New_Algorithm

Andrew Glover

2023-12-16

Setup

Loading Package Functions

```
devtools::load_all()
```

Libraries Used

```
library(tibble)
library(ggplot2)
library(magrittr)
library(dplyr)
library(patchwork)
# loads the functions from the package
```

We are going to assume that we know p_0 exactly. That way it is either a data point in actual performance, with no cost, or we don't count it as an actual performance.

Algorithm Functions:

Helpers

```
# function inside of parameter distance
lim_func <- function(k_1, k_2, tau_1, tau_2) {
  k_1*exp(-1/tau_1)/(1-exp(-1/tau_1))-k_2*exp(-1/tau_2)/(1-exp(-1/tau_2))
}

# computes "parameter distance"
params_dist <- function(params_1, params_2) {
  # ignore params_1[[1]] since it is just the inital value
  abs(lim_func(params_1[[2]], params_1[[3]], params_1[[4]], params_1[[5]]) -
        lim_func(params_2[[2]], params_2[[3]], params_2[[4]], params_2[[5]])
      )
}
```

```r
# plotting function
plot_perf <- function(observed_performance,
                      modeled_performance) {
  data <- tibble(
  "day" = c(0:length(observed_performance)),
  "pred" = modeled_performance,
  "obs" = c(NA,observed_performance)
  )

  plot <- ggplot(data, aes(x = day)) +
  geom_point(aes(y = pred, color = "pred")) +
  geom_point(aes(y = obs, color = "obs")) +
  labs(x = "Day",
       y = "Performance",
       title = "")
  plot
}
```

```r
update_cost <- function(params_i, old_params, obs_indexes, n, training_load, obs_perf,
                        lambda, min_cost_vec, min_params_mat
                      ) {
  params_dist_i <- params_dist(params_i, old_params)
  SSE_i <- 0
  p_0 <- params_i[[1]]
  k_1 <- params_i[[2]]; k_2 <- params_i[[3]]
  coef_1 <- exp(-1/params_i[[4]]); coef_2 <- exp(-1/params_i[[5]])
  T_1 <- 0; T_2 <- 0
  lower_index <- 1
  j <- 0
  for (new_index in obs_indexes) {
    for (t in lower_index:new_index) {
      T_1 <- coef_1*(T_1 + k_1*training_load[[t]])
      T_2 <- coef_2*(T_2 + k_2*training_load[[t]])
    }
    SSE_i <- SSE_i + (p_0 + T_1 - T_2 - obs_perf[[new_index]])^2
    j <- j + 1
    lower_index <- new_index + 1
    cost_ij <- sqrt(SSE_i/j) + lambda*params_dist_i
    if (cost_ij < min_cost_vec[[j]]) {
      min_cost_vec[[j]] <- cost_ij
      min_params_mat[j, ] <- params_i
    }
  }
  return(list("params" = min_params_mat, "cost" = min_cost_vec))
}


min_params_new <- function(params_matrix,
                        training_load,
                        obs_perf,
                        init_params,
                        lambda) {
  obs_indexes <- which(!is.na(obs_perf))
```

```
  n <- length(obs_indexes)
  min_cost_vec <- c(rep(Inf, n))
  min_params_mat <- matrix(0, nrow = n, ncol = 5)
  update_cost(init_params, init_params, obs_indexes, n, training_load, obs_perf, lambda, min_cost_vec, m
  for (i in 1:nrow(params_matrix)) {
    params_i <- as.numeric(params_matrix[i, ])
    res <- update_cost(params_i, init_params, obs_indexes, n, training_load, obs_perf, lambda, min_cost_
                       )
    min_cost_vec <- res$cost
    min_params_mat <- res$params
  }
  return(list("opt_params" = min_params_mat, "cost" = min_cost_vec))
}


# this function creates (or calls, we will see) the parameter matrix and applies
# it to the previous function. makes things tider in the algorithm funciton
searh_params_mat <- function(bounds_type = list("test")) {
  if(bounds_type == "test") {
    params_matrix <- expand.grid(p_0 = 500,
                                 k_1 = c(1,2,3,4),
                                 k_2 = c(2,4,6,8),
                                 tau_1 = c(5:35),
                                 tau_2 = c(5:35))
  }
  params_matrix <- params_matrix[params_matrix$tau_1 >= params_matrix$tau_2,]
  return(params_matrix)
}
```

```
set.seed(4)
days_test <- 200
perf_sim_1 <- c(500, perf_tv(p_0 = 500,
          k_1 = 1,
          tau_1 = 25,
          k_2 = 2,
          tau_2 = 10,
          days = days_test,
          training_stim = list("constant", 100))$performance
)
noise_fn <- function(x,
                     sd){
  x+rnorm(1, mean = 0, sd = sd)
}

perf_sim_1 = purrr::map_dbl(perf_sim_1, noise_fn, sd = 20)

# this is one off
samp_index <- c(sample(c(1:100), 50), sample(c(101:days_test), 10))
actual_perf <- rep(NA, days_test)
for (i in samp_index) {
  actual_perf[[i]] = perf_sim_1[[i+1]]

}
```

```
params_matrix_1 <- as.matrix(expand.grid(p_0 = 500,
                                         k_1 = c(1,2,3,4),
                                         k_2 = c(2,4,6,8),
                                         tau_1 = c(5:35),
                                         tau_2 = c(5:35))
)

res_1 <- min_params_new(params_matrix_1,
                training_load = c(rep(100,days_test)),
                actual_perf,
                c(500,1,2,25,10),
                lambda = .4)
res_1
```

```
## $opt_params
##       [,1] [,2] [,3] [,4] [,5]
##  [1,]  500    3    4   35   25
##  [2,]  500    3    4   15   10
##  [3,]  500    1    2   23    9
##  [4,]  500    1    2   23    9
##  [5,]  500    3    4    8    7
##  [6,]  500    1    2   11    7
##  [7,]  500    1    2   15    8
##  [8,]  500    1    2   20    9
##  [9,]  500    1    2   20    9
## [10,]  500    1    2   26   10
## [11,]  500    1    2   26   10
## [12,]  500    1    2   20    9
## [13,]  500    1    2   26   10
## [14,]  500    1    2   20    9
## [15,]  500    1    2   25   10
## [16,]  500    1    2   25   10
## [17,]  500    1    2   25   10
## [18,]  500    1    2   25   10
## [19,]  500    1    2   25   10
## [20,]  500    1    2   25   10
## [21,]  500    1    2   25   10
## [22,]  500    1    2   25   10
## [23,]  500    1    2   25   10
## [24,]  500    1    2   25   10
## [25,]  500    1    2   25   10
## [26,]  500    1    2   25   10
## [27,]  500    1    2   25   10
## [28,]  500    1    2   25   10
## [29,]  500    1    2   25   10
## [30,]  500    1    2   25   10
## [31,]  500    1    2   25   10
## [32,]  500    1    2   25   10
## [33,]  500    1    2   25   10
## [34,]  500    1    2   25   10
## [35,]  500    1    2   25   10
## [36,]  500    1    2   25   10
## [37,]  500    1    2   25   10
```

```
## [38,]  500    1    2   25   10
## [39,]  500    1    2   25   10
## [40,]  500    1    2   25   10
## [41,]  500    1    2   25   10
## [42,]  500    1    2   25   10
## [43,]  500    1    2   25   10
## [44,]  500    1    2   25   10
## [45,]  500    1    2   25   10
## [46,]  500    1    2   25   10
## [47,]  500    1    2   25   10
## [48,]  500    1    2   25   10
## [49,]  500    1    2   25   10
## [50,]  500    1    2   25   10
## [51,]  500    1    2   25   10
## [52,]  500    1    2   25   10
## [53,]  500    1    2   25   10
## [54,]  500    1    2   25   10
## [55,]  500    1    2   25   10
## [56,]  500    1    2   25   10
## [57,]  500    1    2   25   10
## [58,]  500    1    2   25   10
## [59,]  500    1    2   25   10
## [60,]  500    1    2   25   10
##
## $cost
##  [1]  2.975334 10.493632 13.636350 12.084368 14.615753 15.474318 16.768189
##  [8] 19.375550 18.389605 18.711438 17.872232 19.001801 19.026110 20.149552
## [15] 20.520467 19.901364 19.831708 19.397690 19.105595 18.639519 19.044895
## [22] 19.950703 19.744330 20.674581 20.430117 20.182204 19.807305 19.519953
## [29] 19.308072 19.152890 19.454501 19.514441 19.221936 18.953818 18.729321
## [36] 18.550459 18.306724 18.071574 18.100792 17.882214 17.700467 17.503834
## [43] 17.387693 17.751751 17.980414 17.817113 17.717565 17.644155 18.689371
## [50] 18.573513 18.393344 18.253484 18.173422 18.150433 18.271368 18.137543
## [57] 18.138594 18.211873 18.176565 18.170220
```

```r
new_pred_perf_1 <- function(init_params,
                            training_load,
                            obs_perf,
                            bounds_type = list("test"),
                            lambda) {
  # initializations
  curr_params <- init_params
  ####
  # curr_params take form c(p_0, k_1, k_2, tau_1, tau_2), to work with the time-invariant
  # functions
  ###
  matrix_params <- matrix(0, nrow = length(training_load), ncol = 5)
  colnames(matrix_params) <- c("p_0", "k_1", "k_2", "tau_1", "tau_2")

  days <- length(training_load)
  perf_out <- c(rep(curr_params[[1]], days + 1))
  T_1 <- 0
  T_2 <- 0
```

```
res_1 <- min_params_new(searh_params_mat(bounds_type = bounds_type),
                        training_load,
                        obs_perf,
                        init_params,
                        lambda)
opt_params_mat <- res_1$opt_params
cost_vec <- res_1$cost

# print(opt_params_mat) # for troubleshooting
# doing the algorithm

j = 1
for (i in 1:length(obs_perf)) {
  # update params if there is new information
  if (is.na(obs_perf[[i]]) == FALSE) {
    curr_params <- as.numeric(opt_params_mat[j, ])
    j <- j + 1
  } else {
  } # pass
  T_1 <- exp(-1 / curr_params[[4]]) * (T_1 + curr_params[[2]] * training_load[[i]])
  # training load index is  since i starts at 2
  T_2 <- exp(-1 / curr_params[[5]]) * (T_2 + curr_params[[3]] * training_load[[i]])
  perf_out[[i+1]] <- curr_params[[1]] +  T_1 - T_2
  ###############
  # fix cost it is wrong,
  #############
  matrix_params[i, ] <- as.numeric(curr_params)
}

#plotting
params_data <- tibble(
  "day" = 0:length(training_load),
  "k_1" = as.numeric(c(matrix_params[1, "k_1"], matrix_params[, "k_1"])),
  "k_2" = as.numeric(c(matrix_params[1, "k_2"], matrix_params[, "k_2"])),
  "tau_1" = as.numeric(c(matrix_params[1, "tau_1"], matrix_params[, "tau_1"])),
  "tau_2" = as.numeric(c(matrix_params[1, "tau_2"], matrix_params[, "tau_2"])),
  "p_0" = as.numeric(c(matrix_params[1, "p_0"], matrix_params[, "p_0"]))
)
k_plot <- ggplot(params_data, aes(x = day)) +
  geom_line(aes(y = k_1, color = "k_1")) +
  geom_line(aes(y = k_2, color = "k_2"))

tau_plot <- ggplot(params_data, aes(x = day)) +
  geom_line(aes(y = tau_1, color = "tau_1")) +
  geom_line(aes(y = tau_2, color = "tau_2"))

#outputs
out_list <- list()
out_list$performance <- perf_out
out_list$cost_vec <- cost_vec
out_list$perf_plot <- plot_perf(obs_perf, perf_out)
out_list$params_plots <- k_plot + tau_plot
return(out_list)
```

```
}
```

```
set.seed(1)
days_test <- 200
perf_sim_1 <- c(500, perf_tv(p_0 = 500,
          k_1 = 1,
          tau_1 = 25,
          k_2 = 2,
          tau_2 = 10,
          days = days_test,
          training_stim = list("constant", 100))$performance
)
noise_fn <- function(x, sd){
  x+rnorm(1, mean = 0, sd = sd)
}

perf_sim_1 = purrr::map_dbl(perf_sim_1, noise_fn, sd = 20)

# this is one off
samp_index <- c(sample(c(1:100), 50), sample(c(101:days_test), 20))
actual_perf <- rep(NA, days_test)
for (i in samp_index) {
  actual_perf[[i]] = perf_sim_1[[i+1]]

}

model_perf <- new_pred_perf_1(
  c(500, 1,2, 25, 10),
  c(rep(100, days_test)),
  actual_perf,
  lambda = 1
)
model_perf$perf_plot
model_perf$params_plots
```

The Important Function

```
new_pred_perf <- function(init_params,
                          training_load,
                          obs_perf,
                          bounds_type = list("test"),
                          lambda) {
  # initializations
  curr_params <- init_params
  curr_cost <- 0
  ####
  # curr_params take form c(p_0, k_1, k_2, tau_1, tau_2), to work with the time-invariant
  # functions
  ###

  matrix_params <- matrix(0, nrow = length(training_load), ncol = 5)
```

```r
colnames(matrix_params) <- c("p_0", "k_1", "k_2", "tau_1", "tau_2")

days <- length(training_load)
perf_out <- c(rep(curr_params[[1]], days + 1))
cost_vec <- c(rep(0, days + 1))
T_1 <- 0
T_2 <- 0

# doing the algorithm
for (i in 1:length(obs_perf )) {
  # update params if there is new information
  if (is.na(obs_perf[[i]]) == FALSE) {
    params_output <- update_params_opt(
      old_params = init_params,
      training_load = training_load,
      sub_obs_perf = obs_perf[1:i],
      # the subset up to observation
      bounds_type = bounds_type,
      lambda = lambda
    )
    curr_params <- params_output$opt_params
    curr_cost <- params_output$cost

  } else {
  } # pass
  T_1 <- exp(-1 / curr_params[[4]]) * (T_1 + curr_params[[2]] * training_load[[i]])
  # training load index is i-1 since i starts at 2
  T_2 <- exp(-1 / curr_params[[5]]) * (T_2 + curr_params[[3]] * training_load[[i]])
  perf_out[[i+1]] <- curr_params[[1]] +  T_1 - T_2
  cost_vec[[i]] <- curr_cost
  ###############
  # fix cost it is wrong,
  ##############
  matrix_params[i, ] <- as.numeric(curr_params)
}

#plotting
params_data <- tibble(
  "day" = 0:length(training_load),
  "k_1" = as.numeric(c(matrix_params[1, "k_1"], matrix_params[, "k_1"])),
  "k_2" = as.numeric(c(matrix_params[1, "k_2"], matrix_params[, "k_2"])),
  "tau_1" = as.numeric(c(matrix_params[1, "tau_1"], matrix_params[, "tau_1"])),
  "tau_2" = as.numeric(c(matrix_params[1, "tau_2"], matrix_params[, "tau_2"])),
  "p_0" = as.numeric(c(matrix_params[1, "p_0"], matrix_params[, "p_0"]))
)
k_plot <- ggplot(params_data, aes(x = day)) +
  geom_line(aes(y = k_1, color = "k_1")) +
  geom_line(aes(y = k_2, color = "k_2"))

tau_plot <- ggplot(params_data, aes(x = day)) +
  geom_line(aes(y = tau_1, color = "tau_1")) +
  geom_line(aes(y = tau_2, color = "tau_2"))
```

```
  #outputs
  out_list <- list()
  out_list$performance <- perf_out
  out_list$cost_vec <- cost_vec
  out_list$perf_plot <- plot_perf(obs_perf, perf_out)
  out_list$params_plots <- k_plot + tau_plot
  return(out_list)
}
```

Algorithm Pseudo-code, not complete

Let $z^{(t)} = (p_0^t, k_1^t, k_2^t, \tau_1^t, \tau_2^t)$ be the set of parameters on day t. For the sake of pseudo-code, set a cost function

$$C(\text{Error}(1,\ldots,i), w^{(t)}, w^{(t-1)}) = f(\text{Error}(1,\ldots,i)) + g(w^{(t)}, w^{(t-1)})$$

where $\text{Error}(1,\ldots,i)$ is the i dimensional vector that has the error for the 1 through ith performance days, f is some function that penalizes error, and the actual performance, like a scaler times SSE, MSE, or RMSE, and g is a function that penalizes the distance between the two parameter sets in some way. We wish to estimate the time varying parameters through Algorithm 1 .

Algorithm 1: Parameter estimation algorithm

Data: Training Load, a vector `w` of length n
Observed Performance, `obs perf` a vector of length n
Initial Parameters, a vector $z^{(0)}$
Settings for the Grid of values to search over, S
Coeficent term for the Limit Cost term, a $\lambda > 0$
Result: A vector of the predicted performance
for $i = 1,\ldots,n$ **do**
  if *Observed Performance at i is a number* **then**
    $G \leftarrow$ a grid of sets of parameters determined by some conditions
    $w^{(i)} = \text{argmin}_{w \in G} C(\text{Error}(1,\ldots,i-1), z, z^{(i-1)})$
    compute $P(i)$ with $w^{(i)}$
  else
    compute $P(i)$ with $w^{(i-1)}$
  end
end

Include here how exactly these algorithms work, this is still a work in progress.

Testing New Algorthm on simulated data

basic testing

As a sanity check, we give the algorithm the true values of the parameters as the initial parameters, tested aginst the the actual performance with no noise

```
set.seed(1)
days_test <- 200
perf_sim_1 <- c(500, perf_tv(p_0 = 500,
          k_1 = 1,
```

```
          tau_1 = 25,
          k_2 = 2,
          tau_2 = 10,
          days = days_test,
          training_stim = list("constant", 100))$performance
)
noise_fn <- function(x,
                     sd){
  x+rnorm(1, mean = 0, sd = sd)
}

# perf_sim_1 = purrr::map_dbl(perf_sim_1, noise_fn, sd = 20)

# this is one off
samp_index <- c(sample(c(1:100), 30), sample(c(101:days_test), 10))
actual_perf <- rep(NA, days_test)
for (i in samp_index) {
  actual_perf[[i]] = perf_sim_1[[i+1]]

}

model_perf <- new_pred_perf(
  c(500, 1,2, 25, 10),
  c(rep(100, days_test)),
  actual_perf,
  lambda = 10
)
model_perf$perf_plot
model_perf$params_plots
```

This checks out. It has passed the sanity check. The algorithm gives the same resutls no matter what the value of `lambda` is.

adding noise

Running the same simulation with gaussian noise to the simulated performance, with standard deviation of 20. We will see that increasing lambda smoothes out the performance curves. This first simulation is with `lambda = 0` which just finds the minimum parameters with respect to SSE.

```
set.seed(5)
perf_sim_1 = purrr::map_dbl(perf_sim_1, noise_fn, sd = 10)
samp_index <- c(sample(c(1:100), 50), sample(c(101:days_test), 10))
actual_perf <- rep(NA, days_test)
for (i in samp_index) {
  actual_perf[[i]] = perf_sim_1[[i+1]]
}
model_perf <- new_pred_perf(
  c(500, 1,2, 25, 10),
  c(rep(100, days_test)),
  actual_perf,
  lambda = .3
)
```

```
model_perf$perf_plot
model_perf$params_plots
```

This kind of wanders off

```
model_perf <- new_pred_perf(
  c(500, 1,2, 25, 10),
  c(rep(100, days_test)),
  actual_perf,
  lambda = .1
)
model_perf$perf_plot
model_perf$params_plots
```

The wandering is fixed when we introduce some penalty for the distance function

```
model_perf <- new_pred_perf(
  c(500, 1,2, 25, 10),
  c(rep(100, days_test)),
  actual_perf,
  lambda = .5
)
model_perf$perf_plot
model_perf$params_plots
```

More complicated simulation

Working on simulating a situation where the parameter change over time, but the simulated perforamance level is not realistic. Right now, the algorthm cannot handle this case.

```
set.seed(2)
days_test <- 500
perf_sim_2 <- c(500, perf_tv(p_0 = 500,
          k_1 = c(1,1,1.1),
          tau_1 = c(15, 20, 10),
          k_2 = c(2, 2, 2),
          tau_2 = c(5,7, 20),
          change_days = c(100, 200),
          days = days_test,
          training_stim = list("constant", 100))$performance
)
samp_index <- c(sample(c(1:100), 10), sample(c(101:days_test), 20))
actual_perf <- rep(NA, days_test)
for (i in samp_index) {
  actual_perf[[i]] = perf_sim_2[[i]]
}

plot_check <- plot_perf(actual_perf,
                        perf_sim_2) +
  labs(title = "Check that sampling from the simulation works")
plot_check
```

```
model_perf_2 <- new_pred_perf(
  c(500, 1,2, 25, 10),
  c(rep(100, days_test)),
  actual_perf,
  lambda = 10
)
model_perf_2$perf_plot
model_perf_2$params_plots
```

It does a pretty bad job. This is because the algorithm, at every new data point, tries to fit the best Time-*Invariant* curve. In this situation, all of the Time-Invariant functions don't fit well when the parameters first change.

Applying to Real Data

This is a work in progress. Right now, the algorithm cannot handle this case This runs the

```
real_data <- data_2
day_vec_real <- real_data[[1]]
training_load_real <- real_data[[2]]
obs_perf_real <- real_data[[3]]
init_params_real <- c(262, .1, .1, 11, 11)
obs_new_alg <- new_pred_perf(
  init_params_real,
  training_load_real,
  obs_perf_real,
  lambda = 1
)

obs_new_alg$perf_plot
obs_new_alg$params_plots
```

Here, it does not change because the algorithm has decided that all of the options that we have told it to search over are all bad so it stuck with the initial estimate which is probably a good sign.

```
min_params <- function(params_matrix,
                       training_load,
                       obs_perf,
                       old_params,
                       lambda) {
  min_params <- old_params
  obs_indexes <- which(!is.na(obs_perf))
  n <- length(obs_indexes)
  min_cost <- sqrt(SSE(old_params, training_load, obs_perf)/n)
  for (i in 1:nrow(params_matrix)){
    params_i <- as.numeric(params_matrix[i, ])
    params_dist_i <- params_dist(params_i, old_params)
    SSE_i <- 0; cost_i <- 0
    p_0 <- params_i[[1]]
    k_1 <- params_i[[2]]; k_2 <- params_i[[3]]
    coef_1 <- exp(-1/params_i[[4]]); coef_2 <- exp(-1/params_i[[5]])
```

```
    T_1 <- 0; T_2 <- 0
    lower_index <- 1
    j <- 0
    for (new_index in obs_indexes) {
      for (t in lower_index:new_index) {
        T_1 <- coef_1*(T_1 + k_1*training_load[[t]])
        T_2 <- coef_2*(T_2 + k_2*training_load[[t]])
      }
      SSE_i <- SSE_i + (p_0 + T_1 - T_2 - obs_perf[[new_index]])^2
      j <- j + 1
      lower_index <- new_index + 1
      if(min_cost < sqrt(SSE_i/n) + lambda*params_dist_i) {
        break # goes to next set of parameters
        # for a lot of the sets of parameters,
        # the condition should trigger on the first few data points
      }
      if (j == n && min_cost > sqrt(SSE_i/n) + lambda*params_dist_i) {
        # the second condition helps with stability, if two different sets
        # of parameters have the same cost -- namely when they both have 0 cost --
        # we will pick our previous point
        min_cost <- sqrt(SSE_i/n) + lambda*params_dist_i
        min_params <- params_i

      }
      else {
        # do nothing; go to the next loop
      }
    }
  }
  return(list("cost" = min_cost, "opt_params" = min_params))
}

# this function creates (or calls, we will see) the parameter matrix and applies
# it to the previous function. makes things tider in the algorithm funciton
update_params_opt <- function(old_params,
                          training_load,
                          sub_obs_perf,
                          bounds_type = list("test"),
                          lambda
                          ) {
  if(bounds_type == "test") {
    params_matrix <- expand.grid(p_0 = 500,
                                 k_1 = c(1,2,3,4),
                                 k_2 = c(2,4,6,8),
                                 tau_1 = c(5:35),
                                 tau_2 = c(5:35))
  }
  params_matrix <- params_matrix[params_matrix$tau_1 <= params_matrix$tau_2,]

  getting_params_ouput <- min_params(params_matrix,
                                 training_load,
                                 sub_obs_perf,
                                 old_params,
```

```
                                    lambda = lambda)

  out_list <- list()
  out_list$opt_params <- getting_params_ouput$opt_params
  out_list$cost <- getting_params_ouput$cost
  return(out_list)
}
```