final-project-vignette

The `blblm` package is helpful for making linear models using the Bag of Little Bootstraps (BLB) algorithm. BLB is a good way to deal with bootstrapping for big data, so we provide some examples of how to use it over traditional bootstrap methods here for the user's understanding.

First, please make sure the package is installed. Please run `install.packages(blblm)` if needed. Note that the `blblm` package has dependencies with the following packages: `purrr`, `stats`, `magrittr %>%`, `parallel`, and `furrr`

```
library(blblm)
library(future)
```

We start by creating a a BLB Linear Model with the `blblm` function. We will be using the `Theoph` datset for demonstration purposes.

```
head(Theoph)
#>   Subject   Wt Dose Time  conc
#> 1       1 79.6 4.02 0.00  0.74
#> 2       1 79.6 4.02 0.25  2.84
#> 3       1 79.6 4.02 0.57  6.57
#> 4       1 79.6 4.02 1.12 10.50
#> 5       1 79.6 4.02 2.02  9.66
#> 6       1 79.6 4.02 3.82  8.58
```

We can use `blb.lm` to create a BLB Linear Model. To speed up computation, we can set `parallelize = TRUE` to allow the user to use more cores for faster calculations. Please note that the number of cores available to use depends on the user's computer.

```
blb.lm <- blblm::blblm(Wt ~ Dose * conc, m = floor(sqrt(nrow(Theoph))), data = Theoph, parallelize
  = TRUE)
blb.lm2 <- blblm::blblm(Wt ~ Dose * conc, m = floor(sqrt(nrow(Theoph))), data = Theoph,
  parallelize = FALSE)
```

We can view our model with the `print.blblm` function.

```
print(blb.lm)
#> blblm model: Wt ~ Dose * conc
```

Using this model we created, we can get summary statistics and model information. For example, we can get the coefficients using `coef(blb.lm)`

```
coef(blb.lm)
#> (Intercept)        Dose        conc   Dose:conc
#> 135.5183804 -14.2962452  -1.0027698   0.2397703
```

We will benchmark to see how the parallelized version compares to a non-parallelized version for a dataset

we generate here.

```
new.x <- runif(10000)
new.x2 <- rnorm(10000)
new.y <- rpois(10000, 5)

new.df <- data.frame(new.x, new.x2, new.y)

bench::mark(
  coef(blblm(new.y ~ new.x * new.x2, m = 10, data = new.df, parallelize = TRUE)),
  coef(blblm::blblm(new.y ~ new.x * new.x2, m = 10, data = new.df, parallelize = FALSE)),
  check = FALSE
)
#> Warning: Some expressions had a GC in every iteration; so filtering is disabled.
#> # A tibble: 2 x 6
#> # ... with 6 more variables: expression <bch:expr>, min <bch:tm>,
#> #   median <bch:tm>, itr/sec <dbl>, mem_alloc <bch:byt>, gc/sec <dbl>
```

We see that the parallelized version is faster, so it should be used when datasets are large. For smaller datasets, parallelization may be slower than a non-parallelized version.

Another common use of a BLB'd Linear Model is to create confidence intervals for certain parameters.

```
confint(blb.lm, c("Dose", "conc"))
#>             2.5%       97.5%
#> Dose -14.905255 -13.7141643
#> conc  -1.534285  -0.4674047
```

We can calculate σ for our model with the `sigma` function.

```
sigma(blb.lm)
#> [1] 0.852151
```

We can also create a confidence interval for σ with the `confidence` parameter.

```
sigma(blb.lm, confidence = TRUE)
#>     sigma       lwr       upr
#> 0.8521510 0.7479628 0.9421498
```

We can also predict values using new parameters with the `predict` function.

```
predict(blb.lm, data.frame(Dose = c(3.9, 4.0), conc = c(1, 1.5), Wt = c(80, 85)))
#>        1        2
#> 79.69536 78.26787
```

Additionally, we can create confidence intervals for these parameters.

```
predict(blb.lm, data.frame(Dose = c(3.9, 4.0), conc = c(1, 1.5), Wt = c(80, 85)), confidence =
  TRUE)
#>        fit      lwr      upr
#> 1 79.69536 79.09996 80.30021
#> 2 78.26787 77.76493 78.78087
```

We also test our speed for model creation and compare the c++ version fastlm with base R lm.

```
x <- runif(100)
y <- runif(100)
df <- data.frame(x = x, y = y)
bench::mark(
  blblm(y ~x, data = df, m = 10, use_cpp = FALSE, B = 1000),
  blblm(y ~ x, data = df, m = 10, use_cpp = TRUE, B = 1000),
  check = FALSE
)
#> # A tibble: 2 x 6
#>   expression                                                  min   median
#>   <bch:expr>                                                <bch:tm> <bch:tm>
#> 1 blblm(y ~ x, data = df, m = 10, use_cpp = FALSE, B = 1000)    669ms    669ms
#> 2 blblm(y ~ x, data = df, m = 10, use_cpp = TRUE, B = 1000)     201ms    219ms
#> # ... with 3 more variables: itr/sec <dbl>, mem_alloc <bch:byt>, gc/sec <dbl>
```

For more information on functions in this package, please run `?[function_name]` to see the documentation page we have updated via ROxygen. These documentations were created for each function in this package to be used. Our functions tend to need `data.frame` objects and often call on the model we generate with the `blblm` function. There are some under the hood functions in the package that are not exported, but still documented for clarity's sake.