Package 'cbcTools'

June 2, 2023

Title Design and Evaluate Choice-Based Conjoint Survey Experiments

Version 0.3.3

Maintainer John Helveston <john.helveston@gmail.com>

Description Design and evaluate choice-based conjoint survey experiments in R. Generate survey designs, including randomized designs and Bayesian D-efficient designs as well as designs with ``no choice" options and labeled designs. Conveniently inspect the design balance and overlap, and simulate choice data for a survey design either randomly or according to a multinomial or mixed logit utility model defined by user-provided prior parameters. Conduct power analyses on a survey design by estimating the same model multiple times using different subsets of the data to simulate different sample sizes. Choice simulation and model estimation are handled using the 'logitr' package, and Bayesian D-efficient designs are obtained using the 'idefix' package. For more details see Helveston (2023) <doi:10.18637/jss.v105.i10> and Traets et al (2020) <doi:10.18637/jss.v096.i03>.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.2.3

Depends R (>= 3.5.0)

Suggests testthat, tibble

Imports dplyr, fastDummies, ggplot2, idefix, logitr (>= 1.0.1), MASS, parallel, randtoolbox, rlang, stats, utils

URL https://github.com/jhelvy/cbcTools

BugReports https://github.com/jhelvy/cbcTools/issues

NeedsCompilation no

Author John Helveston [cre, aut, cph] (<https://orcid.org/0000-0002-2657-9191>)

Repository CRAN

Date/Publication 2023-06-02 17:20:02 UTC

R topics documented:

`cbc_balance` *Counts of attribute balance*

Description

This function prints out a summary of the individual and pairwise counts of each level for each attribute across all choice questions in the design.

Usage

```
cbc_balance(design)
```

Arguments

`design`	A data frame of a survey design.

Value

Prints the individual and pairwise counts of the number of times each attribute levels in shown in the design.

Examples

```
library(cbcTools)

# A simple conjoint experiment about apples

# Generate all possible profiles
profiles <- cbc_profiles(
  price     = c(1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5),
  type      = c("Fuji", "Gala", "Honeycrisp"),
  freshness = c('Poor', 'Average', 'Excellent')
)
```

```
# Make a randomized survey design
design <- cbc_design(
  profiles = profiles,
  n_resp   = 300, # Number of respondents
  n_alts   = 3, # Number of alternatives per question
  n_q      = 6 # Number of questions per respondent
)

# Inspect the design balance
cbc_balance(design)

# Inspect the design overlap
cbc_overlap(design)
```

`cbc_choices` *Simulate choices for a survey design*

Description

Simulate choices for a survey design, either randomly or according to a utility model defined by user-provided prior parameters.

Usage

```
cbc_choices(design, obsID = "obsID", priors = NULL, n_draws = 100)
```

Arguments

`design`	A data frame of a survey design.
`obsID`	The name of the column in `design` that identifies each choice observation. Defaults to `"obsID"`.
`priors`	A list of one or more prior parameters that define a prior (assumed) utility model used to simulate choices for the `survey` data frame. If `NULL` (the default), choices will be randomly assigned.
`n_draws`	The number of Halton draws to use for simulated choices for mixed logit models. Defaults to `100`.

Value

Returns the `design` data frame with an additional `choice` column identifying the simulated choices.

Examples

```
library(cbcTools)

# A simple conjoint experiment about apples

# Generate all possible profiles
```

```
profiles <- cbc_profiles(
  price     = c(1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5),
  type      = c("Fuji", "Gala", "Honeycrisp"),
  freshness = c('Poor', 'Average', 'Excellent')
)

# Make a randomized survey design
design <- cbc_design(
  profiles = profiles,
  n_resp   = 300, # Number of respondents
  n_alts   = 3, # Number of alternatives per question
  n_q      = 6 # Number of questions per respondent
)

# Simulate random choices
data <- cbc_choices(
  design = design,
  obsID  = "obsID"
)

# Simulate choices according to a prior utility model
data <- cbc_choices(
  design = design,
  obsID = "obsID",
  priors = list(
    price     = 0.1,
    type      = c(0.1, 0.2),
    freshness = c(0.1, 0.2)
  )
)

# Simulate choices according to a prior model with interactions
data <- cbc_choices(
  design = design,
  obsID = "obsID",
  priors = list(
    price     = 0.1,
    type      = c(0.1, 0.2),
    freshness = c(0.1, 0.2),
    `price*type` = c(0.1, 0.5)
  )
)

# Simulate choices according to a prior utility model with random parameters
data <- cbc_choices(
  design = design,
  obsID = "obsID",
  priors = list(
    price = 0.1,
    type = randN(mean = c(0.1, 0.2), sd = c(1, 2)),
    freshness = c(0.1, 0.2)
  )
)
```

`cbc_design` *Make a random or Bayesian D-efficient choice-based conjoint survey design*

Description

This function creates a data frame containing a choice-based conjoint survey design where each row is an alternative. Designs can be either a randomized or Bayesian D-efficient, in which case an implementation of the CEA or Modfed Federov algorithm is used via the idefix package

Usage

```
cbc_design(
  profiles,
  n_resp,
  n_alts,
  n_q,
  n_blocks = 1,
  n_draws = 50,
  no_choice = FALSE,
  n_start = 5,
  label = NULL,
  priors = NULL,
  prior_no_choice = NULL,
  probs = FALSE,
  method = "CEA",
  max_iter = 50,
  parallel = TRUE
)
```

Arguments

`profiles`	A data frame in which each row is a possible profile. This can be generated using the `cbc_profiles()` function.
`n_resp`	Number of survey respondents.
`n_alts`	Number of alternatives per choice question.
`n_q`	Number of questions per respondent.
`n_blocks`	Number of blocks used in Bayesian D-efficient design. Max allowable is one block per respondent, defaults to 1, meaning every respondent sees the same set of choice questions.
`n_draws`	Number of draws used in simulating the prior distribution used in Bayesian D-efficient designs. Defaults to `50`.
`no_choice`	Include a "no choice" option in the choice sets? Defaults to `FALSE`. If `TRUE`, the total number of alternatives per question will be one more than the provided `n_alts` argument.

`n_start` A numeric value indicating the number of random start designs to use in obtaining a Bayesian D-efficient design. The default is `5`. Increasing `n_start` can result in a more efficient design at the expense of increased computational time.

`label` The name of the variable to use in a "labeled" design (also called an "alternative-specific design") such that each set of alternatives contains one of each of the levels in the `label` attribute. Currently only compatible with randomized designs. If used, the `n_alts` argument will be ignored as its value is defined by the unique number of levels in the `label` variable. Defaults to `NULL`.

`priors` A list of one or more assumed prior parameters used to generate a Bayesian D-efficient design. If `NULL` (the default), a randomized design will be generated.

`prior_no_choice` Prior utility value for the "no choice" alternative. Only required if `no_choice = TRUE`. Defaults to `NULL`.

`probs` If `TRUE`, for Bayesian D-efficient designs the resulting design includes average predicted probabilities for each alternative in each choice set given the sample from the prior preference distribution. Defaults to `FALSE`.

`method` Which method to use for obtaining a Bayesian D-efficient design, `"CEA"` or `"Modfed"`? Defaults to `"CEA"`. See `?idefix::CEA` and `?idefix::Modfed` for more details.

`max_iter` A numeric value indicating the maximum number allowed iterations when searching for a Bayesian D-efficient design. The default is 50.

`parallel` Logical value indicating whether computations should be done over multiple cores. The default is `TRUE`.

Value

A data frame containing a choice-based conjoint survey design where each row is an alternative.

Examples

```
library(cbcTools)

# A simple conjoint experiment about apples

# Generate all possible profiles
profiles <- cbc_profiles(
  price     = c(1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5),
  type      = c("Fuji", "Gala", "Honeycrisp"),
  freshness = c('Poor', 'Average', 'Excellent')
)

# Make a randomized survey design
design_rand <- cbc_design(
  profiles = profiles,
  n_resp   = 300, # Number of respondents
  n_alts   = 3,   # Number of alternatives per question
  n_q      = 6    # Number of questions per respondent
)
```

```
# Make a randomized survey design with a "no choice" option
design_rand_nochoice <- cbc_design(
  profiles  = profiles,
  n_resp    = 300, # Number of respondents
  n_alts    = 3,   # Number of alternatives per question
  n_q       = 6,   # Number of questions per respondent
  no_choice = TRUE
)

# Make a randomized labeled survey design with each "type" appearing in
# each choice question
design_rand_labeled <- cbc_design(
  profiles  = profiles,
  n_resp    = 300, # Number of respondents
  n_alts    = 3,   # Number of alternatives per question
  n_q       = 6,   # Number of questions per respondent
  label     = "type"
)

# Make a Bayesian D-efficient design with a prior model specified
# Note that by default parallel = TRUE.
design_deff <- cbc_design(
    profiles  = profiles,
    n_resp    = 300, # Number of respondents
    n_alts    = 3,  # Number of alternatives per question
    n_q       = 6,  # Number of questions per respondent
    n_start   = 1,
    priors = list(
        price     = -0.1,
        type      = c(0.1, 0.2),
        freshness = c(0.1, 0.2)
    ),
    parallel = FALSE
)
```

`cbc_overlap` *Counts of attribute overlap*

Description

This function prints out a summary of the amount of "overlap" across attributes within the choice questions. For example, for each attribute, the count under `"1"` is the number of choice questions in which the same level was shown across all alternatives for that attribute (because there was only one level shown). Likewise, the count under `"2"` is the number of choice questions in which only two unique levels of that attribute were shown, and so on.

Usage

```
cbc_overlap(design)
```

Arguments

design	A data frame of a survey design.

Value

Prints the counts of the number of choice questions that contain the unique number of levels for each attribute.

Examples

```
library(cbcTools)

# A simple conjoint experiment about apples

# Generate all possible profiles
profiles <- cbc_profiles(
  price     = c(1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5),
  freshness = c("Excellent", "Average", "Poor"),
  type      = c("Fuji", "Gala", "Honeycrisp")
)

# Make a randomized survey design
design <- cbc_design(
  profiles = profiles,
  n_resp   = 300, # Number of respondents
  n_alts   = 3, # Number of alternatives per question
  n_q      = 6 # Number of questions per respondent
)

# Inspect the design balance
cbc_balance(design)

# Inspect the design overlap
cbc_overlap(design)
```

---

cbc_power *Estimate the same model on different size subsets of data*

Description

This function estimates the same model multiple times using different size subsets of a set of choice data and then returns a data frame of the estimated model coefficients and standard errors for each sample size. This is useful for determining the required sample size for obtaining a desired level of statistical power on each coefficient. The number of models to estimate is set by the nbreaks argument, which breaks up the data into groups of increasing sample sizes. All models are estimated models using the logitr package. For more details see Helveston (2023) doi:10.18637/jss.v105.i10.

Usage

```
cbc_power(
  data,
  outcome,
  obsID,
  pars,
  randPars = NULL,
  nbreaks = 10,
  n_q = 1,
  return_models = FALSE,
  panelID = NULL,
  clusterID = NULL,
  robust = FALSE,
  predict = FALSE,
  n_cores = NULL,
  ...
)
```

Arguments

data	The data, formatted as a data.frame object.
outcome	The name of the column that identifies the outcome variable, which should be coded with a 1 for TRUE and 0 for FALSE.
obsID	The name of the column that identifies each observation.
pars	The names of the parameters to be estimated in the model. Must be the same as the column names in the data argument.
randPars	A named vector whose names are the random parameters and values the distribution: 'n' for normal or 'ln' for log-normal. Defaults to NULL.
nbreaks	The number of different sample size groups.
n_q	Number of questions per respondent. Defaults to 1 if not specified.
return_models	If TRUE, a list of all estimated models is returned. This can be useful if you want to extract other outputs from each model, such as the variance-covariance matrix, etc. Defaults to FALSE.
panelID	The name of the column that identifies the individual (for panel data where multiple observations are recorded for each individual). Defaults to NULL.
clusterID	The name of the column that identifies the cluster groups to be used in model estimation. Defaults to NULL.
robust	Determines whether or not a robust covariance matrix is estimated. Defaults to FALSE. Specification of a clusterID will override the user setting and set this to 'TRUE' (a warning will be displayed in this case). Replicates the functionality of Stata's cmcmmixlogit.
predict	If TRUE, predicted probabilities, fitted values, and residuals are also included in the returned model objects. Defaults to FALSE.

n_cores	The number of cores to use for parallel processing. Set to 1 to run serially Defaults to NULL, in which case the number of cores is set to parallel::detectCores() - 1. Max cores allowed is capped at parallel::detectCores().
...	Other arguments that are passed to logitr::logitr() for model estimation. See the logitr documentation for details about other available arguments.

Value

Returns a data frame of estimated model coefficients and standard errors for the same model estimated on subsets of the data with increasing sample sizes.

Examples

```
library(cbcTools)

# A simple conjoint experiment about apples

# Generate all possible profiles
profiles <- cbc_profiles(
  price     = c(1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5),
  type      = c("Fuji", "Gala", "Honeycrisp"),
  freshness = c('Poor', 'Average', 'Excellent')
)

# Make a randomized survey design
design <- cbc_design(
  profiles = profiles,
  n_resp   = 300, # Number of respondents
  n_alts   = 3, # Number of alternatives per question
  n_q      = 6 # Number of questions per respondent
)

# Simulate random choices
data <- cbc_choices(
  design = design,
  obsID  = "obsID"
)

# Conduct a power analysis
power <- cbc_power(
  data    = data,
  pars    = c("price", "type", "freshness"),
  outcome = "choice",
  obsID   = "obsID",
  nbreaks = 10,
  n_q     = 6,
  n_cores = 2
)
```

cbc_profiles *Make a data frame of all combinations of attribute levels*

Description

This function creates a data frame of of all possible combinations of attribute levels.

Usage

```
cbc_profiles(...)
```

Arguments

`...`	Any number of named vectors defining each attribute and their levels, e.g. `price = c(1, 2, 3)`. Separate each vector by a comma.

Value

A data frame of all possible combinations of attribute levels.

Examples

```
library(cbcTools)

# Generate all profiles for a simple conjoint experiment about apples
profiles <- cbc_profiles(
  price     = c(1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5),
  type      = c("Fuji", "Gala", "Honeycrisp"),
  freshness = c('Poor', 'Average', 'Excellent')
)
```

cbc_restrict *Obtain a restricted set of profiles*

Description

This function returns a restricted set of profiles as a data frame.

Usage

```
cbc_restrict(profiles, ...)
```

Arguments

profiles	A data frame in which each row is a possible profile. This can be generated using the cbc_profiles() function.
...	Any number of restricted pairs of attribute levels, defined as pairs of logical expressions separated by commas. For example, the restriction type == 'Fuji' & freshness == 'Poor' will eliminate profiles such that "Fuji" type apples will never be shown with "Poor" freshness.

Value

A restricted set of profiles as a data frame.

Examples

```
library(cbcTools)

# Generate all profiles for a simple conjoint experiment about apples
profiles <- cbc_profiles(
  price     = c(1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5),
  type      = c("Fuji", "Gala", "Honeycrisp"),
  freshness = c('Poor', 'Average', 'Excellent')
)

# Obtain a restricted subset of profiles based on pairs of logical
# expressions. The example below contains the following restrictions:

# - `"Gala"` apples will not be shown with the prices `1.5`, `2.5`, & `3.5`.
# - `"Honeycrisp"` apples will not be shown with prices less than `2`.
# - `"Honeycrisp"` apples will not be shown with the `"Poor"` freshness.
# - `"Fuji"` apples will not be shown with the `"Excellent"` freshness.

profiles_restricted <- cbc_restrict(
    profiles,
    type == "Gala" & price %in% c(1.5, 2.5, 3.5),
    type == "Honeycrisp" & price > 2,
    type == "Honeycrisp" & freshness == "Poor",
    type == "Fuji" & freshness == "Excellent"
)
```

---

miscmethods.cbc_errors

Methods for cbc_errors objects

Description

Miscellaneous methods for cbc_errors class objects.

Usage

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

Arguments

x	is an object of class cbc_errors.
...	further arguments.

Value

Returns a ggplot2 object plotting standard errors versus sample size.

Examples

```
library(cbcTools)

# A simple conjoint experiment about apples

# Generate all possible profiles
profiles <- cbc_profiles(
  price     = c(1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5),
  type      = c("Fuji", "Gala", "Honeycrisp"),
  freshness = c('Poor', 'Average', 'Excellent')
)

# Make a randomized survey design
design <- cbc_design(
  profiles = profiles,
  n_resp   = 300, # Number of respondents
  n_alts   = 3, # Number of alternatives per question
  n_q      = 6 # Number of questions per respondent
)

# Simulate random choices
data <- cbc_choices(
  design = design,
  obsID  = "obsID"
)

# Conduct a power analysis
power <- cbc_power(
  data    = data,
  pars    = c("price", "type", "freshness"),
  outcome = "choice",
  obsID   = "obsID",
  nbreaks = 10,
  n_q     = 6
)

# Visualize the results
plot(power)
```

miscmethods.cbc_models *Methods for cbc_models objects*

Description

Miscellaneous methods for cbc_models class objects.

Usage

```
## S3 method for class 'cbc_models'
print(
  x,
  digits = max(3, getOption("digits") - 2),
  width = getOption("width"),
  ...
)
```

Arguments

x	is an object of class cbc_models.
digits	the number of digits for printing, defaults to 3.
width	the width of the printing.
...	further arguments.

Value

No return value, prints a summary of estimated models.

randLN *Define prior (assumed) model parameter as log-normally-distributed.*

Description

Define prior (assumed) model parameter as log-normally-distributed. Used in the cbc_choices() function.

Usage

```
randLN(mean = 0, sd = 1)
```

Arguments

mean	Mean of the distribution on the log scale, defaults to 0.
sd	Standard deviation of the distribution on the log scale, defaults to 1.

Value

A list defining log-normally-distributed parameters of the prior (assumed) utility model used to simulate choices in the `cbc_choices()` function.

Examples

```
# Insert example
```

`randN` *Define a prior (assumed) model parameter as normally-distributed.*

Description

Define a prior (assumed) model parameter as normally-distributed. Used in the `cbc_choices()` function.

Usage

```
randN(mean = 0, sd = 1)
```

Arguments

`mean`	Vector of means, defaults to `0`.
`sd`	Vector of standard deviations, defaults to `1`.

Value

A list defining normally-distributed parameters of the prior (assumed) utility model used to simulate choices in the `cbc_choices()` function.

Examples

```
# Insert example
```

Index