Chapter 14 Coding Homework

Nick Huntington-Klein

Updated 2024-06-05

Follow the below instructions and turn in both your code and results:

1. Load the `nsw_mixtape` data that can be found in the **causaldata** package associated with the book, or download it fromLoad the `dengue.csv` file provided to you, or from this site. Documentation on the variables is available through the package, or here.

Then, drop the `data_id` variable from the data.

Language-specific instructions:

- In R or Python, store the data set as `nsw`.
- In R, after loading the **tidyverse** you can drop using `select(-droppedvariable)`
- In Stata, it's `drop droppedvariable`
- In Python, after loading **pandas** it's 'nsw.drop('droppedvariable', axis = 1)

2. Let's see where we're at before we do any matching at all. `nsw_mixtape` is from an experiment (read that documentation!) so that should already put us in a pretty good place.

First, create a variable called `weight` in your data equal to 1 for all observations (weights that are all 1 will have no effect, but this will give us a clue as to how we could incorporate matching weights easily).

Second, write code that uses a set of given weights to estimate the effect of `treat` on `re78`, using `weight` as weights, and prints out a summary of the regression results. The easiest way to do this is probably weighted regression; see The Effect Section 13.4.1, but without any controls or predictors other than `treat`. **Keep in mind the standard errors on the estimate won't be quite right, since they won't account for the uncertainty in creating the weights.**

Third, write code that creates and prints out a weighted balance table for all variables across values of `treat`, using `weight` as weighted. See The Effect Section 14.6.3. Don't worry about getting a table with tests of significant differences for now; just the means.

Language-specific instructions:

- One easy way to get the balance table in R is with `sumtable()` in **vtable** by setting the `group` and `group.weight` options (and possibly `group.test`).
- One easy way to get the table in Stata is with `tabstat` using the `by()` option.
- One easy way to get the table in Python is with `.groupby.aggregate()`.

2b. Is there anything potentially concerning about the balance table, given that this is a randomized experiment where `treat` was randomly assigned?

3. Using all of the variables in the data except `treat` and `re78` as matching variables, perform 3-nearest-neighbor Mahalanobis distance matching with replacement and no caliper (The Effect 14.4.1) and calculate the post-matching average treatment on the treated effect of `treat` on `re78`.

Check the documentation of the function you use to be sure you're matching with replacement.

Language-specific notes:

- Due to a namespace conflict (the same name used for two different functions in different packages), loading the **Matching** package may make the `select()` function not work. So be sure to re-load the **tidyverse** again afterwards, or use `dplyr::select()` instead of `select()` to let R know which version of `select()` to use.
- The documentation isn't super clear, but Stata's `teffects nnmatch` does match with replacement by default.

4. Create a post-matching balance table showing balance for all the matching variables (you'll probably want to use the balance function designed to follow the matching function you used, from the same package). Write a sentence commenting on whether the balance looks good. You may have to read the documentation for the function you use to figure out what the results mean.

5. Switching over to propensity score matching, use the same matching variables as in Question 3 to estimate the propensity to be treated (with a logit regression), and then add the treatment propensity to the data set as a new variable called `propensity`. Trim the propensity score, setting to missing any values from 0 to .05 or from .95 to 1 (this is a different method than done in the chapter).

Be careful to get the predicted *probability of treatment* and not the predicted *index function*. You can check this by making sure the values are all between 0 and 1.

(also, note, your estimation shouldn't produce any propensities that end up actually getting trimmed, but write the code to do so anyway, just in case)

Language-specific notes:

- You can set values to missing based on their value using `variable = ifelse(condition, NA_real_, variable)` or `variable = case_when(condition ~ NA_real_, TRUE ~ variable)` inside a `mutate()` in R, `replace variable = . if condition` in Stata, or using `.loc[condition, 'variable'] = pd.NA` in Python.
- It's not necessary, but there are ways of avoiding having to type out every single variable name in each langauge in your regression:
- In R, if your data set contains only your outcome variable and the predictors, you can use the formula `y~.` and that will regress the variable `y` on all the other variables in the data
- In Stata, you can use `a-b` to refer to the full set of variables between `a` and `b`. If your variables are `order`ed properly, this can be the list of predictors
- In Python, you can use **statsmodels.api** instead of **statsmodels.formula.api** and use a matrix of your predictors (or just use the **CausalModel** approach with a matrix, which you're already using)

6. Create a new variable in the data called `ipw` with the inverse probability weight, and then estimate the treatment effect using those weights in a linear regression (keeping in mind the standard errors won't be quite right).

Note that the same tools you used to trim `propensity` conditional on its value can also be used to calculate `ipw` in one way for treated observations and in another way for untreated observations.

Language-specific notes:

- In Python, the trimming process may have converted your `ipw` variable into an "object" instead of a numeric. You may need to use `pd.to_numeric()` to fix the variable before using it as a weight. May as well fix `propensity` while you're at it.

7. Make a common support graph, overlaying the density of the `propensity` variable for treated observations on top of the density of the `propensity` variable for untreated observations. You may want to refer to this guide if you are not familiar with your language's graphing capabilities.

Write a line commenting on how the common support looks.

8. Use the prepackaged command for inverse probability weighting used in the chapter for your language to estimate the treatment effect. Don't apply a trim (as previously established, for this particular problem it doesn't do much).