DoubleML Tutorial

Zachary Goldstein

2023-05-12

Introduction

This is a tutorial for how to use the DoubleML package in R. The package implements Double/Debiased Machine Learning for estimating causal effects.

Load Libraries

```
library(DoubleML)
library(tidyverse)
library(causaldata)
library(mlr3)
library(mlr3learners)
library(data.table)
set.seed(40601)
```

Example Data

For this tutorial, we'll be using data from the Current Population Survey.

The data lets us study the effect of participation in a job-training program on future wages.

The data is observational, participation was not randomized and we can not assume that program participation is independent of the potential outcomes.

Besides the main treatment and outcome variables, we have data on some covariates that we hypothesize to be related to participation and wages, including pre-program wages, race, age, educational attainment, and marital status.

For the purposes of this tutorial, we are going to assume that the observed covariates are sufficient to control for confounding. (comparisons with randomized experiment data have shown this may not actually be the case)

```
# Combine the randomized experiment data with the control-group-only observational data
df = cps_mixtape %>% bind_rows(nsw_mixtape)
df %>% head()
```

```
## # A tibble: 6 x 11
##   data_id treat   age  educ black  hisp  marr nodegree   re74   re75   re78
##   <chr>   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>    <dbl>  <dbl>  <dbl>  <dbl>
## 1 CPS1        0    45    11     0     0     1        1 21517. 25244. 25565.
## 2 CPS1        0    21    14     0     0     0        0  3176.  5853. 13496.
```

```
## 3 CPS1        0    38    12     0     0     1        0 23039. 25131. 25565.
## 4 CPS1        0    48     6     0     0     1        1 24994. 25244. 25565.
## 5 CPS1        0    18     8     0     0     1        1  1669. 10728.  9861.
## 6 CPS1        0    22    11     0     0     1        1 16366. 18449. 25565.
```

Feature engineering

```
df = df %>%
  mutate(age_squared = age**2,
         age_under_35 = as.numeric(age<35),
         black_or_hispanic = as.numeric(black==1|hisp==1),
         # Average the income from the 2 years of data pre-program
         avg_pre_income = re74+re75/2,
         # Binary indicator for 12+ years of educational attainment
         educ_12_plus = as.numeric(educ>=12),
         )
```

Partially Linear Model

Theoretical Overview

Let's start with a relatively simple option for Double Machine Learning, the partially linear model. A partially linear model assumes a linear relationship between the treatment and outcome, but makes no such assumptions about the relationships between the covariates and the outcome. We can use non-parametric machine learning methods of our choice to model the relationship between the covariates and the outcome.

The model is as follows:

$$Y = D\theta_0 + g_0(X) + \zeta$$
$$D = m_0(X) + V$$
$$E[\zeta|D, X] = 0$$
$$E[V|X] = 0$$

Y is the continuous outcome variable **re78**, real earnings in 1978. D is the binary treatment variable **treat** corresponding to job training program participation. θ_0 is the effect of the treatment on the outcome. g_0 is a possibly non-linear function of the covariates X that models their relationship with the outcome. m is the propensity score function that predictions the probability of program participation given the covariates X. and V are random error terms.

DoubleMLPLR function

The DoubleML R package implementation for the partially linear regression model is the function DoubleMLPLR. To create a new model, we use the "new" method, as follows:

```
l_rf = lrn("regr.ranger", max.depth = 5, min.node.size = 1)
m_rf = lrn("classif.ranger", max.depth = 3, min.node.size = 6)

non_covariates = c("data_id","treat","re78")
covariates = setdiff(names(df),non_covariates)
df_dml = df %>%
```

```
  data.table %>%
  double_ml_data_from_data_frame(y_col = "re78", d_cols = "treat",x_cols = covariates)

# Create a wrapper so I can set my own default parameters
# Then throughout the tutorial, I can make it clear which arguments deviate
# from the default in each example
DoubleMLPLR_new_wrapper = function(data = df_dml,
  ml_l = l_rf,
  ml_m = m_rf,
  ml_g = NULL,
  n_folds = 5,
  n_rep = 1,
  score = "partialling out",
  dml_procedure = "dml2",
  draw_sample_splitting = TRUE,
  apply_cross_fitting = TRUE){
  model = DoubleMLPLR$new(
   data=data,ml_l=ml_l,ml_m=ml_m,ml_g=ml_g,n_folds=n_folds,n_rep=n_rep,
   score=score,dml_procedure=dml_procedure,
   draw_sample_splitting=draw_sample_splitting,
   apply_cross_fitting=apply_cross_fitting)
}

# Start out matching default parameters exactly
plm_ref = DoubleMLPLR_new_wrapper()

plm_ref$fit()
```

Printing the model outputs a summary of the variables and methodological parameters. Then it prints out a summary of the results, including the point estimate, standard error, test statistic, and p-value for the average treatment effect.

```
plm_ref
```

```
## ================= DoubleMLPLR Object ==================
##
##
## ------------------ Data summary      ------------------
## Outcome variable: re78
## Treatment variable(s): treat
## Covariates: age, educ, black, hisp, marr, nodegree, re74, re75, age_squared, age_under_35, black_or_h
## Instrument(s):
## No. Observations: 16437
##
## ------------------ Score & algorithm ------------------
## Score function: partialling out
## DML algorithm: dml2
##
## ------------------ Machine learner   ------------------
## ml_l: regr.ranger
## ml_m: classif.ranger
##
## ------------------ Resampling        ------------------
## No. folds: 5
```

```
## No. repeated sample splits: 1
## Apply cross-fitting: TRUE
##
## ------------------ Fit summary       ------------------
##  Estimates and significance testing of the effect of target variables
##       Estimate. Std. Error t value Pr(>|t|)
## treat     992.5      639.3   1.552    0.121
```

Score

The score is which formula we use for estimating θ_0, the effect of the treatment on the outcome. The package gives us two options for the partially linear model `score` argument, the "partialling out" score and the "IV-type" score. In the former, we set `score='partialling out'` and set learners `ml_l` and `ml_m`. In the latter, we set `score='IV-type` and set learners `ml_l`, `ml_m`, and `ml_g`.

The partialling out approach uses the following formula in which $\hat{l_0}(X_i) = E(Y|X)$:

$$\check{\theta}_0 = \left(\frac{1}{n} \sum_{i \in I} \hat{V}_i \hat{V}_i \right)^{-1} \frac{1}{n} \sum_{i \in I} \hat{V}_i (Y_i - \hat{\ell}_0(X_i))$$

And the IV-type uses the following formula:

$$\check{\theta}_0 = \left(\frac{1}{n} \sum_{i \in I} \hat{V}_i D_i \right)^{-1} \frac{1}{n} \sum_{i \in I} \hat{V}_i (Y_i - \hat{g}_0(X_i))$$

The partialling-out score partials out the associations between: 1) the covariates and the treatment and between 2) the outcome and the covariates given treatment.

This is a form of orthogonalization, which is used to remove regularization bias.

The "IV-type" score is not intended for instrumental variables problems in which we use a valid instrument to estimate a local average treatment effect even though the potential outcomes can't be assumed to be independent of the treatment given the observed covariates. DoubleML supports instrumental variables estimation, but that's done with a separate function: `DoubleMLPLIV`.

The "IV-type" is still intended for a selection-on-observables problem in which all confounding variables are observed. It is called "IV-type" because the scoring procedure resembles that of an instrumental variables estimation procedure like two stage least squares.

According to Chernozhukov and coauthors' 2018 paper, the two scores are "first-order equivalent." The DoubleML package user guide says that a minor advantage of the partialling out estimator is that the nuisance parameters `l` and `m` can both be directly estimated using machine learning, whereas `g` (which is needed for the IV-type score) cannot.

```
plm_iv_type = DoubleMLPLR_new_wrapper(score = "IV-type",
                                      ml_g = l_rf$clone())
plm_iv_type$fit()
```

```
plm_iv_type
```

```
## ================= DoubleMLPLR Object ==================
##
##
## ------------------ Data summary      ------------------
## Outcome variable: re78
```

```
## Treatment variable(s): treat
## Covariates: age, educ, black, hisp, marr, nodegree, re74, re75, age_squared, age_under_35, black_or_
## Instrument(s):
## No. Observations: 16437
##
## ------------------ Score & algorithm ------------------
## Score function: IV-type
## DML algorithm: dml2
##
## ------------------ Machine learner   ------------------
## ml_l: regr.ranger
## ml_m: classif.ranger
## ml_g: regr.ranger
##
## ------------------ Resampling        ------------------
## No. folds: 5
## No. repeated sample splits: 1
## Apply cross-fitting: TRUE
##
## ------------------ Fit summary       ------------------
##  Estimates and significance testing of the effect of target variables
##       Estimate. Std. Error t value Pr(>|t|)
## treat     970.9      611.7   1.587    0.112
```

Learners

The DoubleML package uses the mlr3 R package which supports many commonly used supervised learning algorithms, including generalized linear models with regularization penalties, k-nearest neighbors, naive bayes, neural networks, random forest, support vector machines, and gradient boosting.

You could also select an ensemble of different algorithms, or try multiple options and select the one with the best performance.

The documentation shows regression algorithms are used for estimating l, m and g. However, since m is a propensity score, it predicts a categorical outcome, so I would imagine it is sensible to select a classification algorithm for it, as I have done in the default example in this tutorial, where I selected random forest of regression trees for estimating the conditional mean function `l` and a random forest of classification trees for estimating the propensity score `m`.

DML Procedure

The DML package gives us two options for the `dml_procedure` argument, "dml1" and "dml2". This lets us choose between two slightly different algorithms.

"dml1" estimates the causal parameters first on each of k folds of the data separately, and then averages them together for an estimate of the causal parameter.

In the following equations: N is the sample size of the data. K is the number of folds (and k is a single fold) n is the sample size of a single fold (N/K) I is a single fold. ψ is the score function.

$$\frac{1}{n}\sum_{i\in I_k}\psi(W_i;\check{\theta}_{0,k},\hat{\eta}_{0,k})=0.$$

$$\tilde{\theta}_0 = \frac{1}{K} \sum_{k=1}^{K} \check{\theta}_{0,k}.$$

"dml2" estimates the causal parameters in a single step by solving an equation involving a double summation, first over the folds, and then over the entries within each fold.

$$\frac{1}{N} \sum_{k=1}^{K} \sum_{i \in I_k} \psi(W_i; \tilde{\theta}_0, \hat{\eta}_{0,k}) = 0.$$

According to the paper, DML2 is generally recommended over DML1, however there are some problems (e.g. ATE estimation in the interactive model) where either approach yields the same result. DML2 is generally recommended because the pooled empirical Jacobian is more stable than the separate empirical Jacobians. They also note that DML2 may behave better in small samples.

Since `dml2` is the default, here I'll show a version using `dml1`.

```
plm_dml1 = DoubleMLPLR_new_wrapper(dml_procedure = "dml1")
plm_dml1$fit()
```

```
plm_dml1
```

```
## ================= DoubleMLPLR Object ==================
##
##
## ------------------ Data summary      ------------------
## Outcome variable: re78
## Treatment variable(s): treat
## Covariates: age, educ, black, hisp, marr, nodegree, re74, re75, age_squared, age_under_35, black_or_
## Instrument(s):
## No. Observations: 16437
##
## ------------------ Score & algorithm ------------------
## Score function: partialling out
## DML algorithm: dml1
##
## ------------------ Machine learner   ------------------
## ml_l: regr.ranger
## ml_m: classif.ranger
##
## ------------------ Resampling        ------------------
## No. folds: 5
## No. repeated sample splits: 1
## Apply cross-fitting: TRUE
##
## ------------------ Fit summary       ------------------
##  Estimates and significance testing of the effect of target variables
##       Estimate. Std. Error t value Pr(>|t|)
## treat      1003        643    1.56    0.119
```

Cross Fitting

To avoid overfitting bias, Double Machine Learning splits the sample into subsets. The g and m models are estimated using one subset, while the scoring estimation is performed with the other subset. This is done in the `DoubleMLPLR$new` function by setting `draw_sample_splitting` to `TRUE`. It is always a good idea to do

sample splitting for DML, so if you set this argument to `FALSE`, you'll want to do sample splitting separately using a more customized approach with the `set_sample_splitting` method.

However, a weakness of sample-splitting is the loss of statistical power since each part of the estimation is only being done with half the sample size. To maintain the ability to evaluate on a held-out set without a loss of power, the authors propose "cross-fitting", in which sample-splitting is done twice, with each half of the data getting a turn to do each step. Then, the results are averaged together. You can use cross-fitting by leaving the `apply_cross_fitting` argument at its default value of `TRUE`.

However, having two equally-sized folds, one for nuisance parameter estimation and one for scoring is not the only option. The authors propose another approach in which the data is split into k equally-sized parts, such as k=5, and each time all but one of the folds is used for nuisance parameter estimation, while just one is used for scoring. This is implemented in the DML package by setting the `n_folds` argument.

The authors find that standard errors of estimates are generally lower when using 5-fold cross-fitting than 2-fold. The estimates are more precise when most of the data is used to estimate the nuisance parameters than when the data is split evenly between the two parts of the procedure. However, it is not as straightforward a relationship as "the more folds, the more precise the estimates". The optimal number of folds seems to depend on a number of factors, and is not fully understood.

Repeated Cross-Fitting By default, cross-fitting only happens once. But if you change `n_rep` to a value higher than 1, cross-fitting will be repeated `n_rep` times.

The advantage of repeated cross-fitting is that it can reduce sensitivity to particular sample splits. You can then calculate the point estimate as the median of the point estimates across all the different splits. You can also use the repeated cross-fitting to estimate a standard error. The procedure for this is described in Zivich et al (2021).

The advantage of just doing a single round of cross-fitting is it reduces the runtime needed to carry out the procedure, as well as simplifying the cross-fitting process.

Inference

Bootstrapping

DML supports bootstrapping of coefficients and t-statistics using the `bootstrap()` method and the `boot_coef` and `boot_t_stat` attributes. It supports a "normal", "wild", and "Bayes" bootstrap method. The number of resamples is specified by the `n_rep_boot` argument.

For example:

```
boot_results = plm_ref$bootstrap(method='normal',n_rep_boot=500)
boot_results$boot_coef[1:10]
```

```
##  [1]  -927.69840   605.07403    91.21414   -42.19430    47.77625 -1457.12694
##  [7]  -450.99764 -1026.10615   591.60273   770.74100
```

```
boot_results$boot_t_stat[1:10]
```

```
##  [1] -1.45100404  0.94639040  0.14266715 -0.06599569  0.07472636 -2.27907807
##  [7] -0.70540103 -1.60492265  0.92532008  1.20550850
```

You can then use the bootstrapped results to estimate standard errors and confidence intervals.

Confidence Intervals

Here's a 95% confidence interval for the average treatment effect:

```
boot_results$confint(joint=F,level=.95)
```

```
##           2.5 %   97.5 %
## treat -260.6459 2245.557
```

It contains zero, suggesting the result is not statistically significant.

Beyond the PLR Model

The DML package also supports more complex models than the partially linear regression model.

Instrumental Variables

If you have an instrumental variable, that is, a variable that exogenously affects the treatment without having a separate effect on the outcome, you can use `DoubleMLPLIV` to model that.

Clustered Data

The DML package supports cluster-robust cross-fitting for clustered data using `DoubleMLClusterData`.

Interactive Regression

An interactive regression model using `DoubleMLIRM` does not assume that the treatment has a simple linear relationship with the outcome. It allows for interactions between the treatment term and the covariates.

The DML package also supports interactive regression models with instrumental variables using `DoubleMLIIVM`.

Citations

Bach, P., Chernozhukov, V., Kurz, M. S., and Spindler, M. (2021), DoubleML - An Object-Oriented Implementation of Double Machine Learning in R, arXiv:2103.09603.

Chernozhukov, V., Chetverikov, D., Demirer, M., Duflo, E., Hansen, C., Newey, W., & Robins, J. (2018). Double/debiased machine learning for treatment and structural parameters.

Zivich, P. N., & Breskin, A. (2021). Machine learning for causal inference: on the use of cross-fit estimators. Epidemiology (Cambridge, Mass.), 32(3), 393.