Package 'glmmboot'

December 26, 2018

Type Package

Title Bootstrap Resampling for Mixed Effects and Plain Models

Version 0.3.0

Description Performs bootstrap resampling for most models that update() works for. There are two main functions: BootGlmm() performs block resampling if random effects are present, and case resampling if not; BootCI() converts output from bootstrap model runs into confidence intervals and p-values. By default, BootGlmm() calls BootCI().
Package motivated by Humphrey and Swingley (2018) <arXiv:1805.08670>.

License AGPL-3 | file LICENSE

URL https://github.com/ColmanHumphrey/glmmboot

Encoding UTF-8

LazyData true

Depends R (>= 3.1)

Imports methods, stats

Suggests glmmTMB (>= 0.2.1), testthat (>= 0.11.0), pbapply (>= 1.3.0), parallel (>= 3.0.0), knitr, rmarkdown

RoxygenNote 6.1.1

VignetteBuilder knitr

NeedsCompilation no

Author Colman Humphrey [aut, cre]

Maintainer Colman Humphrey <humphrc@tcd.ie>

Repository CRAN

Date/Publication 2018-12-26 15:30:47

R topics documented:

`BootCI` *Enter first level estimates and second level estimates, get bootstrap interval, from the pivotal bootstrap t (Efron and Tibshirani 1994, also endorsed by Hesterberg 2015).*

Description

Enter first level estimates and second level estimates, get bootstrap interval, from the pivotal bootstrap t (Efron and Tibshirani 1994, also endorsed by Hesterberg 2015).

Usage

```
BootCI(base_coef_se = NULL, resampled_coef_se = NULL, orig_df = NULL,
  alp_level = 0.05, probs = NULL)
```

Arguments

`base_coef_se`	Estimates and SEs from full sample. In matrix form: i.e. a (p+1) x 2 matrix, first column is estimates, second is standard errors. This is the output from using: coef(summary(model_output))[,1:2, drop = FALSE] or coef(summary(model_output))$cond[,1:2, drop = FALSE] if model_output is the output from a random effects model (some may not have cond as the correct pull).
`resampled_coef_se`	List of estimates and SEs from the bootstrapped resamples, each list entry has the same format as the base_coef_se above.
`orig_df`	Degrees of freedom to use to calculate the t-values used for the base interval.
`alp_level`	level of CI - if you fill in `probs`, will use those instead
`probs`	Default NULL, and will use alp_level to set endpoints. Else will calculate these CI endpoints.

Value

A matrix containing: Estimates; Bootstrap interval endpoints; Bootstrap p-value; Base p-value; Base interval endpoints; Relative width of bootstrap interval to base

Examples

```
x <- rnorm(20)
y <- rnorm(20) + x
xy_data = data.frame(x = x, y = y)
first_model <- lm(y ~ x, data = xy_data)
out_list <- BootGlmm(first_model, 20, base_data = xy_data, return_coefs_instead = TRUE)
BootCI(out_list$base_coef_se,
       out_list$resampled_coef_se)
```

```
data(test_data)
library(glmmTMB)
## where subj is some random effect
test_model <- glmmTMB(y ~ x_var1 + (1 | subj), data = test_data, family = binomial)
output_lists <- BootGlmm(test_model, 199, base_data = test_data, return_coefs_instead = TRUE)
BootCI(output_lists$base_coef_se,
       output_lists$resampled_coef_se)
```

`BootGlmm` *computes bootstrap resamples of your data, stores estimates + SEs.*

Description

By default, this will compute bootstrap resamples and then send them to BootCI for calculation. Note - only use parallel methods if your model is expensive to build, otherwise the overhead won't be worth it.

Usage

```
BootGlmm(base_model, resamples = 9999, base_data = NULL,
  return_coefs_instead = FALSE, resample_specific_blocks = NULL,
  unique_resample_lim = NULL, narrowness_avoid = TRUE,
  num_cores = DetectCores() - 1, suppress_sampling_message = FALSE,
  suppress_loading_bar = FALSE, allow_conv_error = FALSE)
```

Arguments

`base_model`	The pre-bootstrap model, i.e. the model output from running a standard model call. Examples: base_model <- glmmTMB(y ~ age + (1 | subj), data = rel_data, family = binomial) base_model <- lm(y ~ x) Assumes the data is accessible by base_model$model (typical lm or glm), or base_model$frame (works for glmmTMB), or base_model@frame (works for lmer, glmer) For now, if you have a different case, extract and send as base_data.
`resamples`	How many resamples of your data do you want to do? 9999 is a reasonable default (see Hesterberg 2015), but start very small to make sure it works on your data properly, and to get a rough timing estimate etc.
`base_data`	Default NULL; In a future version this will become mandatory to supply. In some cases, it can be extracted from the base_model, but this can produce bugs. It's recommended that you supply your data.
`return_coefs_instead`	Logical, default FALSE: do you want the list of lists of results for each bootstrap sample (set to TRUE), or the matrix output of all samples? See return for more details.

`resample_specific_blocks`
: Character vector, default NULL. If left NULL, this algorithm with choose ONE random block to resample over - the one with the largest entropy (often the one with most levels). If you wish to resample over specific random effects as blocks, enter the names here - can be one, or many. Note that resampling multiple blocks is in general quite conservative.

  If you want to perform case resampling but you do have random effects, set resample_specific_blocks to any non-null value that does not contain any random effect variable names.

`unique_resample_lim`
: Should be same length as number of random effects (or left NULL). Do you want to force the resampling to produce a minimum number of unique values in sampling? Don't make this too big... Must be named same as rand cols

`narrowness_avoid`
: Boolean, default TRUE. If TRUE, will resample n-1 instead of n elements in the bootstrap (n being either rows, or random effect levels, depending on existence of random effects). If FALSE, will do typical size n resampling.

`num_cores`
: Defaults to detectCores() - 1 if parallel is loaded, or just 1 if not. How many cores to use if doing parallel work?

`suppress_sampling_message`
: Logical, default FALSE. By default, this function will message the console with the type of bootstrapping: block resampling over random effects - in which case it'll say what effect it's sampling over; case resampling - in which case it'll say as much. Set TRUE to hide message.

`suppress_loading_bar`
: Logical, default FALSE. If TRUE, uses standard mclapply for parallel work, else uses pblapply (from pbapply), which provides a nice loading bar for progress.

`allow_conv_error`
: logical, default FALSE; if the subsample runs but gives a convergence error, is this acceptable? or should we resample?

Value

Returns the output from BootCI: a matrix of output for each fixed variable, including the intercept (estimate, CIs for boot and base, p-values). If return_coefs_instead = TRUE, then will instead return a list of length two: [[1]] will be a matrix containing the base estimates and standard errors. [[2]] will be a list of length resamples, each a matrix of estimates and standard errors. This output is useful for error checking, and if you want to run this function in a distributed way.

Examples

```
x <- rnorm(20)
y <- rnorm(20) + x
xy_data = data.frame(x = x, y = y)
first_model <- lm(y ~ x, data = xy_data)

out_matrix <- BootGlmm(first_model, 20, base_data = xy_data)
out_list <- BootGlmm(first_model, 20, base_data = xy_data, return_coefs_instead = TRUE)
```

```
data(test_data)
library(glmmTMB)
test_formula <- as.formula('y ~ x_var1 + x_var2 + x_var3 + (1|subj)')
test_model <- glmmTMB(test_formula, data = test_data, family = binomial)
output_matrix <- BootGlmm(test_model, 199, base_data = test_data)

output_lists <- BootGlmm(test_model, 199, base_data = test_data, return_coefs_instead = TRUE)
```

`CombineResampledLists` *When running BootGlmm in distributed fashion, combines output*

Description

If you run BootGlmm on e.g. a grid of computers, set return_coefs_instead = TRUE for each. Then enter them all here. Either just list them out, or put them into one list and enter them.

Usage

```
CombineResampledLists(..., return_combined_list = FALSE)
```

Arguments

`...`	Say our output from BootGlmm from three separate computers is output_list1, output_list2, output_list3 We can run: CombineResampledLists(output_list1, output_list2, output_list3) OR: create a list of lists: output_list_list <- list(output_list1, output_list2, output_list3) and then: CombineResampledLists(output_list_list)
`return_combined_list`	Logical, default FALSE. TRUE if you want the combined list of lists, FALSE for just the output from BootCI applied to it.

Value

Returns the same output as BootCI by default, or the combined list (as if you had just run BootGlmm once with all resamples) if return_combined_list = TRUE

Examples

```
data(test_data)
library(glmmTMB)
## where subj is some RE
test_model <- glmmTMB(y ~ x_var1 + (1 | subj), data = test_data, family = binomial)
output_list1 <- BootGlmm(test_model, 99, base_data = test_data, return_coefs_instead = TRUE)
output_list2 <- BootGlmm(test_model, 100, base_data = test_data, return_coefs_instead = TRUE)
output_list3 <- BootGlmm(test_model, 100, base_data = test_data, return_coefs_instead = TRUE)
CombineResampledLists(output_list1, output_list2, output_list3)
```

```
num_blocks = 10
num_total_resamples = 299
reg_list <- list()
for(i in 1:num_blocks){
    if(i < num_blocks){
        block_resamples = floor((num_total_resamples + 1)/num_blocks)
    } else {
        block_resamples = floor((num_total_resamples + 1)/num_blocks - 1)
    }
    reg_list[[i]] = BootGlmm(test_model,
                             resamples = block_resamples,
                             base_data = test_data,
                             return_coefs_instead = TRUE,
                             num_cores = 1, ## increase for parallel
                             suppress_loading_bar = TRUE)
}
boot_ci1 <- CombineResampledLists(reg_list)
full_list <- CombineResampledLists(reg_list, return_combined_list = TRUE)
boot_ci2 <- BootCI(full_list$base_coef_se, full_list$resampled_coef_se)
identical(boot_ci1, boot_ci2)
```

`GetRand` *this takes in a formula with bars and gives back the plain names of the columns*

Description

this takes in a formula with bars and gives back the plain names of the columns

Usage

```
GetRand(form_with_bars)
```

Arguments

`form_with_bars`	A formula used in e.g. lme4 and similar packages. Typically along the lines: y ~ age + (1 | school) etc

Value

A vector of the variables that are treated as random

Examples

```
GetRand('y ~ age + (1 | school)')
GetRand('y ~ income + (1 | school) + (1 | school:section)')
GetRand('y ~ income + (1 | school) + (1 | school/section)')
GetRand(as.formula('y ~ x + (1 | z)'))
GetRand('y ~ x')
```

`test_data` *Simulated data containing three fixed effects and one random effect*

Description

A small normal dataset with a proportional outcome to illustrate the use of this package. The outcome has mean expit(0.5 + 0.1 * x_var1 + 0.2 * x_var2 + 0.3 * x_var3 + SUBJ_VAL) where SUBJ_VAL are the values of the random effect The SD of y is then shrunk by 0.9 relative to a binomial distribution, and then beta values are generated. Arbitrarily close to the endpoints gives zeros and ones.

Usage

```
test_data
```

Format

A data frame with 300 rows and 4 variables:

x_var1 independent normally distributed variable

x_var2 independent normally distributed variable

x_var3 independent normally distributed variable

subj levels of random effect

y outcome: lives in interval [0,1]

Index