Simulation Study: Example 1

Ashley I Naimi

Spring 2024

Contents

1 Evaluating the Performance of IPW versus Marginal Standardization

We are going to wrap up our short course by pulling several of the pieces together that we've covered over the last few days. We'll use the ADEMP framework to develop a simulation study we'll conduct here.

We'll start with the aims of our simulation study: to compare the performance of inverse probability weighting versus marginally standardized estimates of the average treatment effect. However, it would help us to get more specific with our intentions. For instance, we might ask whether we should choose IP-weighting versus marginal standardization with a different number of confounding variables (e.g., 10 versus 25. We might as whether we should choose the bootstrap versus the robust variance estimator when we use IP weighting. Getting more specific with our aims will help us make decisions about how we should construct our data generating mechanisms, analyses, and performance measures.

2 Aims

To start, let's specify some aims of our simulation study. These will help guide how we write the code we need to execute the tasks to fulfill our aims:

- Compare the performance of IP weighting versus marginal standardization when the number of confounding variables is 10 versus 25
- Evaluate the performance of the robust variance estimator versus the bootstrap for the IP weighted estimator

We'll begin with these aims, but as we write our functions, it will be useful to keep in mind other variables that we may be interested in exploring. For example: we may want to look at what happens when the correlation between confounding variables changes; we may want to look at the impact of sample size; etc. We can easily incorporate these as arguments into our simulation function, even though we may decide to not explore them for the sake of, e.g., run time.

3 Data Generating Mechanisms

In terms of generating our data, we will rely on a simple triangle DAG, with a set of confounding variables, a single exposure, and a single outcome:

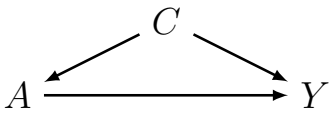


Figure 1: Simple confounding triangle, with exposure A, confounder C, and outcome Y.

This gives us a relatively straightforward causal ordering, as follows:

- The variable C is most exogenous to this system
- The variable A is second most exogenous
- The variable Y is the most endogenous

Importantly, even though we may have a large number of variables C, because they are all exogenous and independent, we can treat them as a single node on the DAG, and simulate them simultaneously (and independently) in the code we write.

4 Estimands

We will focus our attention on the marginally adjusted odds ratio among the exposed versus unexposed:

$$\frac{odds(Y = 1 \mid X = 1)}{odds(Y = 1 \mid X = 0)}$$

where $odds(\bullet) = P(\bullet)/[1 - P(\bullet)]$. Note that, for a binary outcome, we could have also expressed interest in the marginally adjusted risk ratio, or the marginally adjusted risk difference, or some combination of all three. Since the code we will write for this section is already complex enough, we will forego targeting these estimands, but will note when and what changes can be made as we proceed with this example.

5 Methods

5.1 Inverse Probability Weighting

We're interested in comparing IP weighting to marginal standardization. To implement IP weighting, we need to first construct stabilized inverse probability weights, defined as:

$$sw = \begin{cases} \dfrac{P(X=1)}{P(X=1 \mid C)} & \text{if } X = 1 \\ \dfrac{P(X=0)}{P(X=0 \mid C)} & \text{if } X = 0 \end{cases}$$

In this above equation, the $P(X = 1)$ used in the numerator of the top equation can be obtained by simply taking the mean of X when $X \in [0, 1]$. Similarly, the $P(X = 0)$ can be obtained by taking one minus the mean of X.

The denominator of this equation can be obtained using, e.g., predictions from a logistic regression model that regresses the exposure against all confounders. We can fit a model like this in R using code that looks something like:

```
p_model <- glm(x ~ c1 + c2 + c3 + c4 + c5 + c6 + c7 + c8 + c9 +
    c10, data = the_data, family = binomial(link = "logit"))
```

From the above model fit, we can extract the fitted values with:

```
p_model$fitted.values
```

These fitted values will correspond to $P(X = 1 \mid C)$, which we can use directly in our equation to obtain the stabilized weights. To obtain an estimate of $P(X = 0 \mid C)$, we can simply take $1 - P(X = 1 \mid C)$, or:

```
1 - p_model$fitted.values
```

One problem with the above code is that, for a large number of C variables, we will have to write out each in the regression model which can complicate the evaluation of our methods when we change the number of confounding variables. To deal with this problem, we can use a coding trick in R that looks like this:

```
p_model <- glm(x ~ ., data = the_data, family = binomial(link = "logit"))
```

What the code in the above model does is it regresses the `x` variable against everything else in the dataset named `the_data`. This shorthand notation makes it easy to flexibly change what goes into a regression model. However, *we must ensure* that all the other variables in the dataset named `the_data` are those that we want in our model. If this is not the case, we will end up fitting a model that we do not want.

Inverse probability weighting can perform very poorly when the numerical value of the propensity score is very high or very low. For example, for an exposed individual, if we have an overall $P(X = 1) = 0.5$ (the numerator in our weight equation), and a propensity score of 0.00001, this would return a weight of:

$$0.5/0.00001 = 50000$$

This means that the individual with a propensity score of 0.00001 will contribute a total of 50,000 pseudo-individuals in our weighted analysis. If our original sample size was only 1,000, this would suggest an important problem: in the weighted analysis, a single individual is contributing 50,000 observations, which is 50 times the size of the actual data we have.

To address problems like this, we can trim the weights we create using the equation above. Specifically, we can construct a trimmed stabilized IP weight, where any values of the stabilized weights greater than (e.g.) the 97.5th percentile are **reset to the 97.5th percentile value**. For example, using an `ifelse` statement in R, we can construct trimmed weights as:

```
sw_trim <- ifelse(sw > quantile(sw, 0.975), quantile(sw, 0.975),
    analysis_data$sw)
```

5.2 Marginal Standardization

To implement marginal standardization with a large number of confounding variables, we can use the same coding trick above to fit an outcome regression model as follows:

```
mu_model <- glm(y ~ ., data = the_data, family = binomial(link = "logit"))
```

For this modeling approach to work, we must ensure that the dataset used to fit this model (`the_data`) has **only** the outcome, the exposure, and the p confounders we'd like to include in our model. All other variables, including ID variables, the estimated propensity score, the stabilized weights we might construct, and anything else, is excluded from the dataset used to fit this model.

Once estimated, we can then generate predictions from this `mu_model` using the predict function, and averaging the predictions we obtained so that we can get our estimand:

```
muhat1 <- mean(predict(mu_model, newdata = transform(the_data,
    x = 1), type = "response"))
```

Note the use of the `transform` function here, which will set the exposure for each individual in our dataset to 1. Additionally, we've included as an argument to this predict function `type="response"`. When fitting a logistic regression model such as the one above, we can generate predictions on different scales. For instance, we could use `mu_model` to predict the log-odds of the outcome we'd observe if everyone was exposed. If we wanted predictions on the log-odds scale, we could use `type = "link"` as an argument to the predict function. However, using `type="response"` provides us with predictions on the probability scale.

6 Performance Measures

Once we obtain our simulation data, we will use the following performance measures to compare IP weighting to marginal standardization:

- bias of the point estimate
- bias of the standard error
- confidence interval coverage
- confidence interval length

7 Monte Carlo Integration for the True Marginal OR

In order to obtain estimates of our desired performance measures, we will need to compute the true marginal odds ratios for all the scenarios we will explore. In our case, we will limit our simulation to scenarios where the sample size for each Monte Carlo simulation is 500 and 1000, and where the number of confounders in each Monte Carlo simulation is 10 and 25.

This means that we'll have to use Monte Carlo integration to estimate the true marginal odds ratios under *two* settings: when there are 10 and 25 confounders in the data generating mechanism.[1] We can do this using the following code:

[1] Note that we don't need to account for changing sample sizes in the Monte Carlo integration approach, since we are interested in maximizing the sample size for this step.

```
expit<-function(a){1/(1+exp(-a))}


set.seed(123)
simulation_function_true <- function(intercept = -2,
                                     exposure = 1,
                                     c_number = 10,
                                     cov_mat = 0,
                                     diag_cov = 1){
    n = 5e6

    # how many confounders?
    p <- c_number

    ## confounder matrix
    sigma <- matrix(cov_mat, nrow=p, ncol=p)
    diag(sigma) <- diag_cov
    c <- mvtnorm::rmvnorm(n, mean=rep(0,p), sigma=sigma)

    # DESIGN MATRIX FOR THE PROPENSITY SCORE MODEL
    piMat <- model.matrix(
      as.formula(
        paste("~(",
              paste("c[,",1:ncol(c),"]", collapse="+"),
```

```
                ")"
          )
        )
    )[,-1]

    # parameters for confounder exposure relation
    parmsC_pi <- rep(log(1.5), c_number)

    # simulate the exposure
    x <- exposure #rbinom(n, size = 1, expit(-.5 + piMat%*%parmsC_pi))

    # parameters for the confounder outcome relation
    parmsC_mu <- rep(log(2), c_number)

    # simulate the outcome
    pY <- mean(expit(intercept + log(2)*x + piMat%*%parmsC_mu))

    print(paste("the mean of y is: ", pY))

    ## what would we change here if interested in marginally adjusted risk difference or risk ratio?
    res <- pY/(1 - pY)

    return(res)
}

odds1 <- simulation_function_true(intercept = -2, exposure = 1, c_number = 10)
odds0 <- simulation_function_true(intercept = -2, exposure = 0, c_number = 10)

or_marg1 <- log(odds1/odds0)

or_marg1

odds1 <- simulation_function_true(intercept = -2, exposure = 1, c_number = 25)
odds0 <- simulation_function_true(intercept = -2, exposure = 0, c_number = 25)
```

```
or_marg2 <- log(odds1/odds0)

or_marg2

true_log_or1 <- or_marg1
true_log_or2 <- or_marg2

# the true log OR marg1 (c_number = 10) is: 0.4132932
#
# the true log OR marg2 (c_number = 25) is: 0.289803
```

8 Running the Simulation

Now that we have our true values, we can go ahead and proceed with running the simulation.

```
library(parallel)
library(lmtest)
library(sandwich)

expit<-function(a){1/(1+exp(-a))}

set.seed(123)

simulation_function <- function(index,
                                intercept=-2,
                                sample_size = 1000,
                                c_number = 10,
                                cov_mat = 0,
                                diag_cov = 1){

    # printing to console won't work with parallel processing
    print(index)
```

```r
# DATA GENERATION
n <- sample_size
print(n)

# how many confounders to simulate?
p <- c_number
print(p)

## confounder matrix
sigma <- matrix(cov_mat, nrow=p, ncol=p)
diag(sigma) <- diag_cov
c <- mvtnorm::rmvnorm(n, mean=rep(0,p), sigma=sigma)

# DESIGN MATRIX FOR THE PROPENSITY SCORE MODEL
piMat <- model.matrix(
  as.formula(
    paste("~(",
          paste("c[,",1:ncol(c),"]", collapse="+"),
          ")"
          )
    )
  )[,-1]

# parameters for confounder exposure relation
parmsC_pi <- rep(log(1.5), c_number)

# simulate the exposure
x <- rbinom(n, size = 1, expit(-.5 + piMat%*%parmsC_pi))

# parameters for the confounder outcome relation
parmsC_mu <- rep(log(2), c_number)

# simulate the outcome
y <- rbinom(n, size = 1, expit(intercept + log(2)*x + piMat%*%parmsC_mu))
```

```r
# construct dataset
analysis_data <- data.frame(y, x, c)

# ANALYSIS

# marginal standardization
meanY <- mean(analysis_data$y)

m1 <- glm(y ~ ., data = analysis_data, family=binomial(link="logit"))

muhat1 <- mean(predict(m1, newdata = transform(analysis_data, x = 1), type="response"))
muhat0 <- mean(predict(m1, newdata = transform(analysis_data, x = 0), type="response"))

## compute the odds from these average probabilities
odds1<-muhat1/(1-muhat1)
odds0<-muhat0/(1-muhat0)

## marginal log-odds ratio
marginal_standardization_estimate <- log(odds1/odds0)

# IP WEIGHTING
# create the propensity score in the dataset
analysis_data$propensity_score <- glm(x ~ .,
                                      data = analysis_data[,-y],
                                      family = binomial("logit"))$fitted.values

# stabilized inverse probability weights
analysis_data$sw <- (mean(analysis_data$x)/analysis_data$propensity_score)*analysis_data$x +
  ((1-mean(analysis_data$x))/(1-analysis_data$propensity_score))*(1-analysis_data$x)

summary(analysis_data$sw)
quantile(analysis_data$sw, .995)

# trim the weights
```

```r
analysis_data$sw <- ifelse(analysis_data$sw>quantile(analysis_data$sw, .995),
                           quantile(analysis_data$sw, .995),
                           analysis_data$sw)

# outcome model
m2 <- glm(y ~ x, data = analysis_data, weights = sw, family = quasibinomial(link="logit"))

ip_weighting_estimate <- coeftest(m2,
                                  vcov. = vcovHC(m2, type = "HC3"))[2,1:2]

# bootstrap SEs
boot_func <- function(nboot, data){

  a_dat <- data

  index <- sample(1:nrow(a_dat), nrow(a_dat), replace = T)

  boot_dat <- a_dat[index, ]

  # MARGINAL STANDARDIZATION
  m1_ <- glm(y ~ ., data = boot_dat, family=binomial(link="logit"))

  muhat1_ <- mean(predict(m1_, newdata = transform(boot_dat, x=1), type="response"))
  muhat0_ <- mean(predict(m1_, newdata = transform(boot_dat, x=0), type="response"))

  ## compute the odds from these average probabilities
  odds1_ <- muhat1_/(1-muhat1_)
  odds0_ <- muhat0_/(1-muhat0_)

  marginal_stand_ <- log(odds1_/odds0_)

  # IP WEIGHTING
  # create the propensity score in the dataset
  boot_dat$propensity_score <- glm(x ~ .,
```

```
                                    data = boot_dat[,-y],
                                    family = binomial("logit"))$fitted.values

  # stabilized inverse probability weights
  boot_dat$sw <- (mean(boot_dat$x)/boot_dat$propensity_score)*boot_dat$x +
    ((1-mean(boot_dat$x))/(1-boot_dat$propensity_score))*(1-boot_dat$x)

  # trim the weights
  boot_dat$sw <- ifelse(boot_dat$sw>quantile(boot_dat$sw, .995),
                        quantile(boot_dat$sw, .995),
                        boot_dat$sw)

  # outcome model
  m2_ <- glm(y ~ x,
             data = boot_dat,
             weights = sw,
             family = quasibinomial(link="logit"))

  ip_weighting_ <- summary(m2_)$coefficients[2,1]

  return(c(marginal_stand_, ip_weighting_))
}

analysis_data <- analysis_data %>% select(-propensity_score, -sw)

boot_res <- lapply(1:500, function(x) boot_func(x, data = analysis_data))

boot_SE <- apply(do.call(rbind, boot_res), 2, sd)

# SIMULATION FUNCTION OUTPUT

res <- data.frame(intercept = intercept,
                  sample_size = sample_size,
                  c_number = c_number,
```

```
                        cov_mat  = cov_mat,
                        diag_cov = diag_cov,
                        marginal_standardization_estimate,
                        marginal_standardization_SE = boot_SE[1],
                        ip_weighting_estimate = ip_weighting_estimate[1],
                        ip_weighting_robust_SE = ip_weighting_estimate[2],
                        ip_weighting_boot_SE = boot_SE[2])

    return(res)
}

# simulation function parameters

parm_data <- expand.grid(
  index = 1:200,
  sample_size = 1000,
  intercept = -2,
  c_number = c(10, 25),
  cov_mat = 0,
  diag_cov = 1
)

head(parm_data, 3)

tail(parm_data, 3)

simulation_results <- mclapply(1:nrow(parm_data),
                               function(x) simulation_function(index = parm_data[x,]$index,
                                                               intercept = parm_data[x,]$intercept,
                                                               sample_size = parm_data[x,]$sample_size,
                                                               c_number = parm_data[x,]$c_number,
                                                               cov_mat = parm_data[x,]$cov_mat,
                                                               diag_cov = parm_data[x,]$diag_cov),
                               mc.cores = detectCores() - 2)
```

```
sim_res <- do.call(rbind, simulation_results)

# # save the data to file!

head(sim_res)

write_csv(sim_res, here("lectures/06_Section6_SimulationInPractice", "simulation_results_MSIPW.csv"))
```

9 Performance Evaluation

This simulation takes some time to run, so let's be sure we save our results to a file so that we can re-import them without having to run the simulation again to rebuild the entire dataset. We can import our saved data back into R using standard code, such as the `read` functions in the tidyverse:

```
a <- read_csv(here("lectures/06_Section6_SimulationInPractice", "simulation_results_MSIPW.csv"))

head(a)
```

```
## # A tibble: 6 x 10
##   intercept sample_size c_number cov_mat diag_cov marginal_standardization_est~1
##       <dbl>       <dbl>    <dbl>   <dbl>    <dbl>                          <dbl>
## 1        -2        1000       10       0        1                          0.718
## 2        -2        1000       10       0        1                          0.445
## 3        -2        1000       10       0        1                          0.376
## 4        -2        1000       10       0        1                          0.610
## 5        -2        1000       10       0        1                          0.279
## 6        -2        1000       10       0        1                          0.499
## # i abbreviated name: 1: marginal_standardization_estimate
## # i 4 more variables: marginal_standardization_SE <dbl>,
## #   ip_weighting_estimate <dbl>, ip_weighting_robust_SE <dbl>,
## #   ip_weighting_boot_SE <dbl>
```

```
# to make things easier, let's create objects with our true values:

true_log_or1 <- 0.4132932
true_log_or2 <- 0.289803
```

Let's start with the mean of the point estimates obtained from marginal standardization and IP weighting:

```
# the true log OR marg1 (c_number = 10) is: 0.4132932
#
# the true log OR marg2 (c_number = 25) is: 0.289803

a %>%
  group_by(c_number) %>%
  summarize(meanMS = mean(marginal_standardization_estimate),
            meanIPW = mean(ip_weighting_estimate))
```

```
## # A tibble: 2 x 3
##   c_number meanMS meanIPW
##      <dbl>  <dbl>   <dbl>
## 1       10  0.393   0.466
## 2       25  0.295   0.512
```

These results suggest some potential problems with the IP weighting approach. Let's compute the bias:

```
# the true log OR marg1 (c_number = 10) is: 0.4132932
#
# the true log OR marg2 (c_number = 25) is: 0.289803

a %>%
  group_by(c_number) %>%
  summarize(biasMS = mean(marginal_standardization_estimate - true_log_or1),
            biasIPW = mean(ip_weighting_estimate - true_log_or2))
```

```
## # A tibble: 2 x 3
```

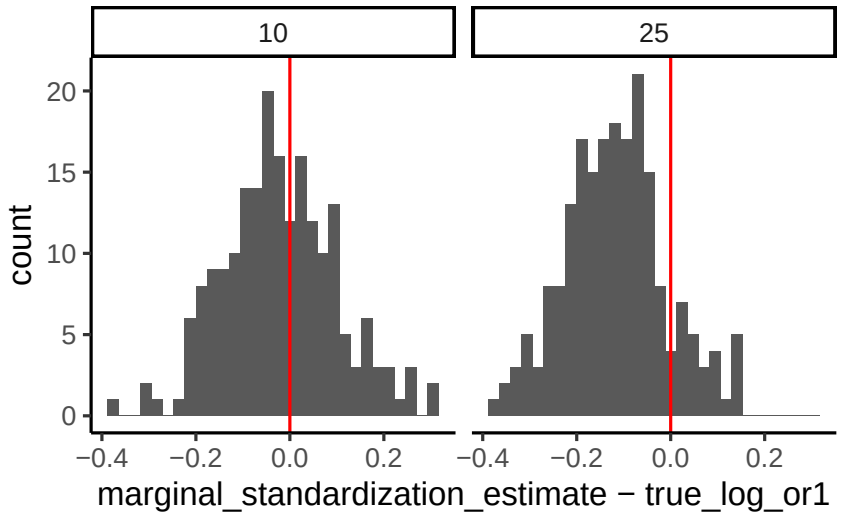
```
##   c_number  biasMS biasIPW
##      <dbl>   <dbl>   <dbl>
## 1       10 -0.0199   0.176
## 2       25 -0.118    0.222
```

These bias estimates seem fairly large, particularly for IP weighting. We can ask if these estimates are compatible with an underlying hypothesis of "no bias", after all, we only did 200 simulations, which is a fairly low number. Let's compute standard errors for each of these bias estimates. Also, it might help us to construct a dataset with all these results:

```
mc_se_bias <- function(x, n){
  sqrt(sum((x - mean(x))^2)/(n*(n-1)))
}

bias_results <- a %>%
  group_by(c_number) %>%
  summarize(biasMS = mean(marginal_standardization_estimate - true_log_or1),
            biasIPW = mean(ip_weighting_estimate - true_log_or2),

            biasMS_se = mc_se_bias(marginal_standardization_estimate - true_log_or1, n = 200),
            biasIPW_se = mean(ip_weighting_estimate - true_log_or2, n = 200),

            biasMS_p.value = round(2*(1 - pnorm(abs(biasMS/biasMS_se))),4),
            biasIPW_p.value = round(2*(1 - pnorm(abs(biasIPW/biasIPW_se))),4))

bias_results
```

```
## # A tibble: 2 x 7
##   c_number  biasMS biasIPW biasMS_se biasIPW_se biasMS_p.value biasIPW_p.value
##      <dbl>   <dbl>   <dbl>     <dbl>      <dbl>          <dbl>           <dbl>
## 1       10 -0.0199   0.176   0.00853      0.176         0.0196           0.317
## 2       25 -0.118    0.222   0.00754      0.222         0                0.317
```

```
ggplot(a) +
  geom_histogram(aes(marginal_standardization_estimate - true_log_or1)) +
  geom_vline(xintercept = 0, color = "red") +
  facet_wrap(~c_number)
```




```
ggplot(a) +
  geom_histogram(aes(ip_weighting_estimate - true_log_or2)) +
  geom_vline(xintercept = 0, color = "red") +
  facet_wrap(~c_number)
```