Big-Picture Takeaways from Selection on Observables

Joel Ferguson

10/18/2021

Big-Picture Takeaways from Selection on Observables

We've now spent about half of the course talking about selection on observables designs. We've also said in almost every class that it's hard to publish using these methods now because we very rarely believe that unconfoundedness holds. As such, you may be wondering "Why do we bother spending half a semester learning about these methods we'll probably never use?" I think there are at least three reasons, which I'll describe in increasing order of their subtlety.

1) A lot of papers use these methods

Probably the most obvious reason to learn all these different selection on observables designs is that a huge number of papers do use them, particularly papers that are a bit dated. Understanding selection on observables designs is important for understanding the good and bad aspects of the methodology employed in these papers given the data available.

As has kind of been alluded to (and will be explored more explicitly a bit later), selection on observables designs are somewhat "interchangeable" in the sense that they all rely on the same fundamental assumption of strong unconfoundedness. In the best cases, particular methods are used because they make use of additional knowledge of the DGP, so understanding why a paper used a particular method can give you insights that are generally useful for doing analysis on a similar question. Understanding the issues can help you develop better approaches and identify what are the additional data requirements (if any) to implement that approach.

Finally, sometimes we think strong unconfoundedness *does* hold! It's important to be able to identify those cases when they do arise in research so we can take advantage of the extremely well-developed methods of analysis in that setting (more on this later too).

2) Understanding why unconfoundedness doesn't hold is important for choosing a good selection on unobservables design

Along with learning these methods, we've learned what goes wrong in selection on observables designs when unconfoundedness doesn't hold. We've learned about omitted variables bias, bad controls, and how we can try to say how much important variation must be stuck in the error term to shrink our estimated treatment effects to zero. We've also learned how to think about what might be getting stuck in the error term. All of these skills are incredibly important for selection on unobservables designs, where the goal is to somehow deal with those things that we don't observe that are correlated with treatment and outcome. It's nearly impossible to deal with these unobservable confounders if we don't know what they are or how they might be messing up a simple OLS regression.

3) Sometimes, we have enough observables that the stuff left in the error doesn't matter much

In the age of *big data*, we occasionally find ourselves in settings where we have huge numbers of observables. If these observables account for the vast majority of the variation in the untreated potential outcomes, we don't lose much by just using a selection on observables design. We got an intuition for this when we learned about Altonji, Elder, and Taber (2005). Recently, it's been shown empirically that there can be reason to believe with enough data (both observations and covariates) we can get estimates that are indistinguishable from unbiased covariates. In particular, Eckles and Bakshy (2020) looked at whether or not they could reproduce an estimate identified by a large RCT using only observables.

JOURNAL OF THE AMERICAN STATISTICAL ASSOCIATION
2021, VOL. 116, NO. 534, 507–517: Applications and Case Studies
https://doi.org/10.1080/01621459.2020.1796393

Bias and High-Dimensional Adjustment in Observational Studies of Peer Effects

Dean Eckles[a,b] and Eytan Bakshy[c]

[a]Sloan School of Management, Massachusetts Institute of Technology, Cambridge, MA; [b]Institute for Data, Systems & Society, Massachusetts Institute of Technology, Cambridge, MA; [c]Facebook, Menlo Park, CA

We've seen this sort of idea before in the LaLonde (1986) study and Dehejia and Wahba's followup (1999). There are two twists here: 1) Eckles and Bakshy are attempting to identify peer effects, particularly how much person A seeing that their friend person B sharing an item on Facebook makes person A more likely to share the same item 2) The data set is massive, with 660 million control observations over 3000 covariates.

The experiment in question randomized whether or not Facebook users were shown a particular link shared by a friend on their feed. Via this large experiment, they found that seeing that person B sharing a link makes person A 6.7 times more likely to share that link. They then dropped their experimental control group (user-link pairs where the link was hidden despite a friend sharing it) and make a non-experimental control group of people whose friends never share the link in question. They then attempted to recover the causal estimate of the treatment effect by adjusting for observables. In particular, they used a propensity score blocking estimator, similar to the one described in the last section, where in the most successful models the propensity score was estimated with a penalized logit model using as observables how many times in the past 6 months the user had shared links from a large collection of websites.

Table 2. Experimental and observational estimates of peer effects.

Model	Demographics	General behavior, degree	Same domain shares	Other domain shares	$\hat{p}^{(0)} \times 10^4$	$\widehat{RR}$	$\hat{\delta} \times 10^4$
exp					1.96 [1.88, 2.03]	6.73 [6.48, 6.97]	11.21 [10.95, 11.47]
naive					0.47 [0.39, 0.54]	28.15 [23.80, 32.51]	12.70 [12.45, 12.96]
D	Yes				0.59 [0.51, 0.68]	22.15 [19.28, 25.03]	12.58 [12.32, 12.83]
A	Yes	Yes			0.66 [0.59, 0.74]	19.81 [17.61, 22.00]	12.51 [12.25, 12.76]
Ds	Yes		Yes		1.51 [1.36, 1.66]	8.72 [7.80, 9.64]	11.66 [11.38, 11.94]
As	Yes	Yes	Yes		1.52 [1.36, 1.68]	8.67 [7.75, 9.58]	11.65 [11.38, 11.93]
M	Yes	Yes		Yes	1.02 [0.91, 1.14]	12.86 [11.67, 14.06]	12.15 [11.89, 12.40]
AM	Yes	Yes		Yes	1.14 [1.00, 1.28]	11.55 [10.43, 12.67]	12.03 [11.75, 12.31]
Ms	Yes	Yes	Yes	Yes	1.72 [1.54, 1.91]	7.64 [6.88, 8.40]	11.45 [11.17, 11.73]
AMs	Yes	Yes	Yes	Yes	1.77 [1.57, 1.96]	7.46 [6.66, 8.25]	11.41 [11.08, 11.73]

NOTE: Estimates of the probability of sharing if not exposed $p^{(0)}$, relative risk (RR), and the risk difference (δ) for each model with 95% bootstrap standard confidence intervals in brackets. $p^{(0)}$ and δ are shown in basis points for readability.

Their results show that constructing the propensity score using measures of previous behavior (Model Ds and below) does a very good job of estimating the counterfactual of interest (shown in the first row). In particular, the best of these models produce estimates that are **statistically indistinguishable** from the experimental result. Intuitively, past behavior does a really good job of predicting present behavior, so much so that the bias induced by not being able to account for shocks to an individuals friend network is essentially 0.

Bonus point 4) for randomistas: Deep knowledge of selection on observables can help you design better experiments

A quick bonus point I wanted to include for any development folks in ARE and students from other departments who might design RCTs is that when we design an experiment we know that selection on observables is satisfied (if there's perfect compliance or we estimate an ITT) because **we assigned treatment**. This means that you can use all of the tools we've learned up to this point to increase power to detect effects, estimate heterogeneous treatment effects (e.g. via the Causal Forest of Wager and Athey 2018), estimate dose-response functions, etc.

Machine Learning and Causal Inference in Selection on Observables Designs

Somewhat related to point 3 above is that selection on observables designs are all about approximating $\mathbb{E}[Y_i(0), Y_i(1)|X_i]$ well. This is exactly what machine learning (ML) excels at and as a result there's been an explosion of work showing how ML can aid in causal inference, particularly in selection on observables settings (of which Belloni, Chernozhukov, and Hansen 2014, which we saw in class, is only the tip of the iceberg).

To illustrate the power of these methods, I'm going to simulate data where potential outcomes and the propensity score are all complicated non-linear functions of observables. I'll also add an "unobservable" determinant element of the propensity score that accounts for very little variation in treatment assignment and show that even when we can't account for this variable, an ML-based selection on observables method estimates the true treatment effect very well.

First, we'll set up the data

```
N <- 1000 # Number of obs
K <- 500 # Number of observables

# Observables are going to be iid U[0,1]
X <- matrix(runif(N*K),nrow = N)

# The unobservable is going to be correlated with one of the observables
W <- sapply(X[,1],function(x) rnorm(N,x)) # W is normally distributed with mean of the 1st observable

XW <- cbind(X,W) # Put X and W together for some estimates

# Make a bunch of cutoffs for cells
cell_edges <- matrix(sapply(c(1:(3*K)), function(x) sort(runif(3))),nrow=3*K,byrow = TRUE)

# Assign cells for treatment and POs
cell_assign_var <- function(x,k,var){
  if (var=="D"){
    m <- 0
  } else if (var=="Y0"){
    m <- 1
  } else if (var=="Y1"){
    m <- 2
  }

  if (x<cell_edges[K*m+k,1]){
    return(0)
  } else if (x<cell_edges[K*m+k,2]){
    return(1)
  } else {
    return(2)
  }
}

cell_assign <- function(r,var){
  # Use ternary to assign each combination of cells by a specific X to a single cell in R^K
  cell <- sum(sapply(c(1:K),function(k) cell_assign_var(X[r,k],k,var)*(3^(k-1))))
  return(cell)
}

# Assign the cells for Treatment and POs
cells_D <- sapply(c(1:N),cell_assign, var="D")
cells_Y0 <- sapply(c(1:N),cell_assign,var="Y0")
cells_Y1 <- sapply(c(1:N),cell_assign,var="Y1")

# P scores will be drawn from U[0.2,0.8]
D_unique <- unique(cells_D)
cell_pscores <- runif(length(D_unique),0.2,0.8)

# Y0 means will be drawn from U[-10,10]
Y0_unique <- unique(cells_Y0)
cell_y0_means <- runif(length(Y0_unique),-10,10)
```

```r
# Y1 means will be drawn from U[-9,11]
Y1_unique <- unique(cells_Y1)
cell_y1_means <- runif(Y1_unique,-9,11)

# Draw treatment and POs
D <- sapply(cells_D, function(x) rbernoulli(1,cell_pscores[which(x==D_unique)]))
Y0 <- sapply(cells_Y0, function(x) rnorm(1,cell_y0_means[x==Y0_unique]))
Y1 <- sapply(cells_Y1, function(x) rnorm(1,cell_y1_means[x==Y1_unique]))

# Print the ATE
ate <- mean(Y1-Y0)
ate
```

```
## [1] 0.694143
```

```r
# Print the naive estimate
mean(Y1*D-Y0*(1-D))
```

```
## [1] 0.4240892
```

```r
# Make a new draw of treatment assignments that is very slightly influenced by W
DW <- sapply(c(1:N), function(x) rbernoulli(1,cell_pscores[which(cells_D[x]==D_unique)]+0.01*W[x]))

# Print the naive estimate in this case
mean(Y1*DW-Y0*(1-DW))
```

```
## [1] 0.2323229
```

```r
# Combine everything into a df
df <- as_tibble(cbind(X,D,DW,Y0,Y1,cells_D)) %>%
  mutate(Y = Y1*D+Y0*(1-D),
         YW = Y1*DW+Y0*(1-DW))
```

```
## Warning: The `x` argument of `as_tibble.matrix()` must have column names if `.name_repair` is omitte
## Using compatibility `.name_repair`.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```

Now treatment and outcomes are all assigned according to a different set of 10-dimensional cells. In this setting, a blocking estimator like the one introduced last section would likely work well, but to maximize power we'd ideally like to know the size of all the cells (in X space). What we'll do instead is try to find the cells with a classification and regression tree (CART), an ML method that excels at this sort of task.

```r
# First, let's just run a linear regression and see how it does
ols <- lm(as.formula(paste0("Y~D+",paste(names(df)[1:K],collapse="+"))),
          df)
ols$coefficients[[2]]
```

```
## [1] 0.8050658
```

```
fml <- as.formula(paste0("D~",paste(names(df)[1:K],collapse="+")))

# Let's choose the max tree depth via cross-validation
depths <- c(1:10)
train_obs <- sample(c(1:N),4*floor(N/5))
df_train <- df[train_obs,]
df_val <- df[-train_obs,]
eval_tree <- function(depth){
  Dtree <- rpart(fml,
                 control = rpart.control(maxdepth = depth),
                data=df_train)
  pscores <- predict(Dtree,df_val)
  return(sum((pscores-df_val$D)^2))
}
depth_perfs <- sapply(depths, eval_tree)
best_depth <- depths[which.min(depth_perfs)]

Dtree <- rpart(fml,
                 control = rpart.control(maxdepth = best_depth),
                data=df)
pscores <- predict(Dtree,df)
df$pscore <- pscores

# Cart P-score ATE
df_su <- df %>%
  filter(pscore >0, pscore <1)
(sum(df_su$D*df_su$Y/df_su$pscore)/sum(df_su$D/df_su$pscore))-sum((1-df_su$D)*df_su$Y/(1-df_su$pscore))
```

```
## [1] 0.8557118
```

Here, OLS and the CART-estimated IPW estimator do about as well at estimating the ATE (CART does a bit worse). Now let's see how it does when we use the treatment assignment that was influenced by the unobservable.

```
ols <- lm(as.formula(paste0("Y~DW+",paste(names(df)[1:K],collapse="+"))),
          df)
ols$coefficients[[2]]
```

```
## [1] -0.02377511
```

```
fml <- as.formula(paste0("DW~",paste(names(df)[1:K],collapse="+")))
depth_perfs <- sapply(depths, eval_tree)
best_depth <- depths[which.min(depth_perfs)]

DWtree <- rpart(fml,
                 control = rpart.control(maxdepth = best_depth),
                data=df)
pscoresW <- predict(DWtree,df)
df$pscoreW <- pscoresW
# Cart P-score ATE
df_su <- df %>%
  filter(pscoreW >0, pscoreW <1)
(sum(df_su$D*df_su$Y/df_su$pscoreW)/sum(df_su$D/df_su$pscoreW))-sum((1-df_su$D)*df_su$Y/(1-df_su$pscoreW
```

```
## [1] 0.8360128
```

Here we get a pretty substantial performance improvement over OLS. CART does a decent job of handling the fact that selection into treatment on the unobservable W doesn't actually matter much, whereas the OVB induced by W is enough to substantially throw off our OLS estimate.