Exercise 4: The generalized random forest

Kristian Urup Olesen Larsen[1]

[1]kuol@econ.ku.dk

March 13, 2020

Problem 4.1.1 - simulating some data

Our first task is to create a simulated dataset. We want to use simulated data because this allows us to observe the ground truth $\tau(x)$ which is otherwise unobserved in real data (recall our discussion of the potential outcomes framework last week). Let us walk through the DGP[1] one equation at a time.

$$T_i = U(0,1) > 0.5 \tag{1}$$

Equation (1) describes how treatment T_i is assigned. In this case we draw a random number from an uniform distribution between 0 and 1. If this value is above 0.5 we set $T_i = 1$, and otherwise $T_i = 0$.

$$Y_i(T_i = 0) = X_i\beta + \epsilon_i \tag{2}$$

This second equation describes how the *baseline outcome* is linearly related to X_i (which is just drawn from a random normal). Remember that X_i is a matrix, and β a vector, so all `N_FEATURES` variables influence the value of $Y_i(0)$.

$$\tau(x_i) = \begin{cases} \frac{10}{1+e^{-\gamma X_0}} + \nu_i & D_i = 0 \\ \nu_i & D_i = 1 \end{cases} \tag{3}$$

Equation (3) governs the treatment effect $\tau(x_i)$. Notice that while D_i is not defined in the math, the code generates it as a dummy which is randomly assigned. In the cases where $D_i = 1$ the treatment effect will just be a random number centered around 0. On the other hand if $D_i = 0$ the treatment effect depends directly on X_0 through a logistic function. In conclusion this DGP exhibits heterogeneity across D_i and X_0.

$$Y_i(T_i = 1) = Y_i(0) + \tau(x_i) \tag{4}$$

The final equation just states that the level of y, under treatment, is the baseline $Y(0)$ plus the treatment effect $\tau(x_i)$

Coding it up

There are only three lines we need to fill in here, first lets compute the treatment effect `Tau`. The code here is somewhat complicated, but let us walk through it.

```
Tau = 10*(1-D)/(1 + np.exp(-GAMMA*X[:,0])) + np.random.normal()
```

Ignore for a moment the random noise ν_i =`np.random.normal()` then the remaining expression is a fraction $\frac{10*(1-D_i)}{1+e^{-\gamma X_0}}$. Note that the numerator is 10 if $D_i = 0$ and 0 if $D_i = 1$. In the denominator we extract all rows (`:`) and the first column (`0`) of `X`, that is X_0 =`X[:,0]`. This is multiplied by γ and we subsequently compute $e^{-\gamma X_0}$.

Next let us tackle $Y_i(1)$ - this is easy once we have $\tau(x)$, simply compute

```
Y1 = Y0 + Tau
```

And finally we will need the observed y. One nice way to write this is as $y = Y_i(0) + T_i(Y_i(1) - Y_I(0))$ which is either $Y_i(0)$ or $Y_i(1)$ dependent on the value of T_i. In python this looks like

```
y = Y0 + T*(Y1 - Y0)
```

[1]data-generating process.

Problem 4.1.2 - Visualizing the dataset

In this problem you are asked to draw two figures, 1) a scatter plot of X_0 against $Y(1)$ and $Y(0)$ and 2) A plot of X_0 against the true treatment effect $\tau(x)$. Let us start out by looking at the final figure, figure 1. In panel **A** we see $Y(0)$ as blue dots, depending on the random values in β you will see a linear relation between X_0 and $Y(0)$. More interesting is the values of $Y(1)$, here about half lie on top of $Y(0)$ while the other half deviates significantly for large values of X_0. This of course, is a result of the way in which we have constructed $\tau(x)$, and you can see this in panel **B**; half of the observations have $\tau \approx 0$, while for the other half it follows a logistic curve.

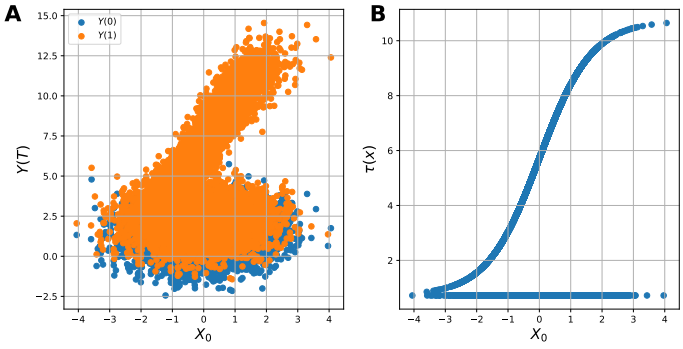

Fig 1. Simulated $y(0)$ and $Y(1)$ and the corresponding simulated treatment effect $\tau(x)$

Now let us look at the code that generated figure 1. The first line sets up a figure, and two "axes". The first two arguments set the number of subplot rows to 1 and the number of subplot columns to 2.

```
fig, ax = plt.subplots(1,2, figsize = (12,6))
```

Now let us work on panel **A**, first we plot the data as two scatterplots

```
ax[0].scatter(X[:,0], Y0, label = '$Y(0)$')
ax[0].scatter(X[:,0], Y1, label = '$Y(1)$')
```

Note that `ax` is a *list* of subplot axes. Thus `ax[0]` is the first subplot in the figure. To finish up panel **A** we set axis labels, add a legend and draw a grid on the figure.

```
ax[0].set_xlabel('$X_0$', fontsize = 16)
ax[0].set_ylabel('$Y(T)$', fontsize = 16)
ax[0].legend()
ax[0].grid(True)
```

The approach for the second subplot is similar, first we draw the data in a scatter plot,

```
ax[1].scatter(X[:,0], Tau, label = '\tau(x)')
```

and then finish up the figure by adding text and a grid.

```
ax[1].set_xlabel('$X_0$', fontsize = 16)
ax[1].set_ylabel('$\\tau(x)$', fontsize = 16)
ax[1].grid(True)
```

To finish up the figure we will add the panel-names using `plt.text`, here the first two arguments are the relative x and y-position of the text, the third argument is of course the text itself.

```
fig.text(0.05,0.85, 'A',fontsize = 24, fontweight = 'bold')
fig.text(0.51,0.85, 'B',fontsize = 24, fontweight = 'bold')
```

Getting text positioned correctly can be a bit tricky in matplotlib, my best advice is to play around with the x and y values until you are happy with the result.

Finally a call to `plt.savefig('my_filename.pdf')` will save your figure on disk.

1 A simple linear regression