predict(type = "conditional")`.
Enable this only when you need them, because they can dominate the memory the fitted object occupies.

`n_deciders`
: [integer(1)]
Number of deciders to simulate when `data = NULL`.

`n_occasions`
: [integer(1) | integer(n_deciders)]
Simulated occasions for each decider.

`n_alternatives`
: [integer(1) | NULL]
Number of simulated alternatives. `NULL` uses `length(alternatives)` or, if `alternatives = NULL`, two unordered or three ordered or ranked alternatives labeled with capital letters.

`covariates`
: [named list() | NULL]
Optional covariate values for the simulated data. Names are covariate columns of the simulated wide `data.frame`, such as `price_A`, and each element is a vector with one value per simulated choice occasion. Unspecified covariates are generated by `choicedata::generate_choice_covariates()`.

`dgp_parameters`
: [named list() | NULL]
The parameters that generate the simulated data, named like the arguments of `choicedata::choice_parameters()`:

- `beta`: the coefficient vector, or a list of one vector per latent class. A named vector is matched to the effects by name and may omit effects, whose coefficients are then drawn.

- `Omega`: the covariance matrix of the random effects, or a list of one matrix per latent class.
- `Sigma`: the error covariance matrix, or the error variance if `choice_type = "ordered"`.
- `gamma`: the thresholds if `choice_type = "ordered"`.
- `weights`: the class weights if `classes > 1`.

Unspecified parameters are drawn at random.

`progress` [`logical(1)`]
Show progress?

Value

An `RprobitB_fit` object, which is a `list` with the components:

- `call`: the matched call.
- `data`: the data used.
- `model`: the model specification. If `class_update` changes the number of classes, `model$latent_classes$history` holds the number of classes at every iteration of every chain, and for `"weight_based"`, `model$latent_classes$changes` holds the change of the heuristic at an iteration (0 none, 1 removal, 2 split, 3 merge).
- `prior`: the prior specification.
- `draws`: the posterior draws.
- `sampler`: iterations, warmup, thinning, chains, and elapsed times.
- `simulation`: the `dgp_parameters` that generated the data and the simulation sizes if `data = NULL`, otherwise `NULL`.

Normalization

Utilities are identified only up to level and scale. `base` selects the alternative whose utility is subtracted from all others, and `scale` fixes one parameter to identify the scale. The sampler fixes the error variance of the first utility difference at one and draws the error covariance by the marginal data augmentation of Imai and van Dyk (2005), which expands the scale of the latent utilities in every iteration and returns to the fixed one afterwards, so that the covariance and the coefficients move more freely. Every retained draw is then rescaled to the normalization in `scale`.

Individual choice sets

Unordered choices may be made from occasion-specific subsets of the alternatives. In long format, an occasion lists only the rows of its available alternatives. The latent utilities of unavailable alternatives are imputed from their conditional distribution without a truncation, so they do not restrict the choice. Predictions assign probability zero to unavailable alternatives. Ordered and ranked models require complete choice sets.

Random effects and mixture models

An unnamed `random_effects` vector is a shorthand for correlated normal effects, so `random_effects = c("price", "time")` is the same as `random_effects = c(price = "cn", time = "cn")`.

A mixture model divides the deciders into `classes` latent classes and allocates every decider to exactly one of them. `latent_class_effects` decides which mixture model this is:

- A random effect named there follows a class-specific normal or log-normal distribution. This is the latent class mixed probit model, reported as `mu[<effect>,<class>]` and `Omega[<effect>,<effect>,<class>]`.
- An effect that is named there but has no random effect is a single coefficient per class. This is the classical latent class model, reported as `beta[<effect>,<class>]`.

Number of latent classes

`class_update` decides how many of the latent classes are used:

- `"fixed"` keeps all `classes` classes occupied, so the posterior is the finite-mixture posterior given that all of them are used.
- `"sparse"` starts from `classes` classes, deliberately more than expected, and empties the superfluous ones through a small symmetric Dirichlet weight prior (Rousseau and Mengersen 2011; Malsiner-Walli et al. 2016; Frühwirth-Schnatter and Malsiner-Walli 2019).
- `"dirichlet_process"` creates and removes occupied classes with Neal's (2000) auxiliary-parameter allocation update, up to `max_classes`. With a gamma hyperprior, its precision is updated with the beta-gamma augmentation of Escobar and West (1995).
- `"weight_based"` splits, removes, and merges classes during warmup and then keeps their number fixed. Every `buffer` iterations it removes the class below `epsmin`, splits the class above `epsmax`, or merges the closest pair of class means below `deltamin`, attempting at most one change in that order. A split moves the two means by `deltashift` times the leading within-class standard deviation. Updates stop after warmup, and retained iterations condition on the dimension selected separately by each chain.

Class labels

Numbering the latent classes differently describes the same mixture, every fit with more than one possible class therefore relabels its retained draws before they are summarized. The representative assignment of deciders to classes is the draw that is closest to the posterior co-clustering matrix in the least-squares sense (Dahl 2006). Every draw is then renumbered to agree with it, using the equivalence classes representatives assignment of Papastamoulis and Iliopoulos (2010), and the same permutation is applied to weights, means, covariances, and allocations. Finally the classes are numbered by decreasing posterior mean weight.

Prior distribution

The prior is conjugate where available; the finite-mixture concentration is updated by a log-scale Metropolis-Hastings step when it has a gamma hyperprior. `prior` is a named list that overrides the following defaults, where `P_f`, `P_l`, and `P_r` count the fixed effects without latent classes, the fixed effects with latent classes, and the random effects, and `J` the alternatives:

- `fixed_mean` [`numeric(P_f)`] and `fixed_covariance` [`matrix(P_f, P_f)`]: normal prior for the fixed coefficients without latent classes, default `rep(0, P_f)` and `10 * diag(P_f)`.

- latent_class_mean [numeric(P_l)] and latent_class_covariance [matrix(P_l, P_l)]: normal prior for the class-specific values of the latent class effects, the same for every class, default rep(0, P_l) and 10 * diag(P_l).
- random_mean [numeric(P_r)] and random_mean_covariance [matrix(P_r, P_r)]: normal prior for the means of the random coefficients, default rep(0, P_r) and 10 * diag(P_r).
- random_covariance_df [integer(1)] and random_covariance_scale [matrix(P_r, P_r)]: inverse Wishart prior for the covariance matrices of the random coefficients, default P_r + 2 and diag(P_r). Entries between uncorrelated random effects must be zero. In a mixture, both priors apply to the block of the random effects with latent classes in every class and to the block without latent classes once.
- class_concentration [numeric(1) | named numeric(2)]: fixed symmetric Dirichlet concentration for finite weights or precision of the Dirichlet process. A named c(shape = ..., rate = ...) instead places a gamma hyperprior on it. Defaults are 1 for a fixed finite or weight-based mixture, c(shape = 1, rate = 200) for a sparse finite mixture, and c(shape = 2, rate = 4) for a Dirichlet process mixture.
- error_covariance_df [integer(1)] and error_covariance_scale [matrix(J - 1, J - 1)]: inverse Wishart prior for the unrestricted error covariance of the utility differences, default J + 1 and diag(J - 1). The prior of the identified covariance is that of the unrestricted covariance divided by its first diagonal element. Not used for ordered models.
- threshold_mean [numeric(J - 2)] and threshold_covariance [matrix(J - 2, J - 2)]: normal prior for the logarithmic increments between the ordered thresholds, default rep(0, J - 2) and diag(J - 2). Only used for ordered models.

Specifying the model formula

The structure of formula is choice ~ A | B | C, i.e., a standard formula object but with three parts on the right-hand side, separated by |, where

- choice is the name of the discrete response variable,
- A are names of **alternative-specific covariates** with **a coefficient that is constant across alternatives**,
- B are names of **covariates that are constant across alternatives**,
- and C are names of **alternative-specific covariates** with **alternative-specific coefficients**.

The following rules apply:

1. By default, intercepts (referred to as alternative-specific constants, ASCs) are added to the model. They can be removed by adding + 0 in the second part, e.g., choice ~ A | B + 0 | C. To not include any covariates of the second type but to estimate ASCs, add 1 in the second part, e.g., choice ~ A | 1 | C. The expression choice ~ A | 0 | C is interpreted as no covariates of the second type and no ASCs.
2. To not include covariates of any type, add 0 in the respective part, e.g., choice ~ 0 | B | C.
3. Some parts of the formula can be omitted when there is no ambiguity. For example, choice ~ A is equivalent to choice ~ A | 1 | 0.
4. Multiple covariates in one part are separated by a + sign, e.g., choice ~ A1 + A2.

5. Arithmetic transformations of covariates in all three parts of the right-hand side are possible via the function `I()`, e.g., `choice ~ I(A1^2 + A2 * 2)`. In this case, a random effect can be defined for the transformed covariate, e.g., `random_effects = c("I(A1^2 + A2 * 2)" = "cn")`.
6. Ordered choice models have a single utility per choice occasion. Their covariates must be placed in the first part, e.g., `choice ~ age + income`. The intercept (named `"ASC"`) is included by default and removed by `0` in the second part.

Specifying random effects

Specify random effects as `"<covariate>" = "<distribution>"`. Each covariate must appear explicitly on the right-hand side of `formula`; use `"ASC"` for alternative-specific constants.

Available distributions are:

- `"cn"`: correlated normal
- `"n"`: uncorrelated normal
- `"cln"`: positively signed correlated log-normal
- `"ln"`: positively signed uncorrelated log-normal
- `"cln-"`: negatively signed correlated log-normal
- `"ln-"`: negatively signed uncorrelated log-normal

Specifying latent class effects

The covariates in `latent_class_effects` have effects that differ between the latent classes of a mixture model; use `"ASC"` for alternative-specific constants. A random effect named here has a class-specific mean and covariance, any other effect a class-specific coefficient. Effects that are not named are the same in every class, and random effects with and without latent class effects are uncorrelated. A model with more than one latent class needs at least one latent class effect.

References

Dahl DB (2006). "Model-Based Clustering for Expression Data via a Dirichlet Process Mixture Model." In Do K, Müller P, Vannucci M (eds.), *Bayesian Inference for Gene Expression and Proteomics*, 201–218. Cambridge University Press. doi:10.1017/CBO9780511584589.011.

Escobar MD, West M (1995). "Bayesian Density Estimation and Inference Using Mixtures." *Journal of the American Statistical Association*, **90**(430), 577–588. doi:10.1080/01621459.1995.10476550.

Malsiner-Walli G, Frühwirth-Schnatter S, Grün B (2016). "Model-Based Clustering Based on Sparse Finite Gaussian Mixtures." *Statistics and Computing*, **26**(1–2), 303–324. doi:10.1007/s1122201495002.

Frühwirth-Schnatter S, Malsiner-Walli G (2019). "From Here to Infinity: Sparse Finite versus Dirichlet Process Mixtures in Model-Based Clustering." *Advances in Data Analysis and Classification*, **13**(1), 33–64. doi:10.1007/s116340180329y.

Greene WH, Hensher DA (2003). "A latent class model for discrete choice analysis: contrasts with mixed logit." *Transportation Research Part B: Methodological*, **37**(8), 681–698. doi:10.1016/S01912615(02)000462.

Imai K, van Dyk DA (2005). "A Bayesian Analysis of the Multinomial Probit Model Using Marginal Data Augmentation." *Journal of Econometrics*, **124**(2), 311–334. doi:10.1016/j.jeconom.2004.02.002.

Neal RM (2000). "Markov Chain Sampling Methods for Dirichlet Process Mixture Models." *Journal of Computational and Graphical Statistics*, **9**(2), 249–265. doi:10.1080/10618600.2000.10474879.

Oelschläger L, Bauer D (2021). "Bayes Estimation of Latent Class Mixed Multinomial Probit Models." In *Proceedings of the 100th Annual Meeting of the Transportation Research Board*. https://trid.trb.org/view/1759753.

Oelschläger L (2026). *Overcoming Challenges in Modeling Choice Behavior Heterogeneity*. Ph.D. thesis, Bielefeld University, Bielefeld, Germany. https://pub.uni-bielefeld.de/record/3014719.

Papastamoulis P, Iliopoulos G (2010). "An Artificial Allocations Based Solution to the Label Switching Problem in Bayesian Analysis of Mixtures of Distributions." *Journal of Computational and Graphical Statistics*, **19**(2), 313–331. doi:10.1198/jcgs.2010.09008.

Rousseau J, Mengersen K (2011). "Asymptotic Behaviour of the Posterior Distribution in Overfitted Mixture Models." *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, **73**(5), 689–710. doi:10.1111/j.14679868.2011.00781.x.

Examples

```
### Fit a probit model to panel choice data
data("Train", package = "mlogit")
Train$price_A <- Train$price_A / 100 / 2.20371 # price in Euro
Train$price_B <- Train$price_B / 100 / 2.20371
Train$time_A <- Train$time_A / 60 # time in hours
Train$time_B <- Train$time_B / 60
model <- fit(
  choice ~ price + time + change + factor(comfort) | 0,
  data = Train,
  column_decider = "id",
  column_occasion = "choiceid",
  scale = c(price = -1), # other coefficients are willingness-to-pay
  chains = 1
)
summary(model)
interpret(model)

### Simulate choice data and compare the estimates with the truth
set.seed(1)
simulated <- fit(
  choice ~ x | y,
  dgp_parameters = list(beta = c(x = 1, y_B = -1)),
  covariates = list(y = rpois(400, lambda = 3)),
  iterations = 2000,
  chains = 1,
  n_deciders = 400
)
head(model.frame(simulated))
summary(simulated)
confint(simulated)
```

formula.RprobitB_fit *Extract the fitted formula*

Description

Returns the normalized three-part model formula stored in a fitted model.

Usage

```
## S3 method for class 'RprobitB_fit'
formula(x, ...)
```

Arguments

x	[RprobitB_fit] Fitted choice model.
...	Currently not used.

Value

A formula object.

Examples

```
set.seed(1)
model <- fit(
  choice ~ x, dgp_parameters = list(beta = c(x = 1, ASC_B = -0.5)),
  chains = 1
)
formula(model)
```

interpret *Interpret the estimates of a fitted choice model*

Description

This function translates the posterior draws to scales that are easier to interpret:

- A compensation says how many units of a reference covariate, for example the price, are worth one unit of another covariate, leaving the utility and hence the choice probabilities unchanged.
- A marginal effect says by how much the choice probability of an alternative changes per unit of a covariate, computed by finite differences of the predicted probabilities.

Both are reported with their posterior uncertainty.

Usage

```
interpret(
  object,
  type = c("compensation", "ame", "mea"),
  reference = NULL,
  effects = NULL,
  at = NULL,
  level = 0.95,
  progress = interactive()
)

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

Arguments

object	[RprobitB_fit] Fitted choice model.
type	[character(1)] The quantity to compute: • "compensation": how many units of reference compensate one additional unit of every other effect, so that the utility stays the same. With the price as reference, this is the willingness to pay. • "ame": the average marginal effect, that is, the derivative of the choice probability of an alternative with respect to a covariate, computed for every observed choice occasion and averaged. • "mea": the marginal effect at the average, that is, the same derivative for a single occasion whose covariates equal the averages of the observed ones.
reference	[character(1) | NULL] The effect in whose units compensations are measured. NULL uses the coefficient that scale fixed when fitting, and requires a choice otherwise. Only used if type = "compensation".
effects	[character() | NULL] The effects to express in units of reference. NULL uses every effect other than reference. Only used if type = "compensation".
at	[named numeric() | NULL] Covariate values at which the marginal effects of type = "mea" are evaluated, see the details. NULL uses the average of every covariate.
level	[numeric(1)] Probability of the equal-tailed posterior credible interval.
progress	[logical(1)] Show progress?
x	[RprobitB_interpretation] Output of interpret().
digits	[integer(1)] Number of significant digits to print.

... Currently not used.

Value

A `data.frame` of class `RprobitB_interpretation` with one row per quantity and the columns `mean`, `sd`, `lower`, and `upper` of its posterior distribution. Compensations have a column `effect` and, for a mixture model, a column `class`; marginal effects have the columns `covariate` and `alternative`, and for `type = "mea"` the column `at`.

Compensations

The utility of an alternative is linear in the covariates, so an increase of one unit in an effect with coefficient β_j is compensated by a change of $-\beta_j/\beta_k$ units in the reference effect with coefficient β_k. For a random effect, the coefficient of the median decider enters the ratio. Mixture models report one compensation per class. The ratio is computed for every posterior draw, so the reported uncertainty is the posterior uncertainty of the ratio. Compensations are free of the utility scale normalization, which makes them comparable across models.

Marginal effects

Marginal effects are derivatives of choice probabilities and are computed by finite differences of the predicted probabilities. Marginal effects of a mixed model refer to the population distribution of the random coefficients.

- `type = "ame"` averages the marginal effects over all observed choices.
- `type = "mea"` builds one artificial choice occasion whose covariates are the averages of the observed ones: the mean for numeric covariates and the most frequent value for the others. The argument `at` can be used to replace the average of the covariates.

Examples

```
### travel mode choice where travel time has an alternative-specific effect
data("TravelMode", package = "AER")
TravelMode$choice <- TravelMode$choice == "yes"
TravelMode$vcost <- TravelMode$vcost / 1.6196 # cost in Euro
set.seed(1)
model <- fit(
  choice ~ vcost | 1 | travel,
  data = TravelMode,
  format = "long",
  column_decider = "individual",
  column_alternative = "mode",
  scale = c(vcost = -1),
  iterations = 100,
  chains = 1
)

### travel time must be compensated far more in the plane than in the bus
interpret(
  model, type = "compensation", effects = c("travel_bus", "travel_air")
)
```

```
### the marginal effects at the average covariates, and for a short flight
interpret(model, type = "mea", at = c(travel_air = 40))
```

`latent_class_diagnostics` *Diagnose latent-class occupancy and membership*

Description

Computes label-invariant posterior summaries of a fitted mixture model: the distribution of the occupied class count and the posterior probability that every pair of deciders belongs to the same class. It also returns the probability of every decider belonging to each relabeled class.

Usage

```
latent_class_diagnostics(object)
```

Arguments

`object`	[RprobitB_fit] Fitted model with at least two finite classes, a sparse finite mixture, a Dirichlet-process mixture, or a weight-based heuristic fit.

Value

A list with three elements:

- `occupancy`: a `data.frame` with the columns `n_classes` and `probability`, with one row for every number of classes from one to the maximum number of classes of the model. It gives the share of draws in which exactly `n_classes` classes contain at least one decider, so it indicates how many classes the data support under the chosen prior. For a fixed mixture, the sampler keeps all classes occupied.
- `co_clustering`: a square matrix with one row and one column per decider. Entry `[i, j]` is the share of draws in which deciders `i` and `j` are in the same class, so it indicates whether two deciders behave alike. It does not depend on how the classes are labeled.
- `membership`: a matrix with one row per decider and one column per class, `class_1` to `class_<maximum>`. Entry `[i, k]` is the share of draws in which decider `i` is in class `k` after relabeling, so it answers which class a decider most likely belongs to.

References

Dahl DB (2006). "Model-Based Clustering for Expression Data via a Dirichlet Process Mixture Model." In Do K, Müller P, Vannucci M (eds.), *Bayesian Inference for Gene Expression and Proteomics*, 201–218. Cambridge University Press. doi:10.1017/CBO9780511584589.011.

Papastamoulis P, Iliopoulos G (2010). "An Artificial Allocations Based Solution to the Label Switching Problem in Bayesian Analysis of Mixtures of Distributions." *Journal of Computational and Graphical Statistics*, **19**(2), 313–331. doi:10.1198/jcgs.2010.09008.

Stephens M (2000). "Dealing with Label Switching in Mixture Models." *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, **62**(4), 795–809. doi:10.1111/14679868.00265.

Examples

```
set.seed(1)
model <- fit(
  choice ~ x | 0, random_effects = "x", latent_class_effects = "x",
  classes = 2, class_update = "dirichlet_process",
  dgp_parameters = list(
    beta = list(c(x = -1), c(x = 2)),
    Omega = list(matrix(0.2), matrix(0.2)),
    weights = c(0.6, 0.4)
  ),
  n_occasions = 5,
  chains = 1
)
diagnostics <- latent_class_diagnostics(model)
diagnostics$occupancy
diagnostics$co_clustering[1:5, 1:5]
head(diagnostics$membership)
```

---

logLik.RprobitB_fit *Extract the fitted log-likelihood*

Description

Evaluates the decider-level log-likelihood at the posterior mean parameters.

Usage

```
## S3 method for class 'RprobitB_fit'
logLik(object, ghk_draws = 500L, ...)
```

Arguments

`object`	[RprobitB_fit] Fitted choice model.
`ghk_draws`	[integer(1)] Number of draws of the GHK simulator for multivariate normal probabilities of more than three dimensions, see `oeli::pmvnorm()`.
`...`	Currently not used.

Value

A scalar object of class `logLik`. The `df` attribute counts the free population-level parameters after normalization, and `nobs` is the number of independent likelihood units as counted by `nobs()`.

Examples

```
set.seed(1)
model <- fit(
  choice ~ x | 0, dgp_parameters = list(beta = c(x = 1)), chains = 1
)
logLik(model)
```

---

loo *Compute approximate leave-one-out cross-validation*

Description

Computes Pareto-smoothed importance-sampling leave-one-out cross-validation using `loo::loo()`.

Usage

```
loo(x, ...)

## S3 method for class 'RprobitB_fit'
loo(x, ghk_draws = 500L, progress = interactive(), ...)
```

Arguments

`x`	[`RprobitB_fit`] Fitted choice model.
`...`	Further arguments passed to `loo::loo()`.
`ghk_draws`	[`integer(1)`] Number of draws of the GHK simulator for multivariate normal probabilities of more than three dimensions, see `oeli::pmvnorm()`.
`progress`	[`logical(1)`] Show progress?

Value

A `psis_loo` object from **loo**. It contains estimates and standard errors as well as one Pareto-k diagnostic per decider.

References

Vehtari A, Gelman A, Gabry J (2017). "Practical Bayesian Model Evaluation Using Leave-One-Out Cross-Validation and WAIC." *Statistics and Computing*, **27**(5), 1413–1432. doi:10.1007/s11222-01696964.

Vehtari A, Simpson D, Gelman A, Yao Y, Gabry J (2024). "Pareto Smoothed Importance Sampling." *Journal of Machine Learning Research*, **25**(72), 1–58. https://www.jmlr.org/papers/v25/19-556.html.

Examples

```
set.seed(1)
model <- fit(
  choice ~ x | 0, dgp_parameters = list(beta = c(x = 1)), n_occasions = 5,
  chains = 1
)
loo(model)
```

`model.frame.RprobitB_fit` *Extract the fitted data*

Description

Returns the choice data stored in a fitted model as a `data.frame`.

Usage

```
## S3 method for class 'RprobitB_fit'
model.frame(formula, ...)
```

Arguments

`formula`	[RprobitB_fit] Fitted choice model.
`...`	Currently not used.

Value

A `data.frame` containing the fitted choice data.

Examples

```
set.seed(1)
model <- fit(
  choice ~ x, dgp_parameters = list(beta = c(x = 1, ASC_B = -0.5)),
  chains = 1
)
head(model.frame(model))
```

nobs.RprobitB_fit *Count independent likelihood units*

Description

Counts the observed choice occasions if the model has neither random effects nor latent classes, because the likelihood then factorizes over the occasions. Otherwise the occasions of a decider are dependent through the random coefficients or the class membership, and the deciders with at least one observed response are counted.

Usage

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

Arguments

`object`	[RprobitB_fit] Fitted choice model.
`...`	Currently not used.

Value

An `integer(1)` count.

Examples

```
set.seed(1)
model <- fit(
  choice ~ x, dgp_parameters = list(beta = c(x = 1, ASC_B = -0.5)),
  chains = 1
)
nobs(model)
```

plot.RprobitB_fit *Plot posterior draws*

Description

Creates a standard posterior diagnostic or uncertainty plot with **bayesplot**, or draws the estimated mixing distributions of the random effects.

Usage

```
## S3 method for class 'RprobitB_fit'
plot(
  x,
  y = NULL,
  type = c("trace", "rank", "acf", "density", "interval", "pairs", "mixture"),
  variables = NULL,
  ...
)
```

Arguments

x	[RprobitB_fit] Fitted choice model.
y	[NULL] Currently not used.
type	[character(1)] The plot to create: • "trace" draws the sampled values of each chain over the iterations. • "rank" compares the chains through the ranks of their draws. • "acf" draws the autocorrelation within each chain. • "density" overlays the marginal posterior density of each chain. • "interval" draws posterior point estimates with credible intervals. • "pairs" draws bivariate scatter plots of the variables. • "mixture" draws the estimated mixing distribution of every random effect, the population density of its coefficient averaged over the retained draws. For an effect with latent classes, it is the mixture of the class densities. For simulated data, a dashed line adds the mixing distribution that generated them.
variables	[character() | NULL] Posterior variables to include. NULL includes all varying model parameters and excludes individual coefficients and latent allocations. For type = "mixture", the names of random effects as in mu[<effect>], and NULL includes all random effects.
...	Further arguments passed to the selected **bayesplot** function. Not used for type = "mixture".

Value

A ggplot object or, for a pairs plot, a bayesplot_grid object.

Examples

```
set.seed(1)
model <- fit(
  choice ~ x + z | 0, dgp_parameters = list(beta = c(x = 1, z = -0.5)),
```

```
  chains = 2
)

### convergence and mixing of the chains
plot(model, type = "trace")
plot(model, type = "rank")
plot(model, type = "acf")

### marginal and joint posterior distributions
plot(model, type = "density")
plot(model, type = "interval")
plot(model, type = "pairs")

### estimated mixing distribution of a random coefficient
mixed <- fit(
  choice ~ x | 0, random_effects = "x",
  dgp_parameters = list(beta = c(x = 1), Omega = matrix(0.5)),
  chains = 1, n_occasions = 10
)
plot(mixed, type = "mixture")
```

`predict.RprobitB_fit` *Predict choices*

Description

Computes occasion-level choice probabilities over the retained posterior draws and predicts the alternative with the largest mean probability.

Usage

```
## S3 method for class 'RprobitB_fit'
predict(
  object,
  newdata = NULL,
  type = c("population", "conditional"),
  uncertainty = FALSE,
  level = 0.95,
  ghk_draws = 500L,
  progress = interactive(),
  ...
)
```

Arguments

`object`	[RprobitB_fit] Fitted choice model.

`newdata` [`data.frame` | `NULL`]
Data for prediction. `NULL` uses the fitted data. A response column is optional. In wide format, missing decider and occasion identifiers make every row a choice occasion of its own decider.

`type` [`character(1)`]
Which coefficients to predict with:

- `"population"` integrates over the estimated population distribution of the random coefficients and latent classes and applies to any decider.
- `"conditional"` uses the posterior random coefficients and class allocations of the deciders that were observed when fitting the model, which `newdata` must then name.

`uncertainty` [`logical(1)`]
Add posterior standard deviations and credible intervals?

`level` [`numeric(1)`]
Probability of the credible intervals.

`ghk_draws` [`integer(1)`]
Number of draws of the GHK simulator for multivariate normal probabilities of more than three dimensions, see `oeli::pmvnorm()`.

`progress` [`logical(1)`]
Show progress?

`...` Currently not used.

Details

Conditional prediction is based on the individual-level parameters (Train, 2009, Chapters 11 and 12): the posterior distribution of a decider's coefficients given their observed choices, which the Gibbs sampler provides as draws when the model is fitted with `save_individual_draws = TRUE`.

Value

A `data.frame` with one row per choice occasion, identifier columns, `.prediction`, and one `probability_*` column per alternative. If `uncertainty = TRUE`, `sd_*`, `lower_*`, and `upper_*` columns are added.

References

Train KE (2009). *Discrete Choice Methods with Simulation*, 2 edition. Cambridge University Press, Cambridge. doi:10.1017/CBO9780511805271.

Examples

```
set.seed(1)
model <- fit(
  choice ~ x | 0,
  random_effects = "x",
  dgp_parameters = list(beta = c(x = 1), Omega = matrix(0.5)),
  n_deciders = 20,
  iterations = 300,
  warmup = 150,
```

```
  chains = 1,
  save_individual_draws = TRUE
)
head(predict(model))
head(predict(model, type = "conditional"))
head(predict(model, uncertainty = TRUE))

### new choice occasions
new_data <- data.frame(deciderID = 21:22, x_A = c(1, -1), x_B = c(0, 0))
predict(model, newdata = new_data)
```

`print.RprobitB_fit` *Print a fitted choice model*

Description

Prints the model formula, data size, and retained posterior sample size.

Usage

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

Arguments

`x`	[RprobitB_fit] Fitted choice model.
`...`	Currently not used.

Value

`x`, invisibly.

Examples

```
set.seed(1)
model <- fit(
  choice ~ x, dgp_parameters = list(beta = c(x = 1, ASC_B = -0.5)),
  chains = 1
)
print(model)
```

print.summary.RprobitB_fit

Print a fitted model summary

Description

Prints sampling information followed by the posterior summary table.

Usage

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

Arguments

x	[summary.RprobitB_fit] Model summary returned by summary().
digits	[integer(1)] Number of significant digits to print.
...	Further arguments passed to print.data.frame().

Value

x, invisibly.

residuals.RprobitB_fit

Extract choice residuals

Description

Computes observed choice indicators minus posterior mean occasion-level choice probabilities.

Usage

```
## S3 method for class 'RprobitB_fit'
residuals(object, ...)
```

Arguments

object	[RprobitB_fit] Fitted choice model.
...	Further arguments passed to predict().

Value

A numeric matrix with one row per choice occasion and one column per alternative. Rows with a missing response contain NA. For ranked data, the indicator represents the first-ranked alternative.

Examples

```
set.seed(1)
model <- fit(
  choice ~ x | 0, dgp_parameters = list(beta = c(x = 1)), chains = 1
)
head(residuals(model))
```

---

summary.RprobitB_fit *Summarize a fitted choice model*

Description

Summarizes the marginal posterior distributions of the fitted model parameters with selectable statistics.

Usage

```
## S3 method for class 'RprobitB_fit'
summary(
  object,
  variables = NULL,
  statistics = c("mean", "mode", "sd", "rhat", "ess_bulk"),
  probs = NULL,
  ...
)
```

Arguments

object	[RprobitB_fit] Fitted choice model.
variables	[character() | NULL] Posterior variables to summarize.
statistics	[character()] Posterior statistics to report, in this order. Available are "mean", "median", "mode", "sd", "mcse_mean", "mcse_median", "mcse_sd", "rhat", "ess_bulk", and "ess_tail", see the details.
probs	[numeric() | NULL] Optional unique probabilities for posterior quantiles.
...	Currently not used.

Details

Every statistic describes the marginal posterior of one variable and is computed from the retained draws of all chains. For a class that is empty in some draws, the column `occupied` gives the share of draws in which it is occupied, `mcse_*`, `rhat`, and `ess_*` are `NA`, and the other statistics use only these draws. The statistics are:

- `mean`: the average of the draws, the usual point estimate.
- `median`: the median value of the draws. If it differs clearly from the mean, the posterior is skewed.
- `mode`: the most probable value. For continuous draws, it is the peak of a kernel density estimate; for integer-valued draws, such as the active class count, it is the most frequent value.
- `sd`: the standard deviation of the draws, the posterior uncertainty of the parameter.
- `q<100 * p>`: the quantile of probability `p`. With `probs = c(0.025, 0.975)`, the two columns are the limits of the equal-tailed 95% credible interval.
- `mcse_mean`, `mcse_median`, `mcse_sd`: the Monte Carlo standard errors of the mean, median, and standard deviation, the sampling error of these estimates that more iterations would reduce. Good values are below a tenth of `sd`; larger values mean that the reported digits are not yet reliable and the sampler should run longer.
- `rhat`: the larger of the rank-normalized split-R-hat and its folded version of Vehtari et al. (2021). It compares the variance between the halves of all chains with the variance within them. With a single chain, it compares the two halves of that chain. Good values are at most 1.01; larger values indicate that the chains have not mixed and the sampler should run longer, see `plot(type = "trace")`.
- `ess_bulk`: the effective sample size for the center of the posterior, the number of independent draws that carry the same information as the correlated draws. Good values are at least 100 per chain; smaller values mean that the sampler should run longer.
- `ess_tail`: the smaller of the effective sample sizes of the 5% and 95% quantiles. It governs the precision of extreme quantiles and credible intervals. The same rule applies: at least 100 per chain is good.

Value

A `summary.RprobitB_fit` object.

References

Vehtari A, Gelman A, Simpson D, Carpenter B, Bürkner P (2021). "Rank-Normalization, Folding, and Localization: An Improved $\widehat{R}$ for Assessing Convergence of MCMC." *Bayesian Analysis*, **16**(2), 667–718. doi:10.1214/20BA1221.

Examples

```
set.seed(1)
model <- fit(
  choice ~ x | 0, dgp_parameters = list(beta = c(x = 1)), chains = 1
)
summary(model)
```

`update.RprobitB_fit` *Update and refit a choice model*

Description

Refits a choice model with a modified specification.

Usage

```
## S3 method for class 'RprobitB_fit'
update(object, formula., ..., evaluate = TRUE)
```

Arguments

`object`	[`RprobitB_fit`] Fitted choice model.
`formula.`	[`formula`] Changes to the model formula, see the details.
`...`	Arguments of `fit()` that replace the ones of `object`.
`evaluate`	[`logical(1)`] Refit the model? If `FALSE`, the updated call is returned, where `data` stands for the choice data of `object`.

Details

Arguments that are not specified are taken from `object`.

The model formula is updated part by part, so `. ~ . | . + income` extends the covariates that are constant across alternatives and leaves the other two formula parts unchanged.

The choice data of `object` are reused, also if they were simulated, which makes the updated model comparable to `object`. Supply `data` to fit the updated model to other choice data.

Value

An object of class `RprobitB_fit`, or the updated `call` if `evaluate` is `FALSE`.

Examples

```
### simulate choice data and fit a model with two covariates
set.seed(1)
model <- fit(
  choice ~ x + y | 0, dgp_parameters = list(beta = c(x = 1, y = -0.5)),
  chains = 1
)
summary(model)

### drop `y` from the formula, the other formula parts stay as they are
model_2 <- update(model, . ~ . - y)
```

```
summary(model_2)

### let the coefficient of `x` vary across deciders instead
model_3 <- update(model, random_effects = "x")
summary(model_3)
```

`utility_updates` *Update utilities and thresholds*

Description

Low-level Gibbs sampler kernels for error covariance matrices, latent utilities, and ordered-response thresholds.

Usage

```
d_to_gamma(d)

log_likelihood_ordered(d, y, sys, Tvec)

update_Sigma(n_Sigma_0, V_Sigma_0, N, S)

update_U(U, y, sys, Sigma_inv, available = NULL)

update_U_ranked(U, sys, Sigma_inv)

update_d(d, y, sys, mu_d_0, Sigma_d_0, Tvec, step_scale)
```

Arguments

`d`	[`numeric(J - 2)`] Log-increments between finite ordered thresholds.
`y`	[`integer(1)` | `matrix(N, max(Tvec))`] Chosen alternative for `update_U()`, where `J` denotes the base alternative, or ordered responses by decider and occasion for the threshold functions.
`sys`	[`numeric(J - 1)` | `matrix(N, max(Tvec))`] Systematic utilities matching `U` or `y`.
`Tvec`	[`integer(N)`] Number of observed occasions for each decider.
`n_Sigma_0`	[`integer(1)`] Prior degrees of freedom for the error covariance.
`V_Sigma_0`	[`matrix(J - 1, J - 1)`] Prior scale matrix for the error covariance.
`N`	[`integer(1)`] Number of residual vectors summed in the scatter matrix `S`.

`S` [`matrix(J - 1, J - 1)`]
Error scatter matrix.

`U` [`numeric(J - 1)`]
Current latent utility differences.

`Sigma_inv` [`matrix(J - 1, J - 1)`]
Inverse error covariance matrix.

`available` [`logical(J) | NULL`]
Availability of the `J - 1` non-base alternatives followed by the base alternative. Utilities of unavailable alternatives are drawn without truncation. By default (`NULL`), all alternatives are available.

`mu_d_0` [`numeric(J - 2)`]
Prior mean for threshold log-increments.

`Sigma_d_0` [`matrix(J - 2, J - 2)`]
Prior covariance for threshold log-increments.

`step_scale` [`numeric(J - 2)`]
Random-walk proposal standard deviations, one per log-increment.

Value

The functions return one sampler update or transformation:

- `update_Sigma()`: a `J - 1` by `J - 1` covariance matrix.
- `update_U()` and `update_U_ranked()`: `J - 1` by 1 numeric latent utility matrices.
- `d_to_gamma()`: a numeric column matrix containing ordered thresholds and their infinite bounds.
- `log_likelihood_ordered()`: one numeric log-likelihood value.
- `update_d()`: a `J - 2` by 1 numeric matrix of updated log-increments.

References

Robert CP (1995). "Simulation of Truncated Normal Variables." *Statistics and Computing*, **5**(2), 121–125. doi:10.1007/BF00143942.

Examples

```
### two deciders who choose twice on an ordered scale of four levels
set.seed(1)
d <- c(0, log(2))
y <- matrix(c(1, 2, 3, 2), nrow = 2)
sys <- matrix(0, nrow = 2, ncol = 2)
Tvec <- c(2, 2)

### the thresholds, their likelihood, and their random-walk update
d_to_gamma(d)
log_likelihood_ordered(d, y, sys, Tvec)
update_d(
  d, y, sys, mu_d_0 = c(0, 0), Sigma_d_0 = diag(2), Tvec = Tvec,
```

```
  step_scale = c(0.1, 0.1)
)

### the latent utilities of an unordered and of a ranked choice
update_U(
  c(0, 0), y = 1, sys = c(0, 0), Sigma_inv = diag(2),
  available = c(TRUE, TRUE, TRUE)
)
update_U_ranked(c(0, 0), sys = c(0, 0), Sigma_inv = diag(2))

### the error covariance
update_Sigma(n_Sigma_0 = 4, V_Sigma_0 = diag(2), N = 10, S = diag(2))
```

`vcov.RprobitB_fit` *Extract the posterior covariance matrix*

Description

Computes covariance across all retained draws of global model variables.

Usage

```
## S3 method for class 'RprobitB_fit'
vcov(object, ...)
```

Arguments

`object`	[`RprobitB_fit`] Fitted choice model.
`...`	Currently not used.

Details

The returned matrix is the covariance of the posterior distribution, not the sampling covariance of an estimator. It is reported through `stats::vcov()` because it is the standard way to ask a fitted model for the covariance of its parameters.

Value

A symmetric numeric matrix whose rows and columns are the global posterior variables returned by `coef()`.

Examples

```
set.seed(1)
model <- fit(choice ~ x + y | 0, chains = 1)
vcov(model)
```

WAIC *Compute the widely applicable information criterion*

Description

Computes WAIC from posterior log-likelihood draws using `loo::waic()`.

Usage

```
WAIC(object, ghk_draws = 500L, progress = interactive(), ...)
```

Arguments

`object`	[RprobitB_fit] Fitted choice model.
`ghk_draws`	[integer(1)] Number of draws of the GHK simulator for multivariate normal probabilities of more than three dimensions, see `oeli::pmvnorm()`.
`progress`	[logical(1)] Show progress?
`...`	Further arguments passed to `loo::waic()`.

Value

A `waic` object from **loo**. Its `estimates` matrix contains WAIC, effective parameter counts, and their standard errors.

References

Watanabe S (2010). "Asymptotic Equivalence of Bayes Cross Validation and Widely Applicable Information Criterion in Singular Learning Theory." *Journal of Machine Learning Research*, **11**, 3571–3594. https://www.jmlr.org/papers/v11/watanabe10a.html.

Vehtari A, Gelman A, Gabry J (2017). "Practical Bayesian Model Evaluation Using Leave-One-Out Cross-Validation and WAIC." *Statistics and Computing*, **27**(5), 1413–1432. doi:10.1007/s11222-01696964.

Examples

```
set.seed(1)
model <- fit(
  choice ~ x | 0, dgp_parameters = list(beta = c(x = 1)), n_occasions = 5,
  chains = 1
)
WAIC(model)
```

Index

∗ **hplot**
plot.RprobitB_fit, 28
∗ **models**
as_draws.RprobitB_fit, 3
bayes_factor, 3
class_updates, 5
coef.RprobitB_fit, 8
coefficient_updates, 9
confint.RprobitB_fit, 11
fit, 12
formula.RprobitB_fit, 21
interpret, 21
latent_class_diagnostics, 24
logLik.RprobitB_fit, 25
loo, 26
model.frame.RprobitB_fit, 27
nobs.RprobitB_fit, 28
predict.RprobitB_fit, 30
print.RprobitB_fit, 32
print.summary.RprobitB_fit, 33
residuals.RprobitB_fit, 33
summary.RprobitB_fit, 34
update.RprobitB_fit, 36
utility_updates, 37
vcov.RprobitB_fit, 39
WAIC, 40

as_draws.RprobitB_fit, 3

bayes_factor, 3
bridgesampling::bf(), *3*

choicedata::choice_parameters(), *15*
choicedata::generate_choice_covariates(), *15*
class_updates, 5
coef.RprobitB_fit, 8
coefficient_updates, 9
confint.RprobitB_fit, 11

d_to_gamma *(*utility_updates*)*, 37

fit, 12
fit(), *36*
formula, *18*
formula.RprobitB_fit, 21

interpret, 21

latent_class_diagnostics, 24
log_likelihood_ordered *(*utility_updates*)*, 37
logLik.RprobitB_fit, 25
loo, 26
loo::loo(), *26*
loo::waic(), *40*

model.frame.RprobitB_fit, 27

nobs(), *26*
nobs.RprobitB_fit, 28

oeli::pmvnorm(), *4*, *25*, *26*, *31*, *40*

plot.RprobitB_fit, 28
predict.RprobitB_fit, 30
print.RprobitB_fit, 32
print.RprobitB_interpretation *(*interpret*)*, 21
print.summary.RprobitB_fit, 33

residuals.RprobitB_fit, 33

sample_allocation *(*class_updates*)*, 5
stats::confint(), *11*
stats::vcov(), *39*
summary.RprobitB_fit, 34

update.RprobitB_fit, 36
update_b *(*coefficient_updates*)*, 9
update_b_c *(*coefficient_updates*)*, 9
update_classes_dp *(*class_updates*)*, 5
update_classes_wb *(*class_updates*)*, 5

update_coefficient
(coefficient_updates), 9
update_d (utility_updates), 37
update_m (class_updates), 5
update_Omega (coefficient_updates), 9
update_Omega_c (coefficient_updates), 9
update_s (class_updates), 5
update_Sigma (utility_updates), 37
update_U (utility_updates), 37
update_U_ranked (utility_updates), 37
update_z (class_updates), 5
utility_updates, 37

vcov.RprobitB_fit, 39

WAIC, 40