Package 'ordbetareg'

July 22, 2025

Type Package

Title Ordered Beta Regression Models with 'brms'

Version 0.8

Description Implements ordered beta regression models, which are for modeling continuous variables with upper and lower bounds, such as survey sliders, dose-response relationships and indexes. For more information, see Kubinec (2023) <doi:10.31235/osf.io/2sx6y>. The package is a front-end to the R package 'brms', which facilitates a range of regression specifications, including hierarchical, dynamic and multivariate modeling.

BugReports `https://github.com/saudiwin/ordbetareg_pack/issues`

License MIT + file LICENSE

Encoding UTF-8

LazyData true

LazyDataCompression xz

RoxygenNote 7.3.2

Depends R (>= 3.5), brms (>= 2.18.0), stats

Imports dplyr, ggplot2 (>= 3.4.0), tidyr, scales, stringr, abind, checkmate, insight, rstantools

Suggests rmarkdown, quarto, knitr, gt, modelsummary (>= 1.4.1), marginaleffects (>= 0.10.0), haven, Hmisc, collapse, ggthemes, glmmTMB, mice, bayestestR, gganimate, transformr, DeclareDesign

VignetteBuilder quarto

NeedsCompilation no

Author Robert Kubinec [aut, cre] (ORCID: <`https://orcid.org/0000-0001-6655-4119`>)

Maintainer Robert Kubinec <`bobkubinec@gmail.com`>

Repository CRAN

Date/Publication 2025-02-03 19:30:06 UTC

Contents

dordbeta 2
fit_multivariate 3
normalize 3
ordbetareg 4
ord_fit_mean 8
ord_fit_phi 9
pew 9
plot_heiss 10
posterior_epred_ordbeta.brmsfit 12
pp_check_ordbeta 14
rordbeta 15
sim_data 16
sim_ordbeta 17

Index **20**

`dordbeta` *Probability Density Function for the Ordered Beta Distribution*

Description

This function will return the density of given variates of the ordered beta distribution conditional on values for the mean (`mu`), dispersion (`phi`) and cutpoints governing the ratio of degenerate (discrete) to continuous responses.

Usage

```
dordbeta(x = 0.9, mu = 0.5, phi = 1, cutpoints = c(-1, 1), log = FALSE)
```

Arguments

`x`	Variates of the ordered beta distribution (should be in the [0,1] interval).
`mu`	Value of the mean of the distribution. Should be in the \(0,1\) interval (cannot be strictly equal to 0 or 1). If length is greater than 1, should be of length x.
`phi`	Value of the dispersion parameter. Should be strictly greater than 0. If length is greater than 1, should be of length x.
`cutpoints`	A vector of two numeric values for the cutpoints. Second value should
`log`	where to return the log density be strictly greater than the first value.

Value

Returns a vector of length `x` of the density of the ordered beta distribution conditional on `mu` and `phi`.

Examples

```
# examine density (likelihood) of different possible values
# given fixed values for ordered beta parameters

x <- seq(0, 1, by=0.01)

x_dens <- dordbeta(x, mu = 0.3, phi=2, cutpoints=c(-2, 2))

# Most likely value for x is approx 1
# Note discontinuity in density function between continuous/discrete values
# density function is a combined PMF/PDF, so not a real PDF
# can though be used for MLE

plot(x_dens, x)

# discrete values should be compared to each other:
# prob of discrete 0 > prob of discrete 1

x_dens[x==0] > x_dens[x==1]
```

`fit_multivariate` *Fitted Ordered Beta Regression Model (Multivariate regression)*

Description

A fitted ordered beta regression model with two responses, one an ordered beta regression and the other a Gaussian/Normal outcome. Useful for examining mediation analysis.

Usage

```
fit_multivariate
```

Format

an `ordbetareg` object

`normalize` *Normalize Outcome/Response to \[0,1\] Interval*

Description

This function takes a continuous (double) column of data and converts it to have 0 as the lower bound and 1 as the upper bound.

Usage

```
normalize(outcome, true_bounds = NULL)
```

Arguments

`outcome`	Any non-character vector. Factors will be converted to numeric via coercion.
`true_bounds`	Specify this parameter with the lower and upper bound if the observed min/max of the outcome should not be used. Useful when an upper or lower bound exists but the observed data is less than/more than that bound. The normalization function will respect these bounds.

Details

Beta regression can only be done with a response that is continuous with a lower bound of 0 and an upper bound of 1. However, it is straightforward to transform any lower and upper-bounded continuous variable to the \[0,1\] interval. This function does the transformation and saves the original bounds as attributes so that the bounds can be reverse-transformed.

Value

A numeric vector with an upper bound of 1 and a lower bound of 0. The original bounds are saved in the attributes "lower_bound" and "upper_bound".

Examples

```
# set up arbitrary upper and lower-bounded vector
outcome <- runif(1000, min=-33, max=445)

# normalize to \[0,1\]

trans_outcome <- normalize(outcome=outcome)
summary(trans_outcome)

# only works with numeric vectors and factors

  try(normalize(outcome=c('a','b')))
```

`ordbetareg` *Fit Ordered Beta Regression Model*

Description

This function allows you to estimate an ordered beta regression model via a formula syntax.

Usage

```
ordbetareg(
  formula = NULL,
  data = NULL,
  true_bounds = NULL,
  phi_reg = "none",
```

```
  use_brm_multiple = FALSE,
  coef_prior_mean = 0,
  coef_prior_SD = 5,
  intercept_prior_mean = NULL,
  intercept_prior_SD = NULL,
  phi_prior = 0.1,
  dirichlet_prior = c(1, 1, 1),
  phi_coef_prior_mean = 0,
  phi_coef_prior_SD = 5,
  phi_intercept_prior_mean = NULL,
  phi_intercept_prior_SD = NULL,
  extra_prior = NULL,
  manual_prior = NULL,
  init = "0",
  return_stancode = FALSE,
  ...
)
```

Arguments

`formula`	Either an R formula in the form response/DV ~ var1 + var2 etc. *or* formula object as created/called by the `brms` brms::bf function.
`data`	An R data frame or tibble containing the variables in the formula
`true_bounds`	If the true bounds of the outcome/response don't exist in the data, pass a length 2 numeric vector of the minimum and maximum bounds to properly normalize the outcome/response
`phi_reg`	Whether you are including a linear model predicting the dispersion parameter, phi, and/or for the response. If you are including models for both, pass option 'both'. If you only have an intercept for the outcome (i.e. a 1 in place of covariates), pass 'only'. If you only have intercepts for phi (such as a varying intercepts/random effects) model, pass the value "intercepts". To set priors on these intercepts, use the `extra-prior` option with the brms::set_prior function (class="sd"). If no model of any kind for phi, the default, pass 'none'.
`use_brm_multiple`	(T/F) Whether the model should use brms::brm_multiple for multiple imputation over multiple dataframes passed as a list to the `data` argument
`coef_prior_mean`	The mean of the Normal distribution prior on the regression coefficients (for predicting the mean of the response). Default is 0.
`coef_prior_SD`	The SD of the Normal distribution prior on the regression coefficients (for predicting the mean of the response). Default is 5, which makes the prior weakly informative on the logit scale.
`intercept_prior_mean`	The mean of the Normal distribution prior for the intercept. By default is NULL, which means the intercept receives the same prior as `coef_prior_mean`. To zero out the intercept, set this parameter to 0 and `coef_prior_SD` to a very

	small number (0.01 or smaller). NOTE: the default intercept in `brms` is centered (mean-subtracted) by default. To use a traditional intercept, either add `0 + Intercept` to the formula or specify `center=FALSE` in the `bf` formula function for `brms`. See `brms::brmsformula()` for more info.
`intercept_prior_SD`	The SD of the Normal distribution prior for the intercept. By default is NULL, which means the intercept receives the same prior SD as `coef_prior_SD`.
`phi_prior`	The mean parameter of the exponential prior on phi, which determines the dispersion of the beta distribution. The default is .1, which equals a mean of 10 and is thus weakly informative on the interval (0.4, 30). If the response has very low variance (i.e. tightly) clusters around a specific value, then decreasing this prior (and increasing the expected value) may be helpful. Checking the value of phi in the output of the model command will reveal if a value of 0.1 (mean of 10) is too small.
`dirichlet_prior`	A vector of three integers corresponding to the prior parameters for the dirchlet distribution (alpha parameter) governing the location of the cutpoints between the components of the response (continuous vs. degenerate). The default is 1 which puts equal probability on degenerate versus continuous responses. Likely only needs to be changed in a repeated sampling situation to stabilize the cutpoint locations across samples.
`phi_coef_prior_mean`	The mean of the Normal distribution prior on the regression coefficients for predicting phi, the dispersion parameter. Only useful if a linear model is being fit to phi. Default is 0.
`phi_coef_prior_SD`	The SD of the Normal distribution prior on the regression coefficients for predicting phi, the dispersion parameter. Only useful if a linear model is being fit to phi. Default is 5, which makes the prior weakly informative on the exponential scale.
`phi_intercept_prior_mean`	The mean of the Normal distribution prior for the phi (dispersion) regression intercept. By default is NULL, which means the intercept receives the same prior as `phi_coef_prior_mean`. To zero out the intercept, set this parameter to 0 and `phi_coef_prior_SD` to a very small number (0.01 or smaller).
`phi_intercept_prior_SD`	The SD of the Normal distribution prior for the phi (dispersion) regression intercept. By default is NULL, which means the intercept receives the same prior SD as `phi_coef_prior_SD`.
`extra_prior`	An additional prior, such as a prior for a specific regression coefficient, added to the outcome regression by passing one of the `brms` functions brms::set_prior or brms::prior_string with appropriate values.
`manual_prior`	If you want to set your own custom priors with `brms`, use this option to pass any valid `brms` priors such as those created with brms::set_prior or brms::prior_string. Note that this option replaces any other priors set. Useful especially when doing something unorthodox like modeling cutpoints.

`init`	This parameter is used to determine starting values for the Stan sampler to begin Markov Chain Monte Carlo sampling. It is set by default at 0 because the non-linear nature of beta regression means that it is possible to begin with extreme values depending on the scale of the covariates. Setting this to 0 helps the sampler find starting values. It does, on the other hand, limit the ability to detect convergence issues with Rhat statistics. If that is a concern, such as with an experimental feature of `brms`, set this to `"random"` to get more robust starting values (just be sure to scale the covariates so they are not too large in absolute size).
`return_stancode`	If `TRUE`, will pass back the *only* the Stan code for the model as a character vector rather than fitting the model.
`...`	All other arguments passed on to the `brm` function

Details

This function is a wrapper around the brms::brm function, which is a powerful Bayesian regression modeling engine using Stan. To fully explore the options available, including dynamic and hierarchical modeling, please see the documentation for the `brm` function above. As the ordered beta regression model is currently not available in `brms` natively, this modeling function allows a `brms` model to be fit with the ordered beta regression distribution.

For more information about the model, see the paper here: https://osf.io/preprints/socarxiv/2sx6y/.

This function allows you to set priors on the dispersion parameter, the cutpoints, and the regression coefficients (see below for options). However, to add specific priors on individual covariates, you would need to use the brms::set_prior function by specifying an individual covariate (see function documentation) and passing the result of the function call to the `extra_prior` argument.

This function will also automatically normalize the outcome so that it lies in the \[0,1\] interval, as required by beta regression. For furthur information, see the documentation for the normalize function.

Priors can be set on a variety of coefficients in the model, see the description of parameters `coef_prior_mean` and `intercept_prior_mean`, in addition to setting a custom prior with the `extra_prior` option. When setting priors on intercepts, it is important to note that by default, all intercepts in brms are centered (the means are subtracted from the data). As a result, a prior set on the default intercept will have a different interpretation than a traditional intercept (i.e. the value of the outcome when the covariates are all zero). To change this setting, use the `brms::bf()` function as a wrapper around the formula with the option `center=FALSE` to set priors on a traditional non-centered intercept.

Note that while `brms` also supports adding `0 + Intercept` to the formula to address this issue, `ordbetareg` does not support this syntax. Instead, use `center=FALSE` as an option to `brms::bf()`.

Value

A `brms` object fitted with the ordered beta regression distribution.

Examples

```
# load survey data that comes with the package
```

```
library(dplyr)
data("pew")

# prepare data

model_data <- select(pew,therm,
             education="F_EDUCCAT2_FINAL",
             region="F_CREGION_FINAL",
             income="F_INCOME_FINAL")

# It takes a while to fit the models. Run the code
# below if you want to load a saved fitted model from the
# package, otherwise use the model-fitting code

data("ord_fit_mean")


  # fit the actual model

  if(.Platform$OS.type!="windows") {

    ord_fit_mean <- ordbetareg(formula=therm ~ education + income +
      (1|region),
      data=model_data,
      cores=2,chains=2)

  }




# access values of the coefficients

summary(ord_fit_mean)
```

`ord_fit_mean` *Fitted Ordered Beta Regression Model*

Description

A fitted ordered beta regression model to the mean of the thermometer column from the pew data.

Usage

```
ord_fit_mean
```

Format

an `ordbetareg` object

`ord_fit_phi` *Fitted Ordered Beta Regression Model (Phi Regression)*

Description

A fitted ordered beta regression model to the dispersion parameter of the thermometer column from the pew data.

Usage

```
ord_fit_phi
```

Format

an `ordbetareg` object

`pew` *Pew American Trends Panel Wave 28*

Description

A dataset with the non-missing responses for the 28th wave of the Pew American Trends Panel survey.

Usage

```
pew
```

Format

A data frame with 140 variables and 2,538 observations.

Source

https://www.pewresearch.org/social-trends/dataset/american-trends-panel-wave-28/]

plot_heiss *Heiss Plot for Predicted Proportions of Bounded Scale Components*

Description

The Heiss plot, developed by the statistician Andrew Heiss, is a plot of the predicted proportions of components on a bounded scale that are grouped by the unique levels of a grouping variable or factor (such as a random effect) in the model. The plot excels at showing how the scale components–that is, the bottom, middle continuous, and top ends of the scale–vary with a discrete variable while also capturing posterior uncertainty. This plot was the winner of the 2023 ordbetareg Visualization Prize.

Usage

```
plot_heiss(
  object,
  grouping_fac = NULL,
  recode_group_labels = NULL,
  ndraws = NULL,
  show_category_perc_labels = TRUE,
  category_label_font_size = 3,
  category_label_accuracy = 1,
  strip_text_font = element_text(face = "plain", size = 9),
  plot_title = "Predicted Proportions of Bounded Scale Components",
  plot_subtitle = paste0("By Unique Values of ", grouping_fac),
  plot_caption = NULL,
  plot_caption_width = 70,
  calc_func = mean,
  lb = 0.05,
  upb = 0.95,
  plot_font_size = 11,
  plot_font = "",
  y_axis_label = "Predicted Proportions",
  legend_name = "Scale Components",
  component_colors = c("#ef8737", "#bb292c", "#62205f"),
  component_labels = c("0", "(0-1)", "1"),
  ...
)
```

Arguments

`object`	A fitted `ordbetareg()` model object.
`grouping_fac`	A character string indicating the name of the discrete column in the data used for grouping predictions. Must be a valid column name that was passed to `ordbetareg()`.

`recode_group_labels`	Optional. A character vector of new labels for the grouping factor levels. Must match the number and order of unique levels/values in `grouping_fac`.
`ndraws`	Optional. The number of posterior draws to use for predictions. If `NULL`, all available draws are used.
`show_category_perc_labels`	Logical. Whether to display category percentage labels on the plot. Defaults to `TRUE`.
`category_label_font_size`	The `ggplot2` font size for the labels on the scale components (if `show_category_perc_labels` is `TRUE`). Defaults to 3.
`category_label_accuracy`	The accuracy, or amount of rounding, for component label ranges on the plot (if `show_category_perc_labels` is `TRUE`). Default is 1. See `scales::label_percent()` for more info on meaning of `accuracy` parameter.
`strip_text_font`	A `ggplot2::element_text` object defining the font style for facet strip text. Defaults to `element_text(face = "plain", size = 9)`.
`plot_title`	Title of the plot. Defaults to "Predicted Proportions of Bounded Scale Components".
`plot_subtitle`	Subtitle of the plot. Defaults to a message indicating the grouping variable.
`plot_caption`	Caption text for the plot. If NULL, the default, will use a detailed but static description of the plot contents.
`plot_caption_width`	Width (in characters) at which the caption is wrapped. Defaults to 60.
`calc_func`	A function used to calculate the central tendency of predictions. Defaults to `mean`.
`lb`	Lower bound for uncertainty intervals. Defaults to 0.05 (5th percentile).
`upb`	Upper bound for uncertainty intervals. Defaults to 0.95 (95th percentile).
`plot_font_size`	Base font size for the plot. Defaults to 11.
`plot_font`	Base font family for the plot. Defaults to an empty string (uses system default).
`y_axis_label`	Label for the y-axis. Defaults to "Predicted Proportions".
`legend_name`	Legend title. Defaults to "Scale Components".
`component_colors`	A character vector of colors for the plot components (bottom, continuous, top). Defaults to `c("#ef8737", "#bb292c", "#62205f")`.
`component_labels`	A character vector of labels for the scale/outcome components (bottom, continuous, top). Defaults to `c("0", "(0-1)", "1")`.
`...`	Additional arguments passed to [posterior_epred_ordbeta())]. [posterior_epred_ordbeta())]: R:posterior_epred_ordbeta())

Details

For more details of the plot, see:

Heiss, Andrew and Ye, Meng. "Enforcing Boundaries: China's Overseas NGO Law and Operational Constraints for Global Civil Society." Working Paper, 2023. `https://stats.andrewheiss.com/compassionate-clam/notebook/manuscript.html`.

Value

A `ggplot2` object representing the predicted proportions of the components.

Examples

```
# Load a fitted model object and create a plot for
# distinct values of the factor education
#
# data('ord_fit_mean')
#
# plot_heiss(ord_fit_mean,ndraws=100)
#
# See introductory package vignette for more information on function options
```

`posterior_epred_ordbeta.brmsfit`

Calculate Probability of Response Components

Description

This function is an alternative to the `brms` default `posterior_epred` to allow for predictions of the probability of the bottom, top, or middle (i.e. continuous) parts of the response. Useful when wanting to understand what the effect of a covariate is on bottom or top values of the scale.

Usage

```
## S3 method for class 'brmsfit'
posterior_epred_ordbeta(
  object,
  component = "all",
  newdata = NULL,
  re_formula = NULL,
  re.form = NULL,
  resp = NULL,
  dpar = NULL,
  nlpar = NULL,
  ndraws = NULL,
  draw_ids = NULL,
  sort = FALSE,
  ...
)
```

Arguments

`object`	An ordbetareg/brms object
`component`	The type of response component, i.e., the probability of the bottom end of the scale, the top end, or the middle (i.e.) continuous values.
`newdata`	see brms::posterior_epred
`re_formula`	see brms::posterior_epred
`re.form`	see brms::posterior_epred
`resp`	see brms::posterior_epred
`dpar`	see brms::posterior_epred
`nlpar`	see brms::posterior_epred
`ndraws`	see brms::posterior_epred
`draw_ids`	see brms::posterior_epred
`sort`	see brms::posterior_epred
`...`	see brms::posterior_epred

Details

To predict the top, bottom, or "middle" (i.e. continuous) components of the response, set the `component` argument to "top", "bottom" or "continuous". By default, `component` is set to "all", which will replicate behavior of the default `posterior_epred` function.

All other arguments besides `component` are the same as the standard generic `posterior_predict`. For more information on the relevant arguments for `posterior_epred`, see brms::posterior_epred.

Value

An S x N matrix where S is the number of posterior draws and N is the number of observations.

Examples

```
data('ord_fit_mean')

# use function to calculate probability of top end of scale

pr_1s <- posterior_epred_ordbeta(ord_fit_mean,component="top")

# use function to calculate probability of bottom end of scale

pr_0s <- posterior_epred_ordbeta(ord_fit_mean,component="top")

# use function to calculate probability of continuous /
# beta-distributed part of scale

pr_beta <- posterior_epred_ordbeta(ord_fit_mean,component="top")
```

`pp_check_ordbeta` *Accurate Posterior Predictive Plots for Ordbetareg Models*

Description

The standard brms::pp_check plot available via `brms` is not accurate for ordbetareg models because an ordered beta regression has both continuous and discrete components. This function implements a bar plot and a density plot for the continuous and discrete elements separately, and will return accurate posterior predictive plots relative to the data.

Usage

```
pp_check_ordbeta(
  model = NULL,
  dv = NULL,
  type = "both",
  ndraws = 10,
  cores = NULL,
  group = NULL,
  new_theme = NULL,
  outcome_label = NULL,
  animate = FALSE,
  reverse_bounds = TRUE,
  facet_scales = "fixed"
)
```

Arguments

`model`	A fitted ordbetareg model.
`dv`	If you fit a model with multiple DVs/responses, pass the name of the DV as a character value. Note: this must be the same as the name of the column in the data used to fit the model.
`type`	Default is "both" for creating both a discrete (bar) and continuous (density) plot. Can also be "discrete" for only the bar plot for discrete values (0/1) or "continuous" for continuous values (density plot).
`ndraws`	Number of posterior draws to use to calculate estimates and show in plot. Defaults to 10.
`cores`	Number of cores to use to produce posterior predictive distribution. Defaults to NULL or 1 core.
`group`	A factor variable of the same number of rows as the data that is used to broduce grouped (faceted) plots of the posterior distribution.
`new_theme`	Any additional themes to be added to ggplot2 (default is NULL).
`outcome_label`	A character value that will replace the name of the outcome in the plot (default is the name of the response variable in the data frame).

`animate`	Whether to animate each posterior draw for continuous distributions (defaults to FALSE). Requires installation of the `gganimate` and `transformr` R packages.
`reverse_bounds`	Whether to plot data using the original bounds in the data (i.e. not 0 and 1).
`facet_scales`	The option passed on to the `facet_wrap` function in `ggplot2` for the type of scale for facetting if passing a variable for `group`. Defaults to `"fixed"` scales but can be set to `"free_y"` to allow probability density/bar count scales to vary or `"free"` to allow both x and y axes to vary (i.e., also outcome axis ticks).

Value

If "both", prints both plots and returns a list of both plots as `ggplot2` objects. Otherwise, prints and returnst the specific plot as a `ggplot2` object.

Examples

```
# need a fitted ordbetareg model

data("ord_fit_mean")

out_plots <- pp_check_ordbeta(ord_fit_mean)

# view discrete bar plot

out_plots$discrete

# view continuous density plot

out_plots$continuous

# change title using ggplot2 ggtitle function

out_plots$discrete + ggplot2::ggtitle("New title")
```

`rordbeta` *Generate Ordered Beta Variates*

Description

This function will generate ordered beta random variates given values for the mean (`mu`), dispersion (`phi`) and cutpoints governing the ratio of degenerate (discrete) to continuous responses.

Usage

```
rordbeta(n = 100, mu = 0.5, phi = 1, cutpoints = c(-1, 1))
```

Arguments

`n`	Number of variates to generate.
`mu`	Value of the mean of the distribution. Should be in the \(0,1\) interval (cannot be strictly equal to 0 or 1). If length is greater than 1, should be of length `n`.
`phi`	Value of the dispersion parameter. Should be strictly greater than 0. If length is greater than 1, should be of length `n`.
`cutpoints`	A vector of two numeric values for the cutpoints. Second value should be strictly greater than the first value.

Value

A vector of length `n` of variates from the ordered beta distribution.

Examples

```
# generate 100 random variates with an average of 0.7
# all will be in the closed interval \[0,1\]

ordbeta_var <- rordbeta(n=100, mu=0.7, phi=2)

# Will be approx mean = 0.7 with high positive skew

summary(ordbeta_var)
```

`sim_data` *Simulated Ordered Beta Regression Values*

Description

The simulated draws used in the vignette for calculating statistical power.

Usage

```
sim_data
```

Format

A dataframe

`sim_ordbeta` *Power Calculation via Simulation of the Ordered Beta Regression Model*

Description

This function allows you to calculate power curves (or anything else) via simulating the ordered beta regression model.

Usage

```
sim_ordbeta(
  N = 1000,
  k = 5,
  iter = 1000,
  cores = 1,
  phi = 1,
  cutpoints = c(-1, 1),
  beta_coef = NULL,
  beta_type = "continuous",
  treat_assign = 0.5,
  return_data = FALSE,
  seed = as.numeric(Sys.time()),
  ...
)
```

Arguments

`N`	The sample size for the simulation. Include a vector of integers to examine power/results for multiple sample sizes.
`k`	The number of covariates/predictors.
`iter`	The number of simulations to run. For power calculation, should be at least 500 (yes, this will take some time).
`cores`	The number of cores to use to parallelize the simulation.
`phi`	Value of the dispersion parameter in the beta distribution.
`cutpoints`	Value of the two cutpoints for the ordered model. By default are the values -1 and +1 (these are interpreted in the logit scale and so should not be too large). The farther apart, the fewer degenerate (0 or 1) responses there will be in the distribution.
`beta_coef`	If not null, a vector of length `k` of the true predictor coefficients/treatment values to use for the simulation. Otherwise, coefficients are drawn from a random uniform distribution from -1 to 1 for each predictor.
`beta_type`	Can be either `continuous` or `binary`. Use the latter for conventional treatments with two values.

`treat_assign`	If `beta_type` is set to `binary`, you can use this parameter to set the proportion of `N` assigned to treatment. By default, the parameter is set to 0.5 for equal/balanced treatment control groups.
`return_data`	Whether to return the simulated dqta as a list in the `data` column of the returned data frame.
`seed`	The seed to use to make the results reproducible. Set automatically to a date-time stamp.
`...`	Any other arguments are passed on to the brms::brm function to control modeling options.

Details

This function implements the simulation found in Kubinec (2022). This simulation allows you to vary the sample size, number & type of predictors, values of the predictors (or treatment values), and the power to target. The function returns a data frame with one row per simulation draw and covariate `k`.

Value

a tibble data frame with columns of simulated and estimated values and rows for each simulation iteration X coefficient combination. I.e., if there are five predictors, and 1,000 iterations, the resulting data frame will have 1,000 rows. If there are multiple values for `N`, then each value of `N` will have its own set of iterations, making the final size of the data a multiple of the number of sample sizes to iterate over. The data frame will have the following columns: 1.

Examples

```
# This function takes a while to run as it has
# to fit an ordered beta regression to each
# draw. The package comes with a saved
# simulation dataset you can inspect to see what the
# result looks like

data("sim_data")

library(dplyr)

# will take a while to run this


  if(.Platform$OS.type!="windows") {

    sim_data <- sim_ordbeta(N=c(250,750),
    k=1,
    beta_coef = .5,
    iter=5,cores=2,
    beta_type="binary",
    treat_assign=0.3)

    }
```

```
# to get the power values by N, simply summarize/group
# by N with functions from the R package dplyr

sim_data %>%
  group_by(N) %>%
  summarize(mean_power=mean(power))
```

Index

∗ **datasets**
    fit_multivariate, 3
    ord_fit_mean, 8
    ord_fit_phi, 9
    pew, 9
    sim_data, 16

brms::bf, *5*
brms::bf(), *7*
brms::brm, *7*, *18*
brms::brm_multiple, *5*
brms::brmsformula(), *6*
brms::posterior_epred, *13*
brms::pp_check, *14*
brms::prior_string, *6*
brms::set_prior, *5–7*

dordbeta, 2

fit_multivariate, 3

normalize, 3, *7*

ord_fit_mean, 8
ord_fit_phi, 9
ordbetareg, 4, *14*
ordbetareg(), *10*

pew, 9
plot_heiss, 10
posterior_epred_ordbeta
        *(*posterior_epred_ordbeta.brmsfit*)*,
        12
posterior_epred_ordbeta.brmsfit, 12
pp_check_ordbeta, 14

rordbeta, 15

scales::label_percent(), *11*
sim_data, 16
sim_ordbeta, 17