Bag of little bootstraps

```
library(tidyverse)
```

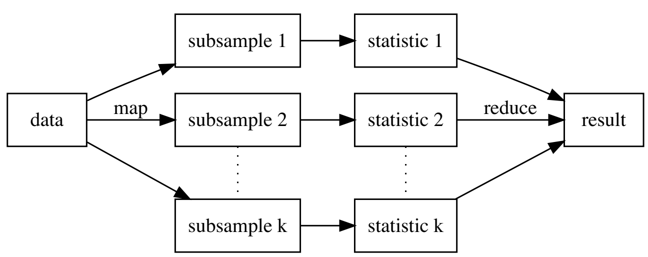
```
## -- Attaching packages --------------------------------------- tidyverse 1.3.0 --
```

```
## v ggplot2 3.2.1     v purrr   0.3.3
## v tibble  2.1.3     v dplyr   0.8.4
## v tidyr   1.0.2     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.4.0
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

Divide and conquer a.k.a. mapreduce

Divide and conquer allows a single task operation to be executed parallelly.



We have seen that in assignment 3 how we could use map and reduce to compute the mean.

```
library(nycflights13)
set.seed(141)
m <- 10
groups <- sample(seq_len(m), nrow(flights), replace = TRUE)
flights_list <- flights %>% split(groups)
```

```
mean_list <- flights_list %>% map(~ mean(.$dep_delay, na.rm = TRUE))
(mean_dep_delay <- mean_list %>% reduce(`+`) / m)
```

```
## [1] 12.63861
```

You may wonder if you could do the same for confidence intervals.

```
ci_list <- flights_list %>% map(~ t.test(.$dep_delay)$conf.int)
(mean_ci <- ci_list %>% reduce(`+`) / m)
```

```
## [1] 12.20391 13.07331
## attr(,"conf.level")
## [1] 0.95
```

Yeah, it gives us a result. But wait, it doesn't look right. Though the mapreduce procedure speeds up the computation, it should give similar result as if we work on the whole dataset.

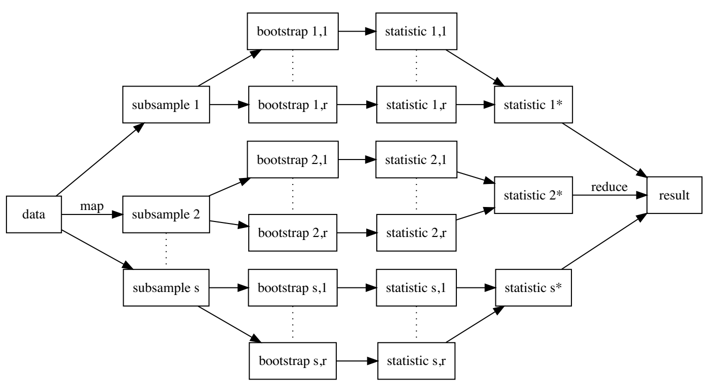
```
t.test(flights$dep_delay)$conf.int
```

```
## [1] 12.50157 12.77657
## attr(,"conf.level")
## [1] 0.95
```

Lesson learned: we cannot combine any statistics in the reduce step by simply taking the average. We may need to scale the statistics analytically which could be hard or impossible.

The bag of little bootstraps (BLB)

It is a procedure which incorporates features of both the bootstrap and subsampling to yield a robust, computationally efficient means of assessing the quality of estimators



- sample without replacement the sample s times into sizes of b
- for each subsample
  - resample each until sample size is n, r times
  - compute the bootstrap statistic (e,g., the mean) for each bootstrap sample
  - compute the statistic (e.g., confidence interval) from the bootstrap statistics
- take the average of the statistics

Bascially, the bag of little bootstraps = subsample + bootstrap. However, for each bootstrap, we sample n from b with replacement instead of sample b from b as in oridinary bootstrap.

A naive (single core) implementation

```
r <- 10  # r should be at least a few thousands, we are using 10 for demo
n <- nrow(flights)

each_boot <- function(i, data) {
  mean(sample(data, n, replace = TRUE), na.rm = TRUE)
}

ci_list <- flights_list %>% map(~ {
  sub_dep_delay <- .$dep_delay
  map_dbl(seq_len(r), each_boot, data = sub_dep_delay) %>%
    quantile(c(0.025, 0.975))
})

reduce(ci_list, `+`) / length(ci_list)
```

```
##     2.5%    97.5%
## 12.53957 12.74183
```

The `sample` above is not memory and computationally efficient.

```
# the frequency table of selecting 1000 items from 1:10 with replacement
table(sample(1:10, 100, replace = TRUE))
```

```
##
##  1  2  3  4  5  6  7  8  9 10
##  9  9 14 10  6 12  9 14  9  8
```

A more efficent way is to first generate the repeitions by multinomial distribution.

```
rmultinom(1, 100, rep(1, 10))
```

```
##       [,1]
##  [1,]    8
##  [2,]   10
##  [3,]   12
##  [4,]   12
##  [5,]   12
##  [6,]    7
##  [7,]    8
##  [8,]   14
##  [9,]    9
## [10,]    8
```

Compute the mean with the frequencies

```
sub_dep_delay <- flights_list[[1]]$dep_delay
# it's important to remove the missing values in this step
sub_dep_delay <- sub_dep_delay[!is.na(sub_dep_delay)]
freqs <- rmultinom(1, n, rep(1, length(sub_dep_delay)))
sum(sub_dep_delay * freqs) / n
```

```
## [1] 12.60671
```

Put everything back

```
r <- 10  # r should be at least a few thousands, we are using 10 for demo
n <- nrow(flights)

each_boot2 <- function(i, data) {
  non_missing_data <- data[!is.na(data)]
  freqs <- rmultinom(1, n, rep(1, length(non_missing_data)))
  sum(non_missing_data * freqs) / n
}

ci_list <- flights_list %>% map(~ {
  sub_dep_delay <- .$dep_delay
  map_dbl(seq_len(r), each_boot2, data = sub_dep_delay) %>%
    quantile(c(0.025, 0.975))
})

reduce(ci_list, `+`) / length(ci_list)
```

```
##     2.5%    97.5%
## 12.54735 12.73536
```

A parallel version using `furrr`.

```
library(furrr)
plan(multiprocess, workers = 5)
```

```
ci_list <- flights_list %>% future_map(~ {
  sub_dep_delay <- .$dep_delay
  map_dbl(seq_len(r), each_boot2, data = sub_dep_delay) %>%
    quantile(c(0.025, 0.975))
})
reduce(ci_list, `+`) / length(ci_list)
```

```
##     2.5%    97.5%
## 12.54574 12.76516
```

A (slow) benchmark

```
r <- 500
naive <- function() {
  flights_list %>% map(~ {
    sub_dep_delay <- .$dep_delay
    map_dbl(seq_len(r), each_boot, data = sub_dep_delay) %>%
      quantile(c(0.025, 0.975))
  })
}
improve <- function() {
```

```r
  flights_list %>% map(~ {
    sub_dep_delay <- .$dep_delay
    map_dbl(seq_len(r), each_boot2, data = sub_dep_delay) %>%
      quantile(c(0.025, 0.975))
  })
}
multi_core <- function() {
  flights_list %>% future_map(~ {
    sub_dep_delay <- .$dep_delay
    map_dbl(seq_len(r), each_boot2, data = sub_dep_delay) %>%
      quantile(c(0.025, 0.975))
  })
}
```

```r
# system.time(naive())  # [skipped] take forver
system.time(improve())  # 4x seconds
system.time(multi_core()) # 1x seconds
```