

# PS Bootstrap Binary Standardization

Hun & Amy

2/9/2022

The goal of this document is to combine a lot of the functions that Hun has made and made them into functions that can easily be called for the main loop of the simulations function.

## Function to find the intercept to attain the desired treatment rate

```
treatment_rate_coef <- function(desired_prop, seq_try, df, logit_formula) {  
  # setting up data setting variables  
  seq_select = tibble(seq_try, mean_A = rep(0, length(seq_try)))  
  
  # for loop that goes through all intercept options.  
  for(i in c(1:length(seq_try))) {  
    beta_0_treat = seq_try[i]  
  
    # this will be where the beta is updated. Must have beta_0_treat in formula  
    A_logit <- eval(parse(text=logit_formula))  
    p_A      <- exp(A_logit)/(1+ exp(A_logit))  
    A        <- rbinom(N, 1, p_A) # this decides treat and and non treated  
  
    # getting the mean treated  
    seq_select$mean_A[i] = mean(A)  
  }  
  
  # # plotting visualizing check  
  # seq_select %>%  
  #   ggplot(aes(x=seq_try, y=mean_A))+  
  #   geom_point()  
  
  # selecting coeff that gives desired treated untreated proportion  
  if (sum(seq_select$mean_A == desired_prop) > 0) {  
    beta_0_treat = seq_select %>% filter(mean_A == desired_prop ) %>% pull(seq_try)  
  } else {  
    beta_0_treat = seq_select %>%  
      filter(mean_A < desired_prop + 0.01) %>%  
      filter(mean_A > desired_prop - 0.01) %>%  
      summarise(  
        avg = mean(seq_try)  
      ) %>% pull(avg)  
  }  
  
  # returning the beta value  
  return(mean(beta_0_treat))  
}
```

```
}
```

## Finding the right beta\_effect estimate for binary data

```
binary_outcome_rate <- function(desired_diff, seq_try, df, logit_formula ) {
  seq_select = tibble(seq_try, prop_save = rep(0, length(seq_try)))
  for(i in c(1:length(seq_try))) {
    beta_effect = seq_try[i]

    # binary outcome
    y_logit <-eval(parse(text=logit_formula ))
    p_outcome <- exp(y_logit)/(1+ exp(y_logit))
    y_binary <- rbinom(N, 1, p_outcome)

    # finding difference between groups
    prop = tibble(y_binary, A = df$A) %>%
      group_by(A) %>%
      summarise( mean_out = mean(y_binary)) %>%
      pivot_wider(
        names_from = A,
        values_from = mean_out
      ) %>%
      mutate (diff = `1`-`0`) %>%
      pull(diff)

    # having the difference
    seq_select$prop_save[i] = prop
  }

  # graphing
  # seq_select %>%
  #   ggplot(aes(x=seq_try, y=prop_save))+
  #   geom_point()

  # saving the best beta.
  beta_effect = seq_select %>%
    filter(prop_save < desired_diff + 0.01) %>%
    filter(prop_save > desired_diff - 0.01) %>%
    summarise(
      avg = mean(seq_try)
    ) %>% pull(avg)
  return( beta_effect)
}

# logit_formula <- "beta_effect*df$A + 0.5*df$L2 + 0.3*df$L3"
# seq_try = seq(-4,4, 0.01)
# desired_diff = 0.4
# beta_effect = binary_outcome_rate(desired_diff, seq_try, df, logit_formula)
```

## Generating data

```
data_gen <- function(N, desired_prop) {

  # Setting the Observed Variables
  L1 <- rnorm(N, 0, 1)
  L2 <- rnorm(N, 0, 1)
  L3 <- rnorm(N, 0, 1)

  # adding the covaraites to dataset
  df <- tibble(L1, L2, L3)

  #desired_prop = 0.2 # TM - is this correct as a constant or should it be a variable?
  seq_try = seq(-2,2, 0.01)
  logit_formula <- "beta_0_treat+ 0.5*df$L1 + 0.4*df$L2"

  # get the desited treatment rate
  beta_0_treat = treatment_rate_coef(desired_prop, seq_try, df, logit_formula)

  # Creating the treatment assigment
  A_logit <- beta_0_treat+ 0.5*df$L1 + 0.4*df$L2
  p_A <- exp(A_logit)/(1+ exp(A_logit))
  A <- rbinom(N, 1, p_A)

  # adding the treated untreated data to the dataset
  df$A = A

  # continuous outcome
  y_continuous <- 1*df$A + 0.5*df$L2 + 0.3*df$L3 + rnorm(N, 0,1)

  df$y_continuous = y_continuous

  # binary outcome
  logit_formula <- "beta_effect*df$A + 0.5*df$L2 + 0.3*df$L3"
  seq_try = seq(-4,4, 0.01)
  desired_diff = 0.4
  beta_effect = binary_outcome_rate(desired_diff, seq_try, df, logit_formula)

  y_logit <-eval(parse(text=logit_formula ))
  p_outcome <- exp(y_logit)/(1+ exp(y_logit))
  y_binary <- rbinom(N, 1, p_outcome)

  df$y_binary = y_binary

  # propensity scores
  exposureModel <- glm(A ~ L1 + L2 + L3, data = df, family = "binomial")
  #Note that Pr[A=0|L] = 1-Pr[A=1|L]

  #getting estimated propensity score
  df$ps <- predict(exposureModel, df, type = "response")

  return(df)
```

```

}

# calling the data gen function
N = 500
desired_prop = 0.2
df = data_gen(N, desired_prop)

```

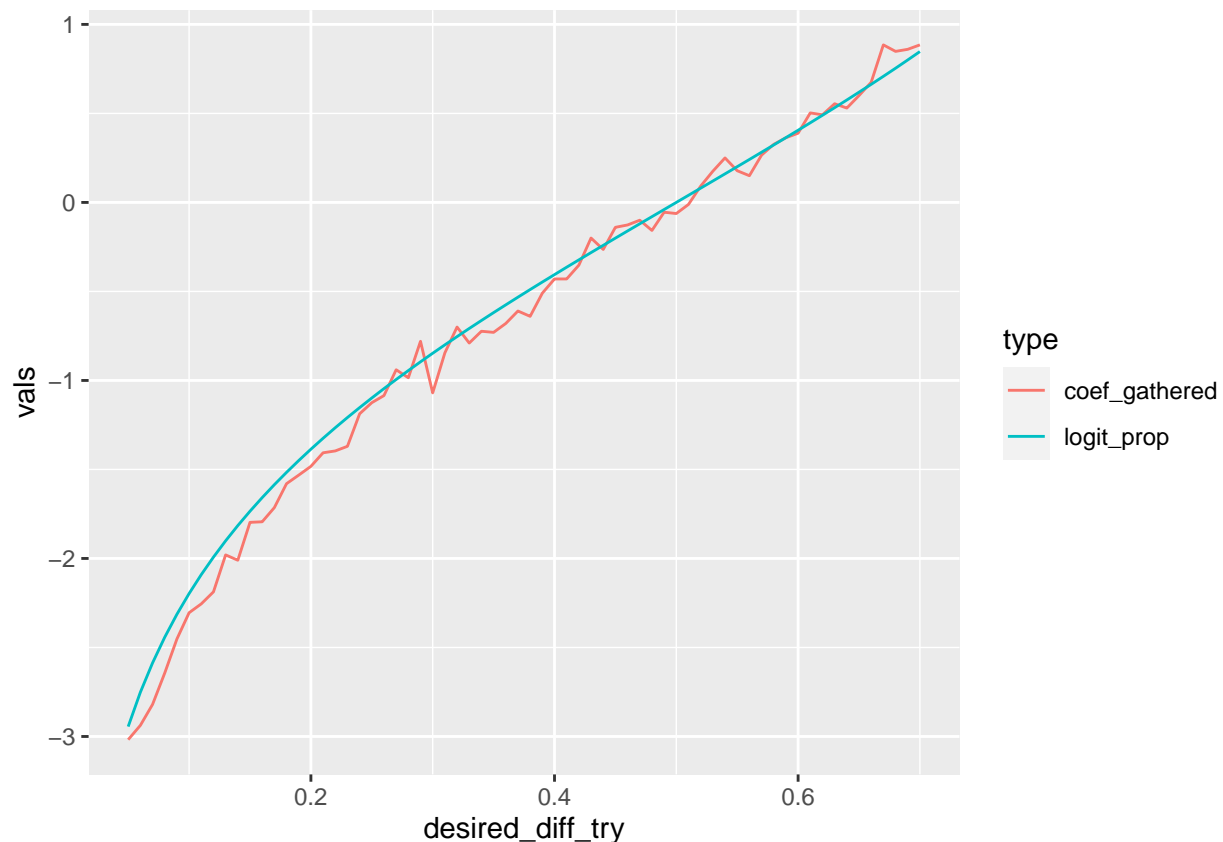
This is in response to Waveley's notion that  $\text{logit}(\text{desired\_prop}) = \alpha_0$

```

seq_try      = seq(-6,6, 0.01)
logit_formula <- "beta_0_treat+ 0.5*df$L1 + 0.4*df$L2"
desired_diff_try <- seq(0.05,0.7, 0.01)
coef_gathered <- map_dbl(1:length(desired_diff_try), function(i) treatment_rate_coef(desired_diff_try[i], seq_try))

# plotting the gathered values from my function against the logit.
tibble(desired_diff_try, coef_gathered,
       logit_prop = log(desired_diff_try / (1-desired_diff_try))) %>%
  pivot_longer(
    cols = c(coef_gathered, logit_prop),
    names_to = "type",
    values_to = "vals"
  ) %>% ggplot(aes(x=desired_diff_try, y=vals, color = type)) +
  geom_line()

```



## Nearest neighbor propensity score matching

```
matched <- matchit(A ~ L1 + L2 + L3, data = df,
  distance = "glm", link = "logit",
  method = "nearest", ratio = 1)

matched_df <- match.data(matched)
```

## simple bootstrap binary

```
nboot <- 100
simple_bootstrap_binary <- function(nboot, matched_df) {

  # set up a matrix to store results
  boots <- data.frame(i = 1:nboot,
    se_ATE = NA,
    se_OR = NA,
    log_OR = NA,
    mean1 = NA,
    mean0 = NA,
    difference = NA
  )

  # loop to perform the bootstrapping
  for (i in 1:nboot) {
    # sample with replacement
    sampl <- matched_df %>% filter(subclass %in% sample(levels(subclass), 500, replace = TRUE))

    bootmod <- glm(y_binary ~ A + ps, data = sampl,
      weights = weights, family = binomial)

    # create new data sets
    sampl.treated <- sampl %>% mutate(A = 1)
    sampl.untreated <- sampl %>% mutate(A = 0)

    # predict values
    sampl.treated$pred.y <- predict(bootmod, sampl.treated, type = "response")
    sampl.untreated$pred.y <- predict(bootmod, sampl.untreated, type = "response")

    # output results
    boots[i, "log_OR"] <- summary(bootmod)$coeff[2,1]
    boots[i, "se_OR"] <- summary(bootmod)$coeff[2,2]
    boots[i, "se_ATE"] <-
      sqrt((summary(bootmod)$coeff[2,2]*mean(sampl.treated$pred.y) *
        (1 - mean(sampl.treated$pred.y)))^2 +
        (summary(bootmod)$coeff[2,2]*mean(sampl.untreated$pred.y) *
        (1 - mean(sampl.untreated$pred.y)))^2)

    boots[i, "mean1"] <- mean(sampl.treated$pred.y)
    boots[i, "mean0"] <- mean(sampl.untreated$pred.y)
    boots[i, "difference"] <- boots[i, "mean1"] - boots[i, "mean0"]
  }
}
```

```

}

mean_log_OR      <- mean(boots$log_OR)
Empirical_se_ATE <- sd(boots$difference)
mean_se_ATE      <- mean(boots$se_ATE)
Empirical_se_log_OR <- sd(boots$log_OR)
mean_se_log_OR   <- mean(boots$se_OR)
ATE              <- mean(boots$difference)
lower_CI_ATE     <- ATE - 1.96*Empirical_se_ATE
upper_CI_ATE     <- ATE + 1.96*Empirical_se_ATE
lower_CI_log_OR  <- mean_log_OR - 1.96*mean_se_log_OR
upper_CI_log_OR  <- mean_log_OR + 1.96*mean_se_log_OR

return(tibble(mean_log_OR, Empirical_se_ATE, mean_se_ATE,
               Empirical_se_log_OR, mean_se_log_OR, ATE,
               lower_CI_ATE, upper_CI_ATE, lower_CI_log_OR,
               upper_CI_log_OR))
}

simple_bootstrap_binary(100, matched_df)

```

```

## # A tibble: 1 x 10
##   mean_log_OR Empirical_se_ATE mean_se_ATE Empirical_se_lo~ mean_se_log_OR ATE
##   <dbl>         <dbl>         <dbl>         <dbl>         <dbl> <dbl>
## 1      1.79      0.00420      0.101      0.0296      0.408 0.284
## # ... with 4 more variables: lower_CI_ATE <dbl>, upper_CI_ATE <dbl>,
## #   lower_CI_log_OR <dbl>, upper_CI_log_OR <dbl>

```

## simple bootstrap continous

```

nboot <- 100
simple_bootstrap_continuous <- function(nboot, matched_df) {
  # set up a matrix to store results
  boots <- data.frame(i = 1:nboot,
                      se = NA,
                      mean1 = NA,
                      mean0 = NA,
                      beta1 = NA,
                      ATE = NA
                      )
  # loop to perform the bootstrapping

  for(i in 1:nboot) {
    # sample with replacement
    sampl <- matched_df %>% filter(subclass %in% sample(levels(subclass),500, replace = TRUE))

    bootmod <- glm(y_continuous ~ A + ps, data = sampl,
                  weights = weights)

    # create new data sets
    sampl.treated <- sampl %>%

```

```

    mutate(A = 1)

    sampl.untreated <- sampl %>%
      mutate(A = 0)

    # predict values
    sampl.treated$pred.y <-
      predict(bootmod, sampl.treated)

    sampl.untreated$pred.y <-
      predict(bootmod, sampl.untreated)

    # output results
    boots[i, "beta1"] <- summary(bootmod)$coeff[2,1]
    boots[i, "se"] <- summary(bootmod)$coeff[2,2]
    boots[i, "mean1"] <- mean(sampl.treated$pred.y)
    boots[i, "mean0"] <- mean(sampl.untreated$pred.y)
    boots[i, "ATE"] <- boots[i, "mean1"] - boots[i, "mean0"]
  }

  Empirical_sd <- sd(boots$ATE)
  ATE <- mean(boots$ATE)
  mean_se <- mean(boots$se)
  lower_CI_ATE <- ATE - 1.96*mean_se
  upper_CI_ATE <- ATE + 1.96*mean_se
  ATE_from_beta <- mean(boots$beta1)

  return(tibble(Empirical_sd, ATE, mean_se, lower_CI_ATE,
                upper_CI_ATE, ATE_from_beta))
}

simple_bootstrap_continuous(nboot, matched_df)

## # A tibble: 1 x 6
##   Empirical_sd  ATE mean_se lower_CI_ATE upper_CI_ATE ATE_from_beta
##         <dbl> <dbl>   <dbl>         <dbl>         <dbl>         <dbl>
## 1      0.0129  1.06   0.158         0.749         1.37         1.06

```