User Guide

This user guide shows how to use the STA141CFinal package. The dataset that will be used is the SGEMM GPU Kernel Performance dataset of the UC Irvine Machine Learning Repository, and it is part of the package.

```
library(STA141CFinal)
```

```
## Loading required package: purrr
```

```
## Loading required package: furrr
```

```
## Loading required package: future
```

```
data("sgemm_product")
```

The `linear_reg_bs` function splits the given dataset into `s` samples, then generates `r` bootstrap samples from each sample. By default, `s` is 10 and `r` is 1000. Afterwards, a linear regression model is fit on all bootstrap samples, and the linear regression coefficient estimates, as well as the error variance estimates, are recorded and returned. This is the first step of a Bag of Little Bootstraps (BLB) procedure for multiple linear regression. Because this operation requires a lot of time and memory, users have the option to use `linear_reg_bs_par` instead. This function is the same as `linear_reg_bs` except that it uses parallel processing, thus using less time and memory. Users can also use `linear_reg_bs_C`, which is the `linear_reg_bs` function written in C++, for the same benefit.

```
x <- sgemm_product[-15]
y <- sgemm_product$`Run1 (ms)`
plan(multiprocess, workers = 4) # Needed for parallelization to work

# Ideally, r should be much larger, but then the code would take too long to run.
bench::mark(lrbs <- linear_reg_bs(x, y, 5, 50))
bench::mark(lrp <- linear_reg_bs_par(x, y, 5, 50))
bench::mark(lrc <- linear_reg_bs_C(x, y, 5, 50))
```

```
## # A tibble: 1 x 6
##   expression                          min median `itr/sec` mem_alloc `gc/sec`
##   <bch:expr>                        <bch:t> <bch:>     <dbl> <bch:byt>    <dbl>
## 1 lrbs <- linear_reg_bs(x, y, 5, 50)   1.82m  1.82m   0.00917    39.1GB     3.90
## # A tibble: 1 x 6
##   expression                           min median `itr/sec` mem_alloc
##   <bch:expr>                         <bch> <bch:>     <dbl> <bch:byt>
## 1 lrp <- linear_reg_bs_par(x, y, 5, 50) 1.13m  1.13m    0.0148      82MB
## # ... with 1 more variable: `gc/sec` <dbl>
## # A tibble: 1 x 6
##   expression                          min median `itr/sec` mem_alloc `gc/sec`
##   <bch:expr>                        <bch:> <bch:>     <dbl> <bch:byt>    <dbl>
## 1 lrc <- linear_reg_bs_C(x, y, 5, 50)  1.43m  1.43m    0.0117    41.8KB        0
```

After the `linear_reg_bs` object is created, it can be used to determine confidence intervals for the regression coefficients and the error variance. It can also be used to determine prediction intervals for new data. Note that the significance level (`alpha` = 0.05 by default) is only accurate for single intervals, and that it should be adjusted accordingly if multiple intervals are to be generated.

For error variance (`s2_CI_par` or `s2_CI_C` can also be used if so desired):

```
lrs2ci <- s2_CI(lrbs)
lrs2ci
```

```
## Lower_Bound    Estimate Upper_Bound
##    395851.7    402886.1    409103.6
```

For regression coefficients (`coef_CI_par` or `coef_CI_C` can also be used if so desired):

```
# This example produces Bonferroni-corrected intervals with 90% confidence for four
# coefficients (the first one is for the intercept)
lrcci <- coef_CI(lrbs, alpha = (0.1 / 4))[1:4, ]
lrcci
```

```
##           Lower_Bounds  Estimates Upper_Bounds
## Intercept   -58.366942 -51.940862   -45.329240
## MWG           3.292529   3.328012     3.362458
## NWG           3.043071   3.075988     3.110098
## KWG           5.041982   5.212429     5.367375
```

For new data (`PI_par` or `PI_C` can also be used if so desired):

```
# This example produces Bonferroni-corrected intervals with 90% confidence for four
# observations
newdata <- x[1:4,]
lrndpi <- PI(lrbs, newdata, alpha = (0.1 / 4))
lrndpi
```

```
##      Lower_Bounds  Estimates Upper_Bounds
## [1,]   -115.31590 -110.91475   -106.02120
## [2,]    -67.14934  -63.67233    -59.30874
## [3,]    -76.63545  -72.35305    -68.21988
## [4,]    -28.81410  -25.11063    -21.53095
```