Does the size of the adjustment set follows the same variance-bias trade off as in classic ML?

Iyar Lin

04 December, 2018

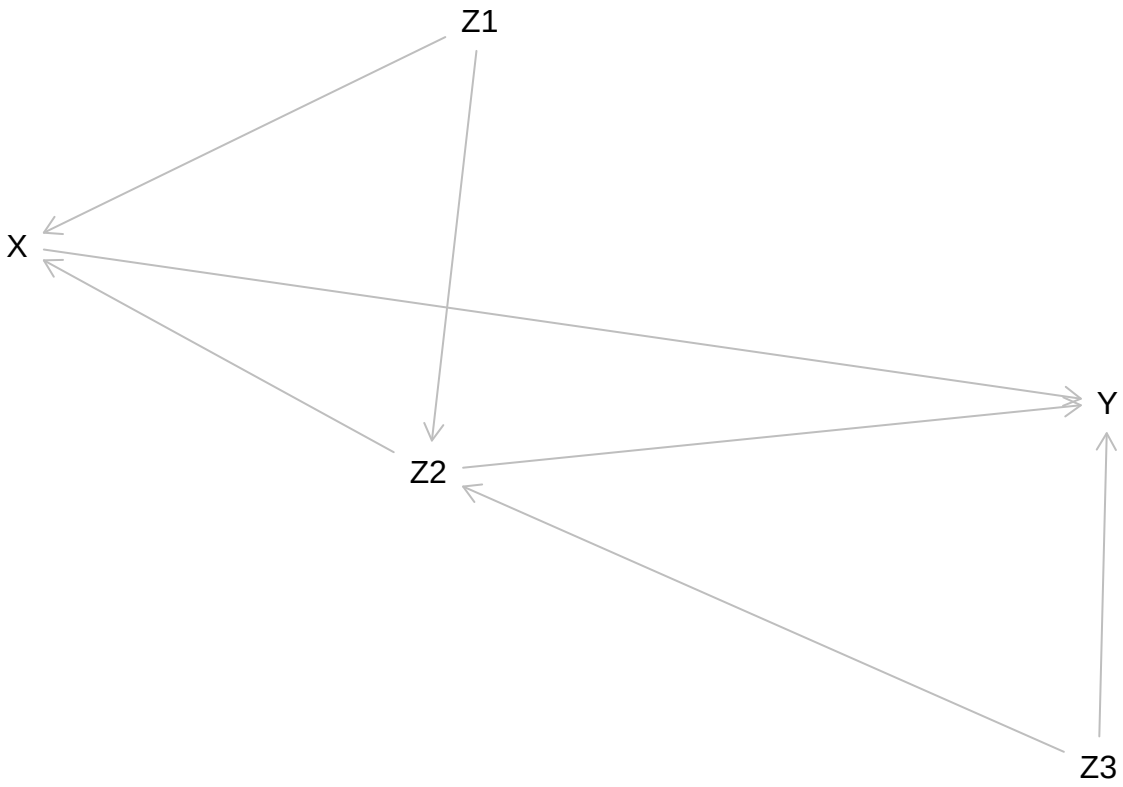
Contents

Does the size of the adjustment set follows the same variance-bias trade off as in classic ML?

Let's look at the DAG below:

```
g <- dagitty("dag {
Y [outcome]
X [exposure]
X -> Y [beta = 0.2]
Z1 -> X [beta = 0.1]
Z1 -> Z2 [beta = -0.1]
Z2 -> X [beta = 0.3]
Z2 -> Y [beta = 0.05]
Z3 -> Z2 [beta = -0.05]
Z3 -> Y [beta = -0.1]
}")

plot(graphLayout(g))
```



If we're interesting in estimating $\mathbb{E}(Y|do(X = x)$ there's several confounding paths that we need to block.

The exact possible adjustment sets are:

```
M = 1000
print(adjustmentSets(g, type = "all"))
```

```
## { Z1, Z2 }
## { Z2, Z3 }
## { Z1, Z2, Z3 }
```

We can see that we can either control for 2 or 3 out of the confounding variables. Assuming we can measure them all, and we know from the graph they are all "relevant" (e.g. not white noise) does it make sense to use all 3 instead of just a subset?

Below I simulate 1000 dataset for a grid of sample size N and compare the standard deviation of the coefficient estimate as well as the R-squered on a hold out test set:

```
N_grid <- seq(4, 40, 10)
model_performance <- data.frame(model = rep(c("small", "large"), length(N_grid)),
                        N = rep(N_grid, each = 2),
                        sd = NA,
                        R_2 = NA)

for(n in N_grid){
  sim_data <- replicate(M, simulateSEM(g, N = n), simplify = F)
  sim_test_data <- replicate(M, simulateSEM(g, N = n), simplify = F)
  sim_small_model <- lapply(sim_data, function(data) lm(Y ~ X + Z1 + Z2, data = data))
  sim_large_model <- lapply(sim_data, function(data) lm(Y ~ X + Z1 + Z2 + Z3, data = data))

  model_performance$sd[model_performance$N == n & model_performance$model == "small"] <- sd(sapply(sim_
  model_performance$sd[model_performance$N == n & model_performance$model == "large"] <- sd(sapply(sim_
```

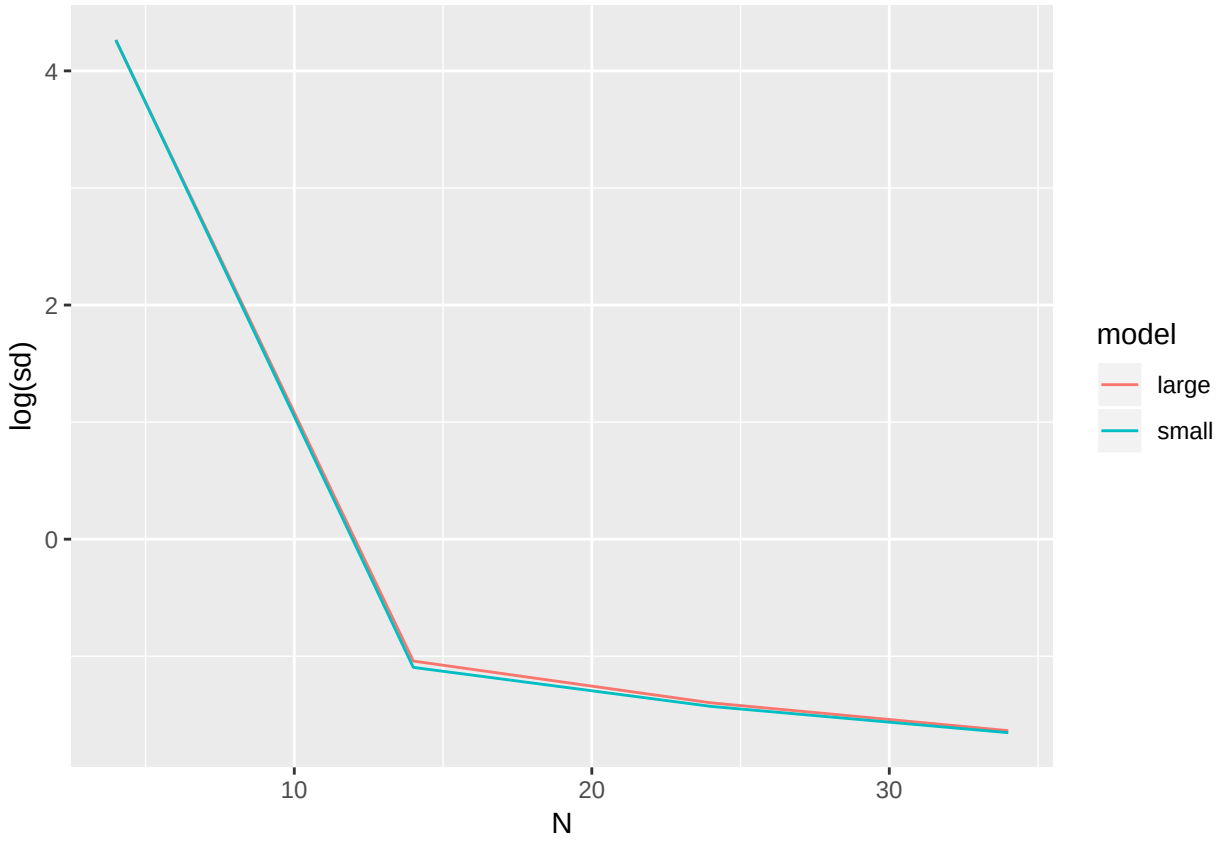
```
  model_performance$R_2[model_performance$N == n & model_performance$model == "small"] <- mean(mapply(fu
    sqrt(mean((predict(model, data) - data$Y)^2))
  },
  data = sim_test_data, model = sim_small_model))

  model_performance$R_2[model_performance$N == n & model_performance$model == "large"] <- mean(mapply(fu
    sqrt(mean((predict(model, data) - data$Y)^2))
  },
  data = sim_test_data, model = sim_large_model))

}
```

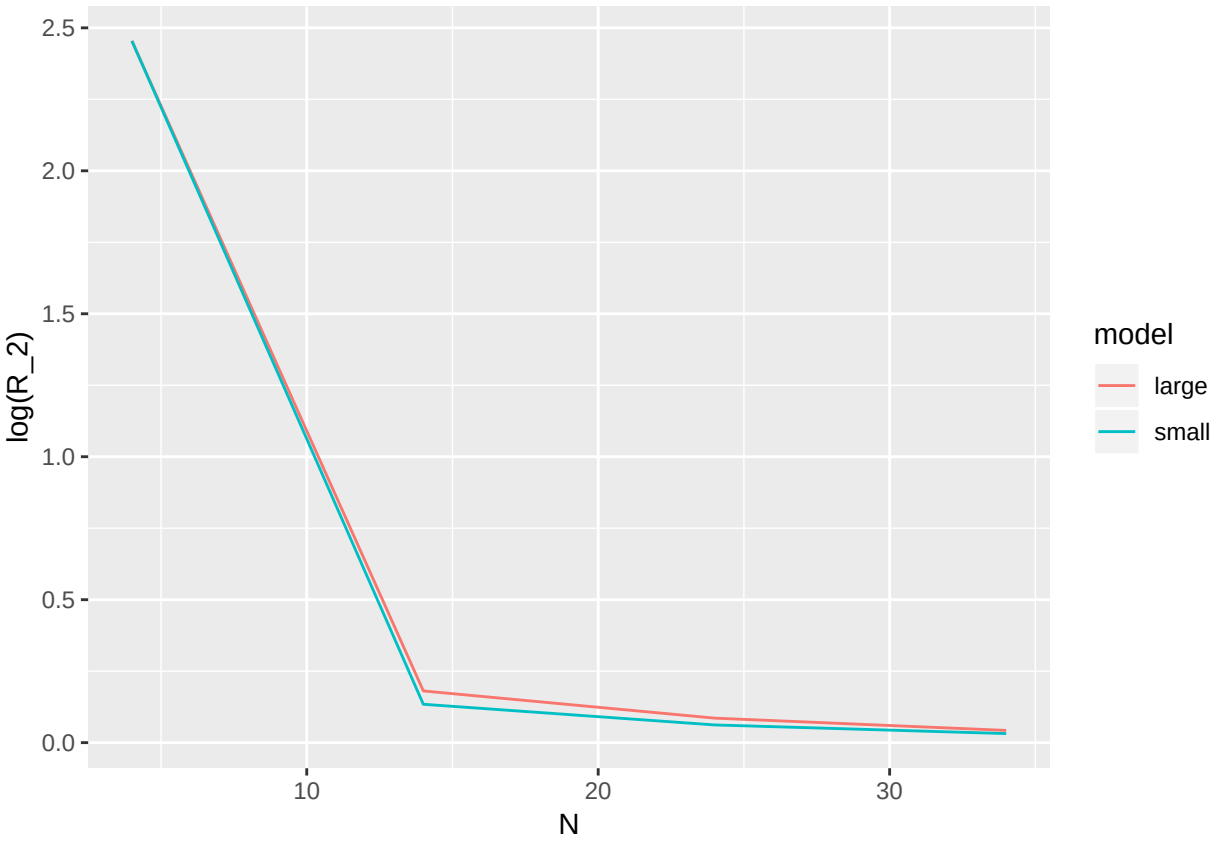
Below are the sd results:

```
model_performance %>% ggplot(aes(N, log(sd), color = model)) + geom_line()
```



Below are the R-squared results:

```
model_performance %>% ggplot(aes(N, log(R_2), color = model)) + geom_line()
```



Oddly enough it looks like if anything, using less variables is better!