blbLinReg vignette

Preface

This following document was developed to help those that are curious about the usage of bag of little bootstrap for linear regression. The idea behind the bag of little bootstrap sampling method is to get appropriate estimators from a given sample without having to generate new data. This method is useful when dealing with big data because it does a multinomial sampling scheme which is more efficient than repeated sampling. How is bag of little bootstrap for linear regression implemented? First, seperate the sample size into m groups, then sample each group using the multinomial scheme repeated r times. For each subsample, perform the estimation, either the mean, median, confidence interval, or even a linear model

Abstract

This vignette is used to help users run the package `blbLinReg`. This package implements the bag of little bootstraps on linear regressions. Our package has three main functions for the users to compute the bag of little bootstraps confidence intervals for linear regression estimators, sigma and predictors. There are helper functions in the package which are anonymous to users but their details include how we resample, split, and map different functions to the data and compute confidence intervals accordingly. There are helpful guides that users can refer to when confused `?blbLinReg::func`

Intro

Bag of little boostrap is compatible with parallel computing algorithms while keeping the statistical efficiency and generic applicability of the original bootstrap.

Setting Up

```r
library(blbLinReg)
```

Functions in Package

Blb.sigma.ci

A function that takes in 6 parameters including the numerical data (data), responsive and independent variables (y and x), the number of groups to split the data into (m), the number of iteration each boot (r) and the significance level (alpha). The function calls a helper function named blb.sigma.ci.list to compute the bag of little bootstrap confidence intervals for sigma of each boot for the linear model. It first splits data with groupSplit function and bootstrap each subsample with each_boot function and uses purrr::map to compute sigmas and their CIs. Then it calls avg_ci to take the average value of the list and returns the confidence interval the user desires.

For sequential computing

```
blb.sigma.ci(data = iris, y = "Sepal.Length", x = c("Sepal.Width",
  "Petal.Width"), m = 10, r = 15, alpha =  0.05)
#>     2.5%   97.5%
#> 1.35066 1.59907
```

For parallel computing

```
blb.sigma.ci(data = iris, y = "Sepal.Length", x = c("Sepal.Width",
  "Petal.Width"), m = 10, r = 15, alpha =  0.05, parallel = TRUE,
  n_cores = 4)
#>      2.5%    97.5%
#> 1.301111 1.513971
```

blb.coef.ci

blb.coef.ci asks for the exact same parameters as Blb.sigma.ci, also check its parameters before calling the helper functions. It has a different helper function, blb.coef.ci.list, which splits the data with the groupSplit function as well. Everything else is the same as blb.sigma.ci.list except blb.coef.ci.list computes confidence intervals for linear regression estimators.

For sequential computing

```
blb.coef.ci(iris, "Sepal.Length", c("Sepal.Width", "Petal.Width"),
  10, 15, 0.05)
#>       (Intercept) Sepal.Width Petal.Width
#> 2.5%     3.337029   0.2236909   0.9559569
#> 97.5%    4.033636   0.4210018   1.0918748
```

For parallel computing

```
blb.coef.ci(iris, "Sepal.Length", c("Sepal.Width", "Petal.Width"),
  10, 15, 0.05, parallel = T, n_cores = 4)
#>       (Intercept) Sepal.Width Petal.Width
#> 2.5%     3.574701  0.02706894   0.9198097
#> 97.5%    4.635826  0.34560540   1.0757682
```

blb.pred.ci

For sequential computing

```
pred_df <- dplyr::tibble(Sepal.Length = c(1,2,3,4,5), Sepal.Width =
  c(1,2,3,4,5))
blb.pred.ci(iris, pred_df, "Petal.Width", c("Sepal.Length",
  "Sepal.Width"), 10, 15, 0.05)
#>               1          2          3          4          5
#> 2.5%  -2.025920 -1.4908319 -0.9721255 -0.5248883 -0.1569675
#> 97.5% -1.311151 -0.9839153 -0.6166787 -0.1643941  0.3487642
```

For parallel computing

```
blb.pred.ci(iris, pred_df, "Petal.Width", c("Sepal.Length",
  "Sepal.Width"), 10, 15, 0.05, parallel = TRUE, n_cores = 4)
#>               1         2          3          4           5
#> 2.5%  -2.054276 -1.459005 -0.8869375 -0.4027636 -0.03429491
#> 97.5% -1.188167 -0.872672 -0.5381189 -0.1019767  0.44931099
```

Helper Functions

The following functions are used to `help` do certain jobs seeminglessly

groupSplit

Takes the original data and separates it into a number of groups users want.

`groupSplit(iris,10)`

each_boot

A function that calculates linear regression based on weights using multinomial distribution. `each_boot(subSample, n, r, y, x)`

each_boot_pred

A helper function that calculates prediction of a linear model based on weights from the multinomial distribution `each_boot_pred(subSample, n, r, y, x, pred_df)`

blb.ceof.ci.list

A helper function for `blb.coef.ci`. This function takes the exact same parameter as `blb.coef.ci` does. It first calls groupSplit to split the data and map each_boot onto the subsamples. Then it calculates the bootstrap percentile for the regression estimator and returns a list of the percentile confidence intervals `blb.coef.ci.list(data, y, x, m, r = 10, alpha = 0.05)`

blb.sigma.ci.list

A helper function for `blb.pred.ci`. It first calls groupSplit to split the data and map each_boot_pred onto the subsamples. Then it calculates the bootstrap percentile for the bootstraps percentile for the prediction values and returns a list of the percentile confidence intervals

`blb.pred.ci.list(data, pred_df, y, x, m, r, alpha)`

avg_ci

A helper function that takes the average of a confidence list

`avg_ci(ci_list)`