blblm Package

Diana Rueda

03/17/2021

```
library(blblm)
```

Implements linear regression analysis through Bag of Little Bootstraps procedure which combines features from bootstrap and subsampling. This package has the option to use multiple cores for faster processing.
Bag of little bootstrap consists on sampling n items from b items available with replacement. It samples s times into sizes of b. Then for each subsample it resamples each r times until sample size is n. It then computes the bootstrap statistic for each bootstrap sample and the statistic from the bootstrap statistics. Finally it takes the average of the statistics.

Functions

blblm

Builds blblm object

Parameters:
formula: Linear regression formula of the form y ~ x. It can be multivariable linear regression analysis.
data: Dataframe containing response and independent variables y and x.
m: Integer indicating the number of items to choose from.
B: Integer indicating the number of times to replicate the linear regression fitting process.
workers: Integer indicating the number of workers. If workers has a value larger than one the process will run in paralelization.
fastlm: If true the algorithm will run with function fastLm if false it will use lm.wfit.

Returns: object class blblm

Creates linear regression objects with little bag of bootstraps. Splits the data into subsamples, and uses future_map when paralization is chosen and map when only using one core to apply the indicated lm function to the subsample.

split_data

Splits data into m parts of approximated equal sizes. Called through blblm() for bootstrap sampling.

Parameters:

data: Dataframe containing response and independent variables y and x.
m: Integer indicating the number of items to choose from.

Returns: List of data frames representing subsamples

Uses `sample.int(m, nrow(data), replace = TRUE)` to reorganize the data given into subsamples. Then uses `split()` to split the observations into groups.

lm_each_subsample

Computes the regression model estimates, and replicates lm1 process. Called through `blblm()` as mapping funciton.

Parameters:
formula: Linear regression formula of the form y ~ x. It can be multivariable linear regression analysis.
data: Dataframe containing response and independent variables y and x.
m: Integer indicating the number of items to choose from.
B: Integer indicating the number of times to replicate the linear regression fitting process.
workers: Integer indicating the number of workers. If workers has a value larger than one the process will run in paralelization.
fastlm: If true the algorithm will run with function fastLm if false it will use lm.wfit.

Returns: 'rpdb' with replicated atomic coordinates

Strctures regression model with methods `.frame()`, `.matrix()` and `.response()` for each subsample and replicates lm1 `B` times without simplification.

lm1

Computes the regression estimates for a blb dataset. Called through lm_each_subsample.

Parameters:

X: Data with independent variables obtained from `model.matrix()`
y: Vector of response data obtained from `model.response()`
n: Number of rows of complete data before subsampling.
fastlm: If true the algorithm will run with function fastLm if false it will use lm.wfit.

Returns: list of model coefficients

Calculates weigths to simulate distribution with the help of `rmultinom()`, and fits model either through lm.wfit or fastLm. Then, calls `blbcoef()` and `sigma()` to compute fit coefficients and places them in a list.

blbcoef

Computes the coefficients from model fit

Parameters:

fit: Fitted blblm model

Returns: list of model coefficients

Obtains blblm model coefficients through `coef()`.

blbsigma

Computes sigma from model fit

Parameters:

fit: Fitted blblm model

Returns: numeric sigma value

Calculates sigma from formula with use of model rank, residuals and weights.

Methods

print.blblm

```
print.blblm <- function(x, ...) {
  cat("blblm model:", capture.output(x$formula))
  cat("\n")
}
```

sigma.blblm

Calculates sigma if `confidence = FALSE` and returns sigma confidence interval if `confidence = TRUE`.

```
sigma.blblm <- function(object, confidence = FALSE, level = 0.95, ...) {
  est <- object$estimates
  sigma <- mean(map_dbl(est, ~ mean(map_dbl(., "sigma"))))
  if (confidence) {
    alpha <- 1 - 0.95
    limits <- est %>%
      map_mean(~ quantile(map_dbl(., "sigma"), c(alpha / 2, 1 - alpha / 2))) %>%
      set_names(NULL)
    return(c(sigma = sigma, lwr = limits[1], upr = limits[2]))
  } else {
    return(sigma)
  }
}
```

coef.blblm

Calculates coefficients estimates through row means and reduce.

```
#' @export
#' @method coef blblm
coef.blblm <- function(object, ...) {
  est <- object$estimates
  map_mean(est, ~ map_cbind(., "coef") %>% rowMeans())
}
```

confint.blblm

Calculates confidence interval for coefficient estimates.

```
confint.blblm <- function(object, parm = NULL, level = 0.95, ...) {
  if (is.null(parm)) {
    parm <- attr(terms(object$formula), "term.labels")
  }
  alpha <- 1 - level
  est <- object$estimates
  out <- map_rbind(parm, function(p) {
    map_mean(est, ~ map_dbl(., list("coef", p)) %>% quantile(c(alpha / 2, 1 - alpha / 2)))
  })
  if (is.vector(out)) {
    out <- as.matrix(t(out))
  }
  dimnames(out)[[1]] <- parm
  out
}
```

predict.blblm

Calculates prediction from new data. If `confidence = TRUE` prediction interval is returned.

```
predict.blblm <- function(object, new_data, confidence = FALSE, level = 0.95, ...) {
  est <- object$estimates
  X <- model.matrix(reformulate(attr(terms(object$formula), "term.labels")), new_data)
  if (confidence) {
    map_mean(est, ~ map_cbind(., ~ X %*% .$coef) %>%
      apply(1, mean_lwr_upr, level = level) %>%
      t())
  } else {
    map_mean(est, ~ map_cbind(., ~ X %*% .$coef) %>% rowMeans())
  }
}
```

Results

```
library(blblm)
library(future)
library(furrr)
library(ggplot2)
library(bench)
```

The following code generates data to use in blblm to benchmark time efficiency.

```
library(tibble)
y = rnorm(100000, mean = 15, sd = 3)
x1 = 0.4 * y + rnorm(100000, sd = 0.25)
x2 = 0.7 * y + rnorm(100000, sd = 0.25)
X = tibble(x1,x2,y)
```

The first result saved in blblm is the result of using the configuraiton of the blblm package before any alterations for efficiency. The result saved in blblm_4wkrs is the result of the configuration where we run the blb process with 4 workers and fastLm. Bellow are the results.

```
result = bench::mark(
  blblm = blblm(y ~ x2 * x1, data = X, m = 5, B = 1000, workers = 1, fastlm = FALSE),
  blblm_4wkrs = blblm(y ~ x2 * x1, data = X, m = 5, B = 1000, workers = 4, fastlm = TRUE),
  relative = FALSE,
  check = FALSE,
  iterations = 10
)
#> Warning: Some expressions had a GC in every iteration; so filtering is disabled.
result
#> # A tibble: 2 x 6
#>   expression      min   median `itr/sec` mem_alloc `gc/sec`
#>   <bch:expr> <bch:tm> <bch:tm>     <dbl> <bch:byt>    <dbl>
#> 1 blblm       49.9s      51s    0.0185    29.4GB    9.52
#> 2 blblm_4wkrs   18.6s    18.9s    0.0519    14.9MB    0.0156
```

Example

```
fit <- blblm(mpg ~ wt * hp, data = mtcars, m = 3, B = 100)
coef(fit)
#> (Intercept)          wt          hp       wt:hp
#> 46.91281652 -7.06338185 -0.11374909  0.02395591
```

```
confint(fit, c("wt", "hp"))
#>                2.5%      97.5%
#> wt -7.0631819 -7.0631819
#> hp -0.1137491 -0.1137491
```

```
sigma(fit)
#> [1] 0
```

```
sigma(fit, confidence = TRUE)
#> sigma   lwr   upr
#>     0     0     0
```

```
predict(fit, data.frame(wt = c(2.5, 3), hp = c(150, 170)))
#>        1        2
#> 21.17546 18.60284
```

```
predict(fit, data.frame(wt = c(2.5, 3), hp = c(150, 170)), confidence = TRUE)
#>        fit      lwr      upr
#> 1 21.17546 21.17546 21.17546
#> 2 18.60284 18.60284 18.60284
```