Variance Reduction and Confounding Bias

MKTG 6279

Week 2

This week

- A/B variance reduction
- Propensity Score Matching (PSM)
- Augmented Inverse Probability Weighting (AIPW)
- DoubleML (Double/Debiased ML)

Variance reduction

To prevent false negatives ("missed detection"), we try to minimize uncertainty in our estimate of the treatment effect

This is reflected in the standard error of the treatment effect

Random assignment ensures that the difference between treatment and control groups is *unbiased* (i.e., on average, the difference is accurate)

- But this does not consider the *uncertainty* in the estimate

Variance is a function of sample size

$$Var(\hat{\tau}) = \frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}$$

For groups 1 and 2 (e.g., "treatment" and "control")

$$CI(\hat{\tau}) = \hat{\tau} \pm z_{\alpha/2}\sqrt{Var(\hat{\tau})}$$

See it in R...

```
var_t = 5 #variance outcomes
var_c = 2
n_t = 100 #number of observations
n_c = 100
tau = 13  #treatment estimate
a = .05   #significance level
z = abs(qnorm(a/2)) #1.96

var_tau = var_t/n_t + var_c/n_c
var_tau; tau - z*sqrt(var_tau); tau + z*sqrt(var_tau)
```

```
## [1] 0.07
```

```
## [1] 12.48144
```

```
## [1] 13.51856
```

Double n

```
n_t = 100*2
n_c = 100*2

var_tau = var_t/n_t + var_c/n_c
var_tau; tau - z*sqrt(var_tau); tau + z*sqrt(var_tau)
```

```
## [1] 0.035
```

```
## [1] 12.63332
```

```
## [1] 13.36668
```

Or half the variance in Y

```
var_t = 5/2 #halved
var_c = 2/2
n_t = 100   #reset
n_c = 100

var_tau = var_t/n_t + var_c/n_c
var_tau; tau - z*sqrt(var_tau); tau + z*sqrt(var_tau)
```

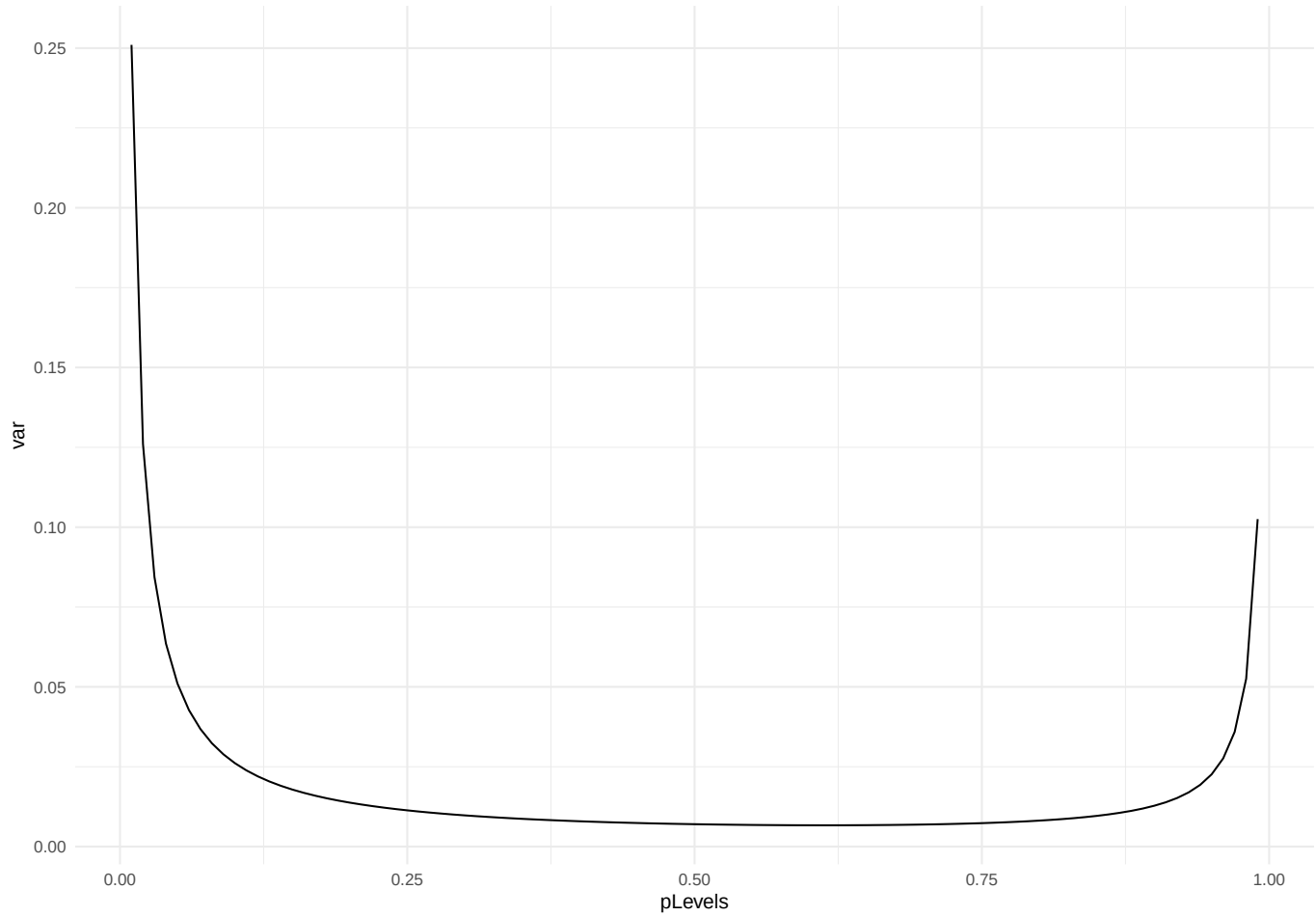
```
## [1] 0.035
```

```
## [1] 12.63332
```

```
## [1] 13.36668
```

Given a fixed n, better to have an even split

```
getTau  = function(p = .5,n = 1000) {
  var_t/(p*n) + var_c/((1-p)*n)}
pLevels = seq(0.01,.99,.01)
splitN = data.frame(pLevels,var = getTau(pLevels))
```



General rule of thumb

$$n \approx \frac{16\sigma^2}{(\mu_1 - \mu_2)^2} = \frac{16}{\Delta^2}$$

where $\Delta = \frac{\mu_1 - \mu_2}{\sigma}$ or the *standardized effect size* or **Cohen's d**

In other words, treatment effect in terms of *standard deviations of difference*

Simple example

Average monthly spend is \$40 pre-treatment with a SD of \$18

We want to run an experiment, with the goal of increasing average monthly spend to \$46

How many observations are needed for enough power?

$$\frac{16 \times 18^2}{(46 - 40)^2} = 144$$

In R

```
library(pwr)
pwr.t.test(d = (46-40)/18,
           sig.level = .05,
           power = .8)
```

```
##
##      Two-sample t test power calculation
##
##               n = 142.2462
##               d = 0.3333333
##       sig.level = 0.05
##           power = 0.8
##     alternative = two.sided
##
## NOTE: n is number in *each* group
```

More realistic

Changing conversion rate from 2% to 2.02% requires 3.8M users...

```
h = ES.h(.02,.0202) ##Cohen's d for proportions
pwr.p.test(h = h,
    sig.level = 0.05,
    power = 0.8,
    alternative = "two.sided")
```

```
##
##      proportion power calculation for binomial distribut
##
##               h = 0.00142509
##               n = 3864753
##       sig.level = 0.05
##           power = 0.8
##     alternative = two.sided
```

Stratified Sampling

Stratified sampling lowers variance by sampling each sub-population according to its distribution at the population level

For example: if 90% of your users are *mobile* and 10% are *desktop* then random sampling should be *close* to this, but why not make it *exact*?

Toy example

```
pop      = 10000
mobile   = rep(1:0,c(pop*.9,pop*.1))
d = sample(0:1,pop,TRUE)
treatment_effect = 3
sales =  10 + treatment_effect*d + 12*mobile +
          rnorm(pop,0,4)
pop_df = data.frame(sales,d,mobile)
#regular sample
n = 300
set.seed(1);reg_df = pop_df %>% sample_n(n)
prop.table(table(reg_df$mobile))
```

```
##
##         0         1
## 0.1266667 0.8733333
```

Stratified Sample

```
n_mobile  = n*.9
n_desktop = n*.1

ss_df = bind_rows(
  pop_df %>% filter(mobile == 1) %>%
  sample_n(n_mobile),
  pop_df %>% filter(mobile == 0) %>%
  sample_n(n_desktop))

prop.table(table(ss_df$mobile))
```

```
##
##   0   1
## 0.1 0.9
```

Compare ATE variance...

```
showlm(lm(sales ~ d,reg_df))
```

```
## # A tibble: 2 x 4
##   term        estimate std.error p.value
##   <chr>          <dbl>     <dbl>   <dbl>
## 1 (Intercept)    20.7      0.481       0
## 2 d               2.86     0.66        0
```

```
showlm(lm(sales ~ d,ss_df))
```

```
## # A tibble: 2 x 4
##   term        estimate std.error p.value
##   <chr>          <dbl>     <dbl>   <dbl>
## 1 (Intercept)    21.3      0.462       0
## 2 d               2.67     0.653       0
```

CUPED

Controlled-experiment Using Pre-Existing Data: variance reduction technique to increase the sensitivity of experiments

In short: reduce the "noise" in the data in order to detect the true impact of changes

Why? Minor product changes tend to produce small effects, but companies still want to be able to detect these changes because they scale across millions of users

Some history

Introduced by researchers at Microsoft in 2013

Widely used by many large companies such as:

- Netflix
- Booking
- Meta
- Airbnb
- DoorDash

and many others. . .

Intuition

From Booking link on previous slide

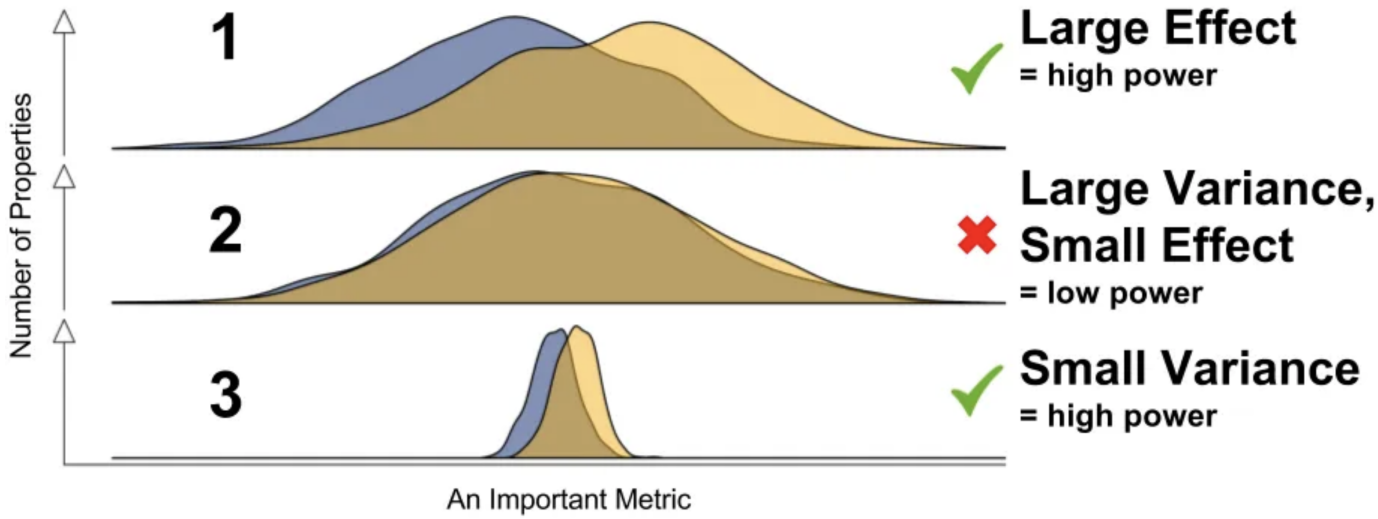


Figure 1. Distributions of properties in base (blue) and variant (yellow) for an important metric in three experiments

Problems with low power

With low power, statistical tests tend to fail even if the change (however small) has a meaningful effect

An experiment is underpowered when the treatment effect is too small relative to the metric's variance for a given sample size

By reducing the variance in the estimator (e.g., the ATE) allows us to:

- Detect smaller effects
- Detect the same effect but with a smaller sample size

Enter CUPED

Intuition: variance in pre-experiment data is unrelated to the treatment effect and can be removed

Example from Booking.com

- The number of bookings per property per day can range from zero to thousands, so it has large variance
- But we know the average bookings per day for each property involved in the experiment
- We can use this to see change in bookings at the property-day level

CUPED uses pre-experiment data to remove variance in a straightforward and scalable way

Details

Y_1 is the outcome of interest

D indicates if the observation is in the treatment (1) or control (0) group

X is pre-treatment variable used to reduce the variance

Good properties of X

- Not affected by the treatment assignment (e.g., recieving a treatment is not a function of X)
- Highly correlated with Y_1

Typically people use pre-treatment Y values: Y_0

Estimation

1) Regress Y_1 onto Y_0 and get $\hat{\theta}$
2) Compute $Y_1^{cuped} = Y_1 - \hat{\theta}(Y_0 - \bar{Y}_0)$
3) Use Y_1^{cuped} instead of Y_1

Booking.com Example

```
load('data/booking.rdata')
str(booking)
```

```
## 'data.frame':    5000 obs. of  3 variables:
##  $ bookings    : num  16 22 20 17 13 12 11 15 17 17 ...
##  $ bookings_pre: int  12 17 14 16 10 6 6 12 13 13 ...
##  $ treated     : int  1 1 1 0 0 1 1 0 1 0 ...
```

Booking ATE

```
y1        = booking$bookings
treated   = booking$treated

showlm(lm(y1~treated))
```

```
## # A tibble: 2 x 4
##   term        estimate std.error p.value
##   <chr>          <dbl>     <dbl>   <dbl>
## 1 (Intercept)    14.1      0.076       0
## 2 treated         2.95     0.107       0
```

Booking CUPED

```
y0 = booking$bookings_pre
lm_booking = lm(y1 ~ y0)
theta      = coef(lm_booking)[2]
y1_cuped   = y1 - theta*(y0 - mean(y0))
showlm(lm(y1_cuped ~ treated))
```

```
## # A tibble: 2 x 4
##   term        estimate std.error p.value
##   <chr>          <dbl>     <dbl>   <dbl>
## 1 (Intercept)    14.1      0.029       0
## 2 treated         2.98     0.041       0
```

Why not just add $y0$ like CATE?

```
showlm(lm(y1 ~ treated + y0))
```

```
## # A tibble: 3 x 4
##   term        estimate std.error p.value
##   <chr>          <dbl>     <dbl>   <dbl>
## 1 (Intercept)     1.97     0.076       0
## 2 treated         2.98     0.041       0
## 3 y0              1.01     0.006       0
```

Repeat 1,000 times on simulated data

```
n = 1000
nSim = 1000
estD = data.frame(ate=rep(NA,nSim),cuped=NA,lm=NA)
te = 5
for(i in 1:nSim){
  y0 = rnorm(n,100,sd=20)
  d  = sample(c(0, 1),n,replace = TRUE)
  y1 = 10 + y0 + te*d + rnorm(n,0,20)
  theta = cov(y0,y1)/var(y0)
  y1_cuped = y1 - theta*(y0 - mean(y0))

  estD[i,'ate']   = coef(lm(y1 ~ d))[2]
  estD[i,'cuped'] = coef(lm(y1_cuped ~ d))[2]
  estD[i,'lm']    = coef(lm(y1 ~ d + y0))[2]
}
```

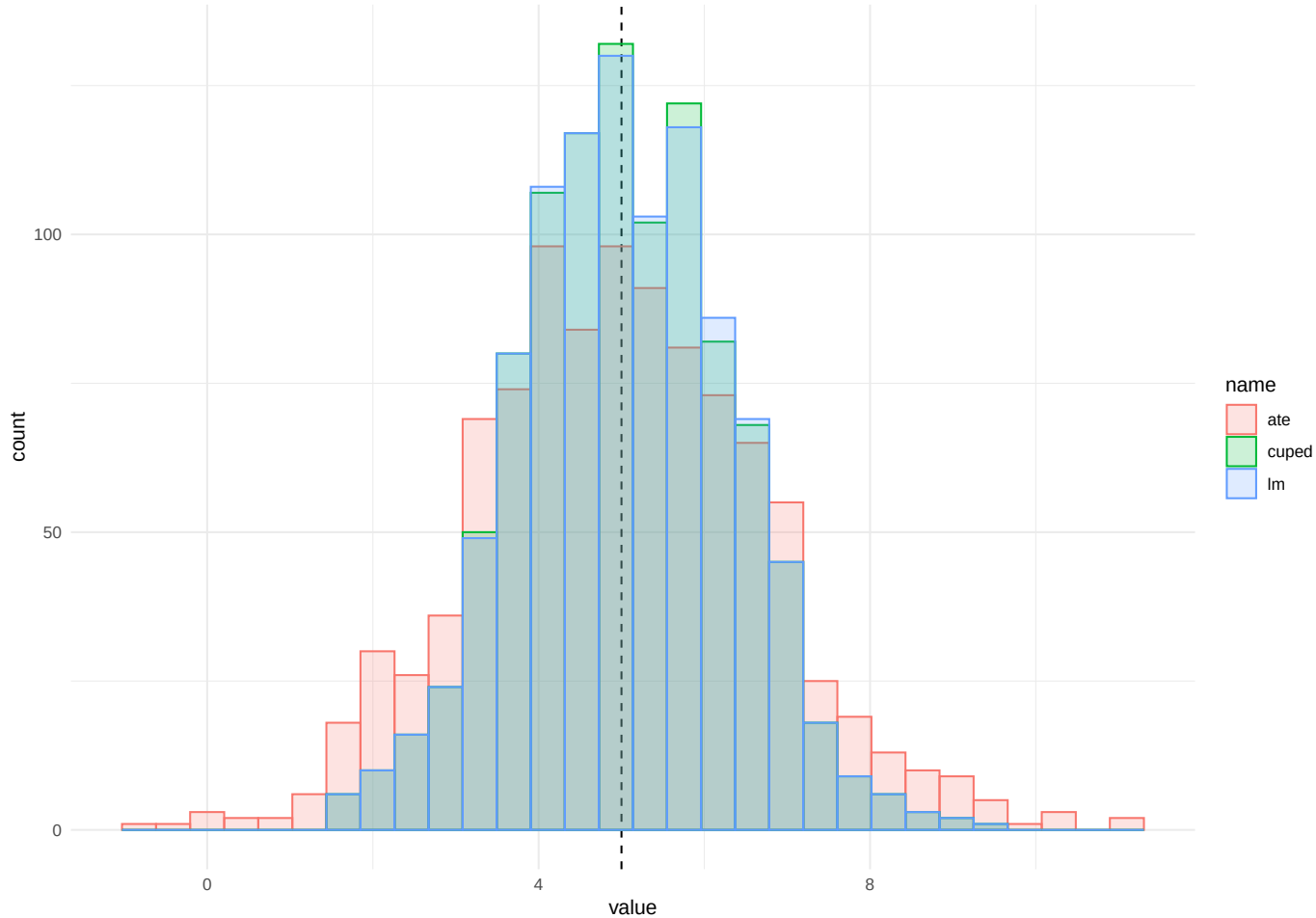
Simlar means and variances...

```
sapply(estD, function(x) c(mean = mean(x),
                           variance = var(x)))
```

```
##                ate    cuped       lm
## mean     4.967991 5.034234 5.039104
## variance 3.007846 1.631237 1.634795
```

CUPED is *very* similar to CATE

At least for "simple average metrics"



Non-Experimental Causal Inference

Treatment assignment is not always random

In many cases, people are not offered treatments randomly

For example, randomization may have occurred at the zip code level, but not the individual level

Or there was no randomization and you only have observational data

- For example, marketing offers sent to customers who meet certain criteria
- On Google Ads, users exposed to the ad might also be more likely to make purchases due to other factors (e.g., search engine activity)

Methods

Three popular approaches to dealing with this:

1) **Propensity Score Matching (PSM)**: match observations from the treatment and control groups based on covariates to increase comparability
2) **Augmented Inverse Probability Weighting (AIPW)**: provides a "doubly robust" estimate of the treatment effect using both propensity scores and outcomes
3) **Double ML**: Improvement from AIPW by using ML for more flexible mappings to the propensity scores and outcomes

Propensity Score Matching (PSM)

We observe treatment and control outcomes, but not necessarily from an RCT

For example, treatments may be selected by the individual rather than randomly assigned

PSM attempts to balance the groups on confounding factors

- Find observations that are very similar, but one received the treatment and the other the control (sort of like RDD)

How does it work?

1) Calculate propensity scores via logistic regression: $Pr(treatment|X)$
2) Check the matched data (some observations may be removed)
3) Estimate treatment effects on the matched data

Casino Example

You work for the Wynn in Las Vegas

You want to evaluate the effectiveness of an offer on return trip spend (i.e., play volume)

Those who received the offer might be different from those who did not

Data

```
load('data/casino.rdata')
str(casino)
```

```
## 'data.frame':    5000 obs. of  5 variables:
##  $ age       : num  42 65 66 49 62 57 60 63 58 64 ...
##  $ income    : num  8501 5154 20335 104036 224515 ...
##  $ loyalty   : num  0 0 0 1 0 0 0 1 1 0 ...
##  $ treatment: int  0 0 0 1 0 0 0 0 0 0 ...
##  $ spend     : num  343 887 682 828 1036 ...
```

ATE

```
showlm(lm(spend ~ treatment,casino))
```

```
## # A tibble: 2 x 4
##   term        estimate std.error p.value
##   <chr>          <dbl>     <dbl>   <dbl>
## 1 (Intercept)     489.      5.33       0
## 2 treatment       298.     11.5        0
```

CATE

A lot of the ATE is simply because those in the LP play at a higher level (and more likely to receive an offer)

```
showlm(lm(spend ~ treatment +
            age + income + loyalty,casino))
```

```
## # A tibble: 5 x 4
##   term         estimate std.error p.value
##   <chr>           <dbl>     <dbl>   <dbl>
## 1 (Intercept)  299.       20.6     0
## 2 treatment    183.       11.2     0
## 3 age            0.372     0.357   0.296
## 4 income         0.003     0       0
## 5 loyalty      118.        9.22    0
```

Match with PSM

```
library(MatchIt)
ps_match = matchit(treatment ~
                   age + income + loyalty,casino)
```

Note: There is also exact matching, see the package for more information

What happened?

Keeps all treated, and finds non-treated that look most similar

```
ps_match
```

```
## A matchit object
##  - method: 1:1 nearest neighbor matching without replace
##  - distance: Propensity score
##              - estimated with logistic regression
##  - number of obs.: 5000 (original), 2160 (matched)
##  - target estimand: ATT
##  - covariates: age, income, loyalty
```

Check matching...

```
summary(ps_match)
```

See "loyalty":

- In the *original* data, 74% of the treated subjects and 31% of the control subjects were in the LP
- In the *matched* data 76% of the control subjects are in the LP

Goal:

- Get the standardized mean difference and empirical CDFs close to 0
- Get the variance ratio close to 1

Check propensity scores

Ideally, unmatched propensity scores are far away from the treated group (and would thus require greater extrapolation if kept)

```
plot(ps_match, type = "jitter", interactive = FALSE)
```

Check densities

```
plot(ps_match, type = "density",
     interactive = FALSE,
     which.xs = ~income + loyalty)
```

Get CATE

```
casino_md = match.data(ps_match)
showlm(lm(spend ~ treatment + age + income + loyalty,
          casino_md))
```

```
## # A tibble: 5 x 4
##   term         estimate std.error p.value
##   <chr>           <dbl>     <dbl>   <dbl>
## 1 (Intercept)  221.       34.6     0
## 2 treatment    201.       12.7     0
## 3 age            0.682     0.538   0.205
## 4 income         0.003     0       0
## 5 loyalty      168.       14.4     0
```

IPW and propensity scores

If you know the propensity scores $e(X_i)$, you can re-weight the observations to balance the covariates between the treatment and control groups

$$\tau_{IPW} = \mathbb{E}\left[\frac{d_i Y_i}{e(X_i)} - \frac{(1-d_i)Y_i}{1-e(X_i)}\right]$$

Casino ATE with IPW

```
response_model = glm(treatment ~ age + income + loyalty,
                     casino,family='binomial')
casino$e = predict(response_model,type='response')
casino = casino %>%
  mutate(ipw = treatment/e + (1-treatment)/(1-e))
```

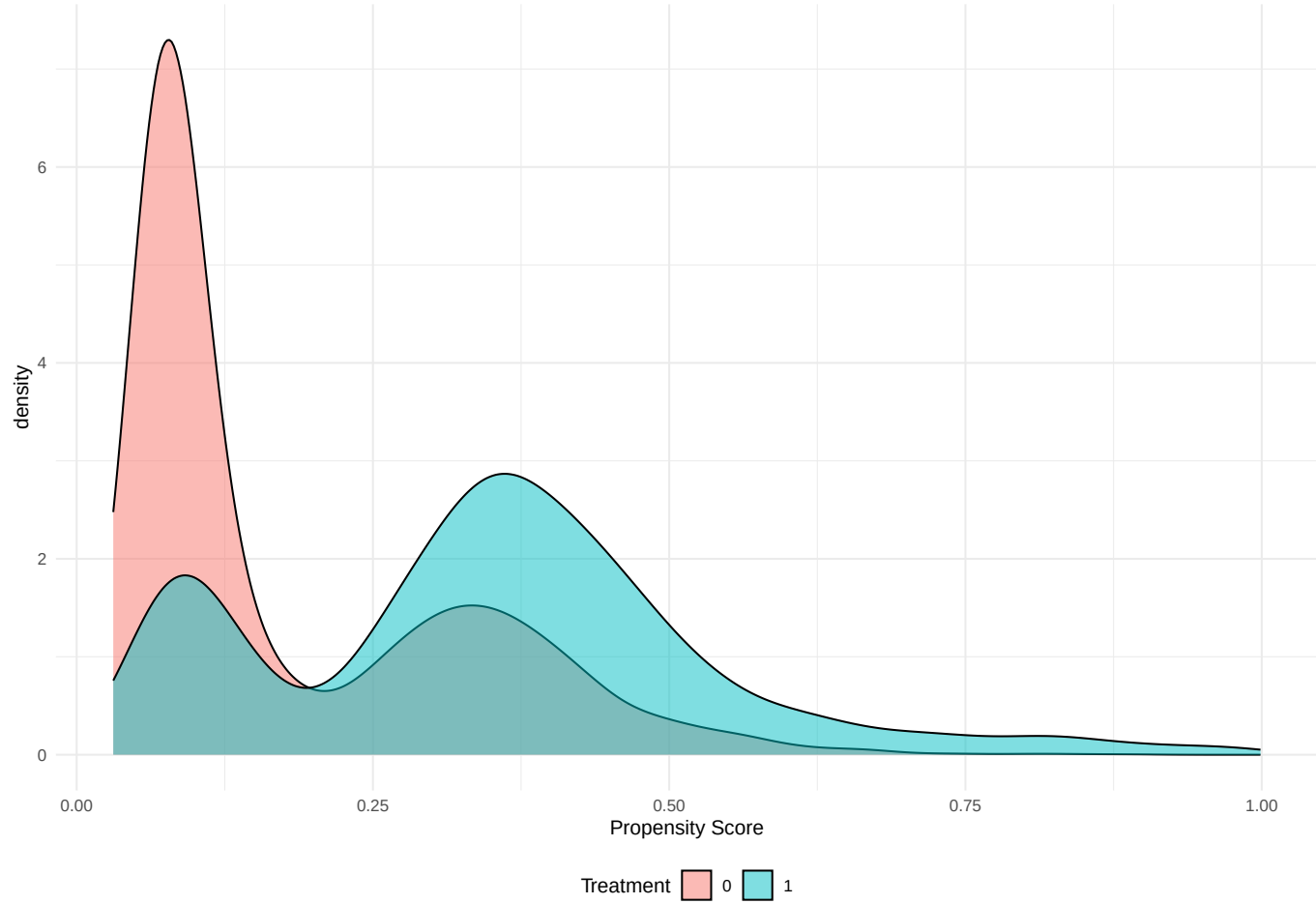
Used IPW as weights in `lm`

```
showlm(lm(spend ~ treatment, weights = ipw,casino))
```

```
## # A tibble: 2 x 4
##   term        estimate std.error p.value
##   <chr>          <dbl>     <dbl>   <dbl>
## 1 (Intercept)     509.      6.84       0
## 2 treatment       155.      9.66       0
```

Checking overlap assumption

When using propensity scores (without matching) make sure covariates are balanced



AIPW or Double-Robust Estimators

Essentially combines IPW with CATE

1) Estimate a response model to get *propensity scores* $e(X) = \mathbb{E}[d|X]$
2) Estimate an outcome model to get the *response function* $u^{(d)}(X) = \mathbb{E}[Y|d, X]$

This is called "doubly robust" because the final estimate is **unbiased** even if one of the two functions is misspecified

The AIPW estimator

This looks way more complicated than it is:

$$\hat{\tau} = \frac{1}{n}\sum_{i=1}^{n}\left(u^1(X_i) - u^0(X_i) + \frac{d}{e}(Y_i - u^1(X_i)) - \frac{1-d}{1-e}(Y_i - u^0(X_i))\right)$$

In R

```
outcome_model = lm(spend ~ treatment +
                     age + income + loyalty,casino)
u1 = predict(outcome_model,casino %>% mutate(treatment = 1)
u0 = predict(outcome_model,casino %>% mutate(treatment = 0)
y = casino$spend
e = casino$e
d = casino$treatment
mean(u1 - u0 + d/e*(y - u1) - (1-d)/(1-e)*(y - u0))
```

```
## [1] 152.6906
```

Or use grf

Notice how we now leave out the `W.hat` argument, this tells `grf` to estimate the propensity scores for you

```
library(grf)
X  = casino[,c('age','income','loyalty')]
cf = causal_forest(X,y,d)
average_treatment_effect(cf)
```

```
##  estimate   std.err
## 152.09302  13.91354
```

DoubleML

Basically AIPW with ML at both steps (the response and outcome model)

Relies on "partialling out" the treatment effect

Partialling out

If you want to isolate the treatment effect d on y in the presence of confounding variables X you can partial out the treatment effect.

Regress the *residuals* from `y ~ X` against the *residuals* of `d ~ X`.

Both of these yield the same coefficient on d:

```
lm(y ~ d + X)
lm(lm(y ~ X)$residuals ~ lm(d ~ X)$residuals)
```

Who cares?

DoubleML uses machine learning to predict both the treatment and the outcome, given the confounders

The residuals from these predictions represent the parts of the treatment and outcome that are independent of the confounders

This effectively "partials out" the influence of the confounders, removing the spurious correlation between the treatment and outcome, allowing for a more accurate estimate of the treatment effect

- Essentially, it removes the variation in the outcome that is explained by the confounders, so that the variation remaining is only due to the treatment

DoubleML steps (don't worry, a package will do all this)

1) Split the sample into two
2) Estimate the the response and outcome models (typically using random forests) both sets
3) Using the estimated models, predict out-of-sample and save residuals
4) Run a linear regression of the outcome residuals on the response residuals
5) Average these two coefficients – this is the DoubleML ATE

See .rmd file for `R` code

Using DoubleML Package

For Python/R:

https://docs.doubleml.org/stable/intro/intro.html#intro

```
library(DoubleML)
library(mlr3)
library(mlr3learners)
#mlr_learners
```

Step 1: Set up data

```
#set up data
dml_data = DoubleMLData$new(casino,
                  y_col = "spend",
                  d_cols = "treatment",
                  x_cols = c("age","income","loyalty"))
#or
y = casino$spend
X = as.matrix(casino %>% select(age,income,loyalty))
d = casino$treatment
dml_data = double_ml_data_from_matrix(X=X, y=y, d=d)
```

Step 2: Set learners

Here we are using a regression tree, but you can use whatever ML method you want

```
learner = lrn("regr.ranger",
              num.trees=500,
              max.depth=5,
              min.node.size=2)
```

```
## Warning: Package 'ranger' required but not installed for
```

```
ml_l = learner$clone() #clone for independent models
ml_m = learner$clone()
```

Step 3: Estimate

```
obj_dml_plr = DoubleMLPLR$new(dml_data,
                              ml_l = ml_l,
                              ml_m = ml_m)
obj_dml_plr$fit()
```

Step 4: Show results

```
print(obj_dml_plr)
```