Week Ten Tasks

Peter Boyd

11/25/2018

Need to compare timing of a few aspects within my function. The apply method below is much faster. I had hoped to use the map function, but I don't think that the purrr package can do what I would need it to do in this situation (a by column operation for matricies). I found this discussion board that points to this conclusion: https://github.com/tidyverse/purrr/issues/341

As a result I implemented apply in my code.

```
library(tidyverse)
library(here)
library(MASS)
library(broom)
library(ggbeeswarm)
```

```
whale <- read_csv(here("data", "sperm_whale_pop.csv"))
```

```
## Parsed with column specification:
## cols(
##   Year = col_integer(),
##   Population = col_integer()
## )
```

```
ntimes <- 100

# setup needed
  df <- whale
  fit.poi <- glm(df$Population ~ df$Year, data = df, family = "poisson")
  best_model <- fit.poi
  n <- max(df$Year) - min(df$Year) +1
  y.hat <- best_model$fitted.values
  y.rep <- matrix(ncol = ntimes, data = c(rpois(n*ntimes, y.hat)))
  years <- c(min(df$Year):max(df$Year))

# using apply
  apply_method <- function(df, best_model, n, y.hat, y.rep){
  y.rep.sort <- -apply(-y.rep, 2, sort)
  y.coefs <- t(apply(y.rep.sort, 2, function(y.col) lm(y.col~years)$coef))
  end.dates <- -1* y.coefs[,1] / y.coefs[,2]
  return(end.dates)
  }

  apply_method(df, best_model, n, y.hat, y.rep)
```

```
##   [1] 2558.934 2577.318 2569.007 2551.198 2556.999 2577.303 2540.984
##   [8] 2562.693 2584.994 2589.143 2576.830 2590.650 2579.324 2574.598
##  [15] 2556.175 2570.868 2566.836 2577.997 2570.868 2548.303 2575.871
##  [22] 2567.968 2580.917 2565.659 2582.251 2559.069 2586.048 2562.023
##  [29] 2571.221 2567.119 2581.335 2575.847 2548.440 2588.201 2557.332
##  [36] 2557.611 2570.344 2554.550 2572.000 2589.329 2565.989 2581.891
##  [43] 2589.150 2566.711 2606.226 2584.425 2578.575 2565.689 2552.717
```

```
##  [50] 2583.258 2570.582 2571.976 2573.497 2576.274 2567.430 2592.127
##  [57] 2560.634 2556.609 2559.147 2572.326 2571.744 2577.679 2581.375
##  [64] 2585.476 2581.966 2596.716 2577.086 2579.970 2574.247 2552.709
##  [71] 2564.523 2547.613 2556.384 2575.814 2563.710 2562.716 2592.009
##  [78] 2576.618 2559.140 2560.477 2563.022 2571.770 2560.645 2566.718
##  [85] 2579.679 2558.901 2559.191 2591.224 2585.122 2547.935 2564.124
##  [92] 2571.617 2550.982 2582.263 2555.137 2561.210 2581.285 2546.668
##  [99] 2569.502 2571.126
```

```
  # using for loop
  loop_method <- function(df, best_model, n, y.hat, y.rep){
    end_dates <- double(ntimes)
    for (i in 1:ntimes){
      sim_reg <- lm(sort(y.rep[,i], decreasing = TRUE) ~
                  years)
      end_dates[i] <- -1*summary(sim_reg)$coefficients[1,1] /
         summary(sim_reg)$coefficients[2,1]
    }
    return(end_dates)
  }

  loop_method(df, best_model, n, y.hat, y.rep)
```

```
##   [1] 2558.934 2577.318 2569.007 2551.198 2556.999 2577.303 2540.984
##   [8] 2562.693 2584.994 2589.143 2576.830 2590.650 2579.324 2574.598
##  [15] 2556.175 2570.868 2566.836 2577.997 2570.868 2548.303 2575.871
##  [22] 2567.968 2580.917 2565.659 2582.251 2559.069 2586.048 2562.023
##  [29] 2571.221 2567.119 2581.335 2575.847 2548.440 2588.201 2557.332
##  [36] 2557.611 2570.344 2554.550 2572.000 2589.329 2565.989 2581.891
##  [43] 2589.150 2566.711 2606.226 2584.425 2578.575 2565.689 2552.717
##  [50] 2583.258 2570.582 2571.976 2573.497 2576.274 2567.430 2592.127
##  [57] 2560.634 2556.609 2559.147 2572.326 2571.744 2577.679 2581.375
##  [64] 2585.476 2581.966 2596.716 2577.086 2579.970 2574.247 2552.709
##  [71] 2564.523 2547.613 2556.384 2575.814 2563.710 2562.716 2592.009
##  [78] 2576.618 2559.140 2560.477 2563.022 2571.770 2560.645 2566.718
##  [85] 2579.679 2558.901 2559.191 2591.224 2585.122 2547.935 2564.124
##  [92] 2571.617 2550.982 2582.263 2555.137 2561.210 2581.285 2546.668
##  [99] 2569.502 2571.126
```

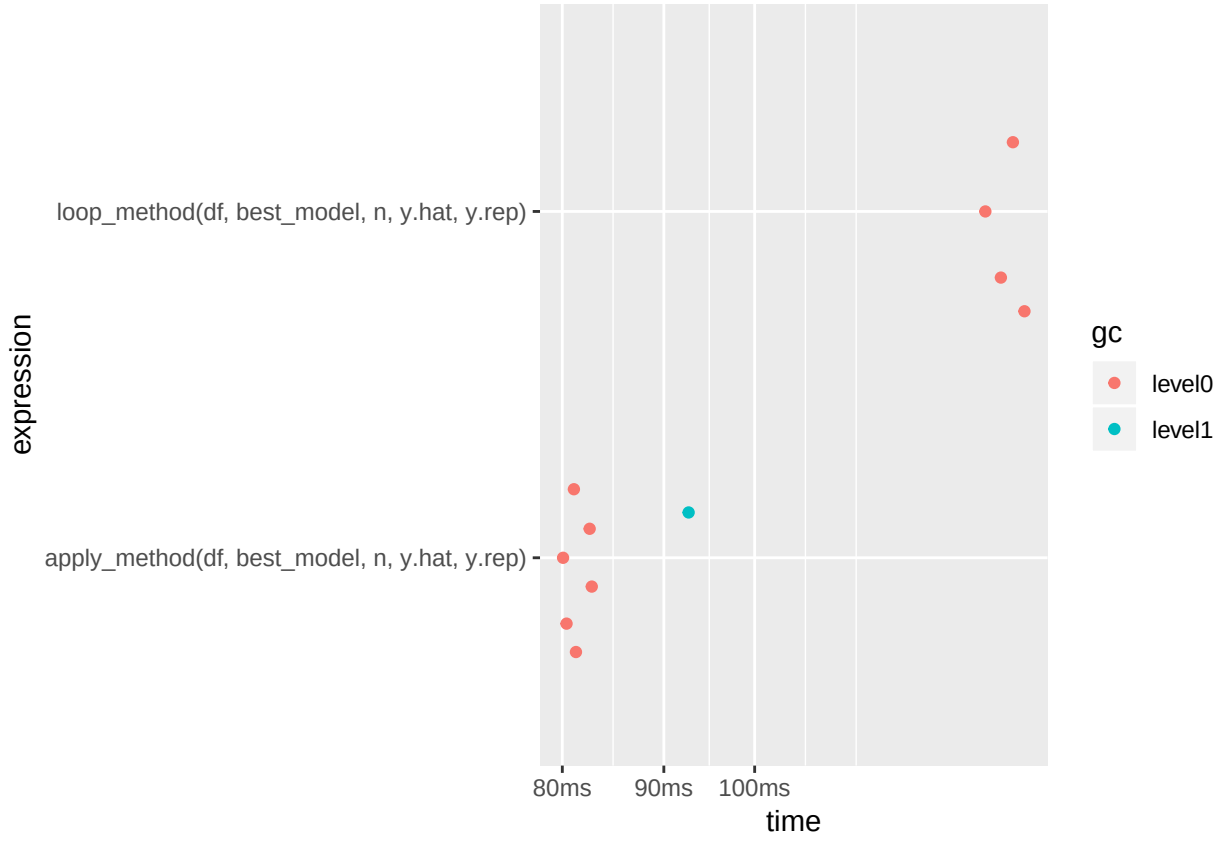
```
  times <- bench::mark(
  apply_method(df, best_model, n, y.hat, y.rep),
  loop_method(df, best_model, n, y.hat, y.rep)
)
```

```
## Warning: Some expressions had a GC in every iteration; so filtering is
## disabled.
```

```
times
```

```
## # A tibble: 2 x 10
##   expression     min  mean  median     max `itr/sec` mem_alloc  n_gc n_itr
##   <chr>      <bch:t> <bch> <bch:t> <bch:t>     <dbl> <bch:byt> <dbl> <int>
## 1 apply_met~  80.1ms  83ms  81.3ms  92.6ms     12.1      563KB    14     7
## 2 loop_meth~ 130.7ms 134ms   134ms 136.8ms      7.47     470KB    12     4
## # ... with 1 more variable: total_time <bch:tm>
```

```
plot(times)
```


loop_method(df, best_model, n, y.hat, y.rep)
apply_method(df, best_model, n, y.hat, y.rep)
expression
80ms
90ms
100ms
time
gc
level0
level1