Recitation 14: Instrumental and discontinuity

Seamus Wagner

2021-11-30

Wald estimator

We will go over some of the Wald Estimator functions from the demo 10. Obviously giving examples of which values satisfy which conditions is not ideal, so we will be discussing what happens and how certain conditions are violated and how to detect them instead.

```
rm(list=ls())
library(data.table)
set.seed(6544568)
confounded_dt <- function(
n_obs,
p_U = .5,
p_D_0 = .5,
p_D_1 = .5,
p_Y_00 = .5,
p_Y_10 = .5,
p_Y_01 = .5,
p_Y_11 = .5
){
data <- data.table(
  U = rbinom(n_obs, 1, p_U),
  D0 = rbinom(n_obs, 1, p_D_0),
  D1 = rbinom(n_obs, 1, p_D_1),
  Y00 = rbinom(n_obs, 1, p_Y_00),
  Y10 = rbinom(n_obs, 1, p_Y_10),
  Y01 = rbinom(n_obs, 1, p_Y_01),
  Y11 = rbinom(n_obs, 1, p_Y_11)
)
data[, D := U * D1 + (1 - U) * D0]
data[, Y := U * D * Y11 + (1 - U) * D * Y01 +
            U * (1 - D) * Y10 + (1 - U) * (1 - D) * Y00]
population_ate <-
  p_U * (p_Y_11 - p_Y_10) + (1 - p_U) * (p_Y_01 - p_Y_00)
diff_in_means <- data[, sum(D * Y) / sum(D) - sum((1 - D) * Y) / sum(1 - D)]

dt <- data.table(population_ate = population_ate,
     diff_in_means = diff_in_means)
dt
}

confounded_dt(
```

```
n_obs = 10000,
p_U = .5,
p_D_1 = .5, p_D_0 = .5,
p_Y_11 = .65, p_Y_01 = .5,
p_Y_10 = .4, p_Y_00 = .5)
```

```
##    population_ate diff_in_means
## 1:          0.125     0.1160239
```

Just the base function from the demo with some different values.

```
simulate_iv_data <- function(
  n_obs,
  p_U = .5,      # base (unobservable)
  p_Z = .5,      # base (encouragement)
  p_D_10 = .25,  # if Z = 1 & U = 0
  p_D_11 = .8,   # if Z = 1 & U = 1
  p_D_00 = .2,   # if Z = 0 & U = 0
  p_D_01 = .7,   # if Z = 0 & U = 1
  p_Y_100 = .4, # if Z = 1 & U = 0 & D = 0
  p_Y_110 = .4, # if Z = 1 & U = 1 & D = 0
  p_Y_101 = .6, # if Z = 1 & U = 0 & D = 1
  p_Y_111 = .6, # if Z = 1 & U = 1 & D = 1
  p_Y_000 = .4, # if Z = 0 & U = 0 & D = 0
  p_Y_010 = .4, # if Z = 0 & U = 1 & D = 0
  p_Y_001 = .6, # if Z = 0 & U = 0 & D = 1
  p_Y_011 = .6  # if Z = 0 & U = 1 & D = 1
) {
  data <- data.table(
    U = rbinom(n_obs, 1, p_U), # base
    Z = rbinom(n_obs, 1, p_Z), # base
    D10 = rbinom(n_obs, 1, p_D_10),
    D11 = rbinom(n_obs, 1, p_D_11),
    D00 = rbinom(n_obs, 1, p_D_00),
    D01 = rbinom(n_obs, 1, p_D_01),
    Y100 = rbinom(n_obs, 1, p_Y_100),
    Y110 = rbinom(n_obs, 1, p_Y_110),
    Y101 = rbinom(n_obs, 1, p_Y_101),
    Y111 = rbinom(n_obs, 1, p_Y_111),
    Y000 = rbinom(n_obs, 1, p_Y_000),
    Y010 = rbinom(n_obs, 1, p_Y_010),
    Y001 = rbinom(n_obs, 1, p_Y_001),
    Y011 = rbinom(n_obs, 1, p_Y_011)
  )
  data[U == 1, `:=`(
    complier = 1 * (D11 == 1 & D01 == 0),
    always_taker = 1 * (D11 == 1 & D01 == 1),
    never_taker = 1 * (D11 == 0 & D01 == 0),
    defier = 1 * (D11 == 0 & D01 == 1)
  )]
  data[U == 0, `:=`(
    complier = 1 * (D10 == 1 & D00 == 0),
    always_taker = 1 * (D10 == 1 & D00 == 1),
```

```
  never_taker = 1 * (D10 == 0 & D00 == 0),
  defier = 1 * (D10 == 0 & D00 == 1)
)]
data[, D :=
    Z       * U       * D11 +
    Z       * (1 - U) * D10 +
    (1 - Z) * U       * D01 +
    (1 - Z) * (1 - U) * D00]
data[, Y :=
    Z       * U       * D       * Y111 +
    Z       * U       * (1 - D) * Y110 +
    Z       * (1 - U) * D       * Y101 +
    Z       * (1 - U) * (1 - D) * Y100 +
    (1 - Z) * U       * D       * Y011 +
    (1 - Z) * U       * (1 - D) * Y010 +
    (1 - Z) * (1 - U) * D       * Y001 +
    (1 - Z) * (1 - U) * (1 - D) * Y000]
population_ate <-
  p_Z       * p_U       * (p_Y_111 - p_Y_100) +
  p_Z       * (1 - p_U) * (p_Y_101 - p_Y_100) +
  (1 - p_Z) * p_U       * (p_Y_011 - p_Y_010) +
  (1 - p_Z) * (1 - p_U) * (p_Y_001 - p_Y_000)
freq_compliers <- data[, mean(complier)]
freq_defiers <- data[, mean(defier)]
freq_always_takers <- data[, mean(always_taker)]
freq_never_takers <- data[, mean(never_taker)]
sample_cate <- data[complier == 1,
  sum(Z * U * (Y111 - Y110)) / sum(Z * U) +
    sum(Z * (1 - U) * (Y101 - Y100)) / sum(Z * (1 - U)) +
    sum((1 - Z) * U * (Y011 - Y010)) / sum((1 - Z) * U) +
    sum((1 - Z) * (1 - U) * (Y001 - Y000)) / sum((1 - Z) * (1 - U))]
diff_in_means <- data[,
  sum(D * Y) / sum(D) - sum((1 - D) * Y) / sum(1 - D)]
intent_to_treat <- data[,
  sum(Z * Y) / sum(Z) - sum((1 - Z) * Y) / sum(1 - Z)]
compliance_effect <- data[,
  sum(Z * D) / sum(Z) - sum((1 - Z) * D) / sum(1 - Z)]
wald <- intent_to_treat / compliance_effect
# assumption checks
# holding U,D constant, no change in p_Y with change in Z
exclusion_restriction <- ifelse(
  p_Y_111 == p_Y_011 & # U = 1 & D = 1
    p_Y_110 == p_Y_010 & # U = 1 & D = 0
    p_Y_101 == p_Y_001 & # U = 0 & D = 1
    p_Y_100 == p_Y_000,  # U = 0 & D = 0
  "satisfied", "violated")
monotonicity <- ifelse(
  p_D_11 > p_D_01 &
    p_D_10 > p_D_00,
  "satisfied", "violated")

list(
  exclusion_restriction = exclusion_restriction,
```

```
    monotonicity = monotonicity,
    freq_compliers = freq_compliers,
    freq_defiers = freq_defiers,
    freq_always_takers = freq_always_takers,
    freq_never_takers = freq_never_takers,
    population_ate = population_ate,
    diff_in_means = diff_in_means,
    sample_cate = sample_cate,
    intent_to_treat = intent_to_treat,
    compliance_effect = compliance_effect,
    wald = wald
  )
}
```

Quick example of what can happen with varying n_obs values. We are not analyzing the rest of the outputs here, though they are still important.

```
set.seed(2785103)
iv10 <- simulate_iv_data(10)
iv100 <- simulate_iv_data(100)
iv1000 <- simulate_iv_data(1000)
iv10000 <- simulate_iv_data(10000)
iv10$wald
```

```
## [1] -3
```

```
iv100$wald
```

```
## [1] -15.33333
```

```
iv1000$wald
```

```
## [1] 0.3093845
```

```
iv10000$wald
```

```
## [1] 0.2725308
```

Ensemble from previous year

```
rm(list=ls())
library(SuperLearner)
load("C:/Users/spw51/Downloads/c2c (1).Rdata")

# useful for which models you can use. Some require extra packages to work (kernlab, arm, etc)
# ksvm is kernel support vector machine, which is why I have kernlan loaded. I don't end up using it
listWrappers()
```

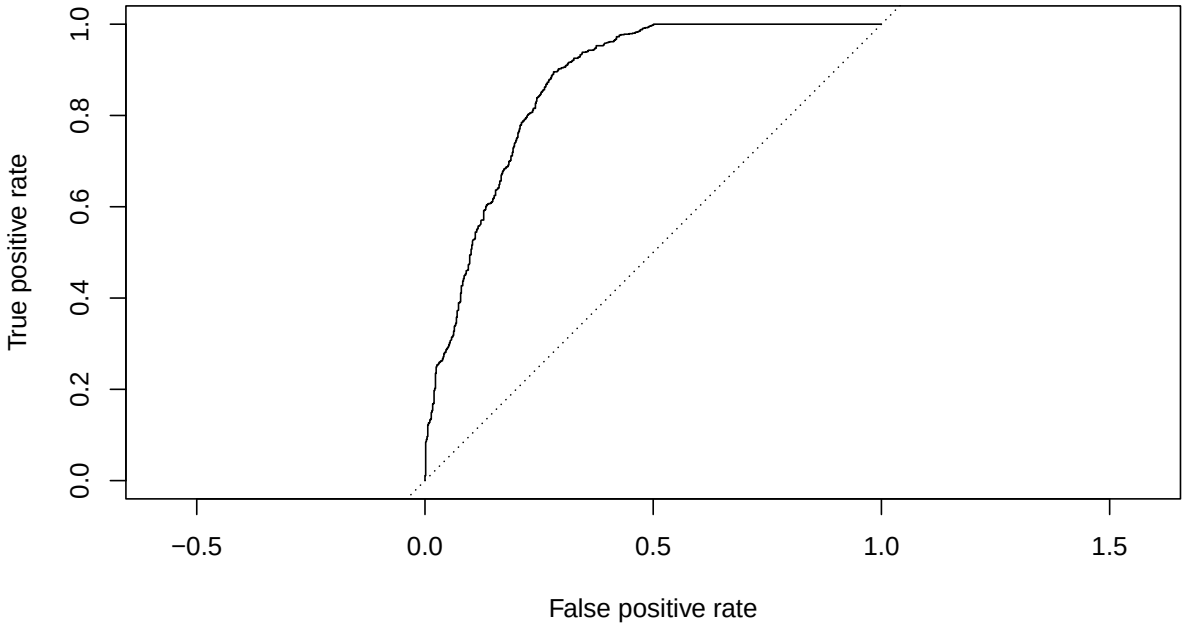
```
##  [1] "SL.bartMachine"      "SL.bayesglm"         "SL.biglasso"
##  [4] "SL.caret"            "SL.caret.rpart"      "SL.cforest"
##  [7] "SL.earth"            "SL.extraTrees"       "SL.gam"
## [10] "SL.gbm"              "SL.glm"              "SL.glm.interaction"
## [13] "SL.glmnet"           "SL.ipredbagg"        "SL.kernelKnn"
## [16] "SL.knn"              "SL.ksvm"             "SL.lda"
## [19] "SL.leekasso"         "SL.lm"               "SL.loess"
## [22] "SL.logreg"           "SL.mean"             "SL.nnet"
## [25] "SL.nnls"             "SL.polymars"         "SL.qda"
## [28] "SL.randomForest"     "SL.ranger"           "SL.ridge"
## [31] "SL.rpart"            "SL.rpartPrune"       "SL.speedglm"
## [34] "SL.speedlm"          "SL.step"             "SL.step.forward"
## [37] "SL.step.interaction" "SL.stepAIC"          "SL.svm"
## [40] "SL.template"         "SL.xgboost"
## [1] "All"
## [1] "screen.corP"           "screen.corRank"        "screen.glmnet"
## [4] "screen.randomForest"   "screen.SIS"            "screen.template"
## [7] "screen.ttest"          "write.screen.template"
```

```
mean(data$Y[data$D==1], na.rm = T) - mean(data$Y[data$D==0], na.rm = T)
```

```
## [1] 0.03605076
```

```
Y <- data$responder # SuperLearner wants a separate X and Y as input
X <- data[, -c("Y", "responder")]
?SuperLearner
sl <- SuperLearner(
Y, X, newX = X, family = binomial(), #binomial as we want a logit in this case since our outcome is res
SL.library = "SL.glm", #the type of model to use
method = "method.AUC", #the type of method to evaluate the model
verbose = FALSE,
cvControl = SuperLearner.CV.control( #cvcontrol is for control in cross valdiation, v is number of fold
V = 10L,
stratifyCV = FALSE,
shuffle = TRUE,
validRows = NULL))
```

```
plot_ROC_curve <- function(yhat, y) {
print(plot(performance( #performance and is from ROCR is a predicition evaluation command, it takes yha
prediction(yhat, y),
measure="tpr", x.measure="fpr" # tpr is true positives and fpr is false positives. Check the details se
), asp = 1, xlim = c(0, 1), ylim = c(0, 1)))
abline(0, 1, lty = 3)
}
plot_ROC_curve(sl$SL.predict, Y)
```




```
## NULL
```

```
calc_AUROC <- function(predictions, labels) {
performance(
prediction(predictions, labels),
measure = "auc")@y.values[[1]] #auc is area under ROC curve. ie, the amount of area beneath the plotted
}
?performance
```

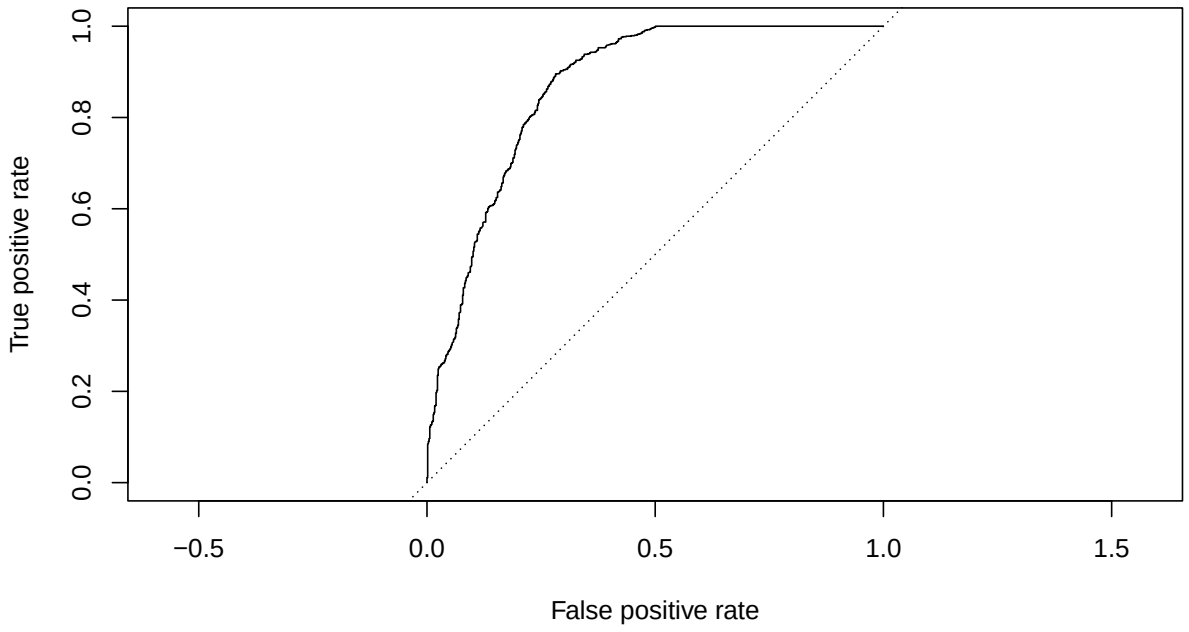
```
library(kernlab)
sl.2 <- SuperLearner(
Y, X, newX = X, family = binomial(),
SL.library = c("SL.glm", "SL.glmnet", "SL.mean", "SL.randomForest"),
method = "method.AUC",
verbose = FALSE,
cvControl = SuperLearner.CV.control(
V = 10L,
stratifyCV = FALSE,
shuffle = TRUE,
validRows = NULL))
```

```
sl.2
```

```
##
## Call:
## SuperLearner(Y = Y, X = X, newX = X, family = binomial(), SL.library = c("SL.glm",
##     "SL.glmnet", "SL.mean", "SL.randomForest"), method = "method.AUC", verbose = FALSE,
```

```
##     cvControl = SuperLearner.CV.control(V = 10L, stratifyCV = FALSE, shuffle = TRUE,
##         validRows = NULL))
##
##
##                           Risk      Coef
## SL.glm_All           0.1762565 0.2499448
## SL.glmnet_All        0.1748856 0.2542939
## SL.mean_All          0.5339966 0.2481737
## SL.randomForest_All 0.1722599 0.2475876
```

```r
plot_ROC_curve <- function(yhat, y) {
  print(plot(
    performance(
      prediction(yhat, y),
      measure = "tpr",
      x.measure = "fpr"
    ),
    asp = 1,
    xlim = c(0, 1),
    ylim = c(0, 1)
  ))
  abline(0, 1, lty = 3)
}
plot_ROC_curve(sl$SL.predict, Y)
```




```
## NULL
```

```r
calc_AUROC <- function(predictions, labels) {
performance(
prediction(predictions, labels),
measure = "auc")@y.values[[1]]
}
calc_AUROC(sl$SL.predict, Y)
```

```
## [1] 0.8673058
```