# Compare distributions of mean via different imputation methods on artifical data

library(keras)

library(dplyr)

N = 10^6

n = 10^4

it <- 50

p\_y\_ep\_control <- 2

rescale <- function(vec) { # rescale to 0,1

(vec - min(vec)) / (max(vec) - min(vec))

}

create\_split <- function(df, y) {

smp\_size <- floor(.75\*nrow(df))

train\_ind <- sample(seq\_len(nrow(df)), size = smp\_size)

x\_train <<- df[train\_ind, ]

x\_test <<- df[-train\_ind, ]

y\_train <<- y[train\_ind]

y\_test <<- y[-train\_ind]

}

create\_validation\_split <- function(x\_train, y\_train) {

# Validation Set

val\_indices <- 1:1000

x\_val <<- x\_train[val\_indices,]

partial\_x\_train <<- x\_train[-val\_indices,]

y\_val <<- y\_train[val\_indices]

partial\_y\_train <<- y\_train[-val\_indices]

}

p\_1 <- rnorm(N, mean = 30, sd = 2.5)

p\_2 <- rnorm(N, mean = 15, sd = 2)

p\_3 <- rnorm(N, mean = 5, sd = 1)

p\_y\_ep <- rnorm(N, mean = 0, sd = p\_y\_ep\_control)

p\_y <- sqrt(p\_1\*p\_2) + p\_3 + p\_y\_ep

p\_pi\_ep <- rnorm(N, mean = 0, sd = 2)

temp\_pi <- sqrt(p\_y) + p\_pi\_ep

temp\_pi <- rescale(temp\_pi)

p\_pi <- temp\_pi \* (n / sum(temp\_pi))

p\_df <- cbind(p\_1, p\_2, p\_3, p\_y, p\_pi)

p\_tbl <- as\_tibble(p\_df)

statistic\_tracker <- data.frame(true\_mean = numeric(it),

oracle\_mean = numeric(it),

naive\_mean = numeric(it),

pi\_naive\_mean = numeric(it),

lin\_imp\_mean = numeric(it),

nn\_imp\_mean = numeric(it))

# Want to estimate:

mu\_y <- (1 / N) \* sum(p\_tbl$p\_y)

# monte carlo simulation of draws from population with comparison methods

for (i in 1:it) {

sample\_population\_by\_pi <- sample\_n(tbl = p\_tbl, size = n, replace = FALSE,

weight = p\_pi)

df <- sample\_population\_by\_pi %>% rename(x\_1 = p\_1, #Since we are not dealing with p\_ anymore

x\_2 = p\_2,

x\_3 = p\_3,

pi = p\_pi,

y = p\_y)

# verify sum of 1/pi over sample df = N

sum(1/df$pi) #= 997824.5, approx 10^6

# Verify sum of pi over df approx n

sum(df$pi) #= 104, approx 100. not correct.

# Can compute some of the statistics before dropping for ease

mean(df$y) #= 26.258

# Drop some labels - weighted to high y

indices <- sample(1:nrow(df), .175\*nrow(df), prob = df$y) #weights are stdized to 1 by sample()

dropped\_obs <- df[indices,]

reduced\_df <- df[-indices,]

#mean of complete cases

statistic\_tracker$naive\_mean[i] <- mean(reduced\_df$y)

# pi-corrected naive mean

hat\_N <- sum(1 / (reduced\_df$pi))

statistic\_tracker$pi\_naive\_mean[i] <- (1 / hat\_N)\*sum((1 / reduced\_df$pi) \*

reduced\_df$y)

# Oracle mean, \bar y ^o

statistic\_tracker$oracle\_mean[i] <- (1 / N) \* sum((1 / reduced\_df$pi) \*

reduced\_df$y)

# mean according to imputed data via linear regression

linear\_model <- lm(y ~ x\_1 + x\_2 + x\_3, data = reduced\_df)

lm\_y\_hat <- predict(linear\_model, dropped\_obs)

statistic\_tracker$lin\_imp\_mean[i] <- (1 / N)\*(sum(reduced\_df$y / reduced\_df$pi)

+ sum(lm\_y\_hat / dropped\_obs$pi))

# Mean accoridng to imputed data via neural network (3 diff methods here,

# can start with naive NN for baseline)

y\_train <- reduced\_df$y

reduced\_df\_nolab <- select(reduced\_df, -c(pi, y))

y\_test <- dropped\_obs$y

dropped\_obs\_nolab <- select(dropped\_obs, -c(pi,y))

reduced\_df\_nolab <- as.matrix(reduced\_df\_nolab)

dropped\_obs\_nolab <- as.matrix(dropped\_obs\_nolab)

x\_train <- reduced\_df\_nolab

x\_test <- dropped\_obs\_nolab

create\_validation\_split(x\_train, y\_train)

model <- keras\_model\_sequential() %>%

layer\_dense(units = 6, activation = "relu",

input\_shape = dim(x\_train)[[2]]) %>%

layer\_dense(units = 6, activation = "relu") %>%

layer\_dense(units = 1)

model %>% compile(

optimizer = "rmsprop",

loss = "mse",

metrics = c("mae")

)

history <- model %>% fit(

partial\_x\_train,

partial\_y\_train,

epochs = 15,

batch\_size = 512,

validation\_data = list(x\_val, y\_val)

)

x\_test <- as.matrix(x\_test)

nn\_y\_hat <- predict(model, x\_test)

mean(nn\_y\_hat)

statistic\_tracker$nn\_imp\_mean[i] <- (1 / N)\*(sum(reduced\_df$y / reduced\_df$pi)

+ sum(nn\_y\_hat / dropped\_obs$pi))

# Mean accoridng to imputed data via neural network w weighted resample

print(i)

}

# MSE of these distributions of means compared to true

(1 / it) \* sum((bar\_y - mu\_y)^2) # where bar\_y is the vector of predicted mean

# MAE of these distributions of means compared to true

(1 / it) \* sum(bar\_y - mu\_y) # where bar\_y is the vector of predicted mean

# Plot distributions of means