Exercise 5: The generalized random forest

Kristian Urup Olesen Larsen[1]

[1]kuol@econ.ku.dk

March 17, 2020

Problem 5.1.1 - simulating some data

[1] 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[2] 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(size = N_SAMPLES)
```

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
```

[1]**Note:** in the exercises, and in a previous version of this writeup the noise ν_i and ϵ_i were generated by calling `np.random.normal()`. This only draws one random number, and the correct code is of course `np.random.normal(size = N_SAMPLES)`

[2]data-generating process.

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 = Yo + T*(Y1 - Yo)
```

Problem 5.1.2 - 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.

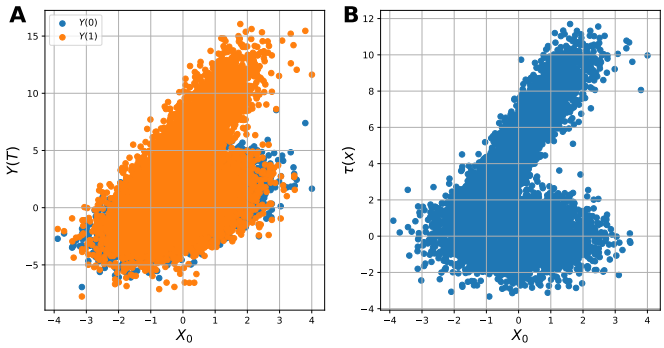

Fig 1. Simulated $y(0)$ and $Y(1)$ and the corresponding simulated treatment effect $\tau(x)$

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], Yo, 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.

Problem 5.1.3 - A simple linear regression

Now on to actually working with the data that we have simulated Linear regression is the workhorse of social science, and we will start by seeing how far we can get using only it. Remember, in reality we only observe X, y and T, and from this we want to estimate $\hat{\tau}(x)$ such that $\hat{\tau(x)} \approx \tau(x)$ with closer equality for larger samples. The regression we set of with is given by

$$y_i = \alpha + \delta T_i + \sum_k \beta_k X_{ik} + \epsilon_i \tag{5}$$

Notice that this equation matches the datagenerating process for $Y(0)$ when $T_i = 0$, with ϵ_i a joint error term $\nu_i + \eta_i$. however it does not match the datagenerating process when $T_i = 1$. In this case it predicts that y_i is simply increased by some constant δ, which we know is wrong.
Now is a good time to make predictions about the estimate $\hat{\delta}$ we will get. From panel **B** of figure 1 among the 50% following the logistic curve, the average treatment effect $\bar{\tau}_{D=0}$ is approximately 5. For the other 50% $\bar{\tau}_{D=1} \approx 0$. Thus on average, we probably have $\bar{\tau} \approx 2.5 \approx \hat{\delta}$. This number is going to vary slightly depending on the random sample of data. To be more precise, we can expect $\hat{\delta} \approx \frac{1}{N}\sum_{i=1}^{N} \tau_i$.

Coding it up

Coding up the linear regression begins by importing OLS from the statsmodels package. Note that statsmodels has two versions of OLS. Here we import the one that requires the regression to be specified as a *formula*, and the data to be a *dataframe*. Another version takes matrices X, y as inputs.

```
from statsmodels.formula.api import ols
```

Now let us collect the relevant data in a `pd.DataFrame`

```
df = pd.concat(
  [
      pd.DataFrame(X, columns=[f'X{i}' for i in range(N_FEATURES)]),
      pd.DataFrame(T, columns = ['T']),
      pd.DataFrame(y, columns = ['y'])
  ],
axis = 1)
```

Had we instead used `statsmodels.api.OLS` we would need to construct X and y matrices. The last step is to estimate the model and read the value of $\hat{\delta}$ to compare with our guess.

```
model = ols('y  ~ T + X0 + X1 + X2 + X3 + X4', df)
res = model.fit()
res.summary()
```

Indeed you should observe that $\hat{\delta} \approx \text{mean}(\tau)$.

Extentions

The above procedure highlights the strength of linear regression; it recovers the *average* treatment effect even when the true effects are highly hetereogenous. At the same time, this is of course also a limitation. Concluding that assigning $T_i = 1$ raises y by $\hat{\delta}$ would be wrong. When $D = 1$ it does not raise y at all, and when $D = 0$ it depends heavily on the value of X_0. I will not go into details with the OLS specification here, but to improve the model we would need to account for *i)* the interaction between D and T, and *ii)* the interaction between T and X_0. Of course realizing this in a real, unsimulated dataset would be close to impossible without some special knowledge of the datagenerating process.

1 Problem 5.1.4 - Moving into R

Next we need to save the data X, y, T, D to a .csv file. Since we already created a dataframe containing X, y and T in the previous problem we can simply add the D column to this dataframe.

```
df['D'] = D
```

Then save the file as a .csv using the .to_csv method of pd.DataFrame. Supplying index = False simply means we do not want to save an additional column containing the index of the data.

```
df.to_csv('simulated_data.csv', index = False)
```

Problem 5.1.5 - Loading the data in R

For this problem i assume that you have all managed to install R and Rstudio. After opening Rstudio begin by running the two following lines to install the packages we will need:

```
install.packages("tidyverse")
install.packages("grf")
```

Then, in Rstudio create a new R-script and name it something like *ex5_rsolution.R*. The first lines of this script will load the two packages we just installed

```
library(tidyverse)
library(grf)
```

In the next two lines we will adjusts the *working directory* to where we saved the data from python, and load the dataset into R.

```
setwd("C:/paht/to/your/data")
df <- read.csv("simulated_data.csv")
```

You should now see a new dataframe in the top right panel of Rstudio. Clicking on this will show you the dataset you just loaded.

Problem 5.1.6 - Extracting T as a matrix

This question should be straight forward, given the example code in the exercise set.

```
W <- df %>%
  select(T) %>%
  as.matrix()
```

Note that in the GRF package they use the notation W to signify the treatment assignment variable, whereas we used T above. To be consistent with the signature of grf::causal_forest we name the treatment-matrix W in R.

Problem 5.1.7 - Estimating a generalized random forest

This is another very simple problem, once you know what to do. Let us first look at the solution.

```
cf <- causal_forest(X, y, W)
```

Thats it. After running this line you can run `tau <- predict(cf, X)` to get a dataframe of predicted treatment effects.

The difficulty in this problem is probably figuring out that `grf::causal_forest` exists. One option is look in the grf *vignette* (Rs version of documentation files). Searching for "causal" quickly brings up the function. Another is to look at the usage examples in readme.md on the GRF repository.

Problem 5.1.8 - Visualizing the fit of $\hat{\tau}$

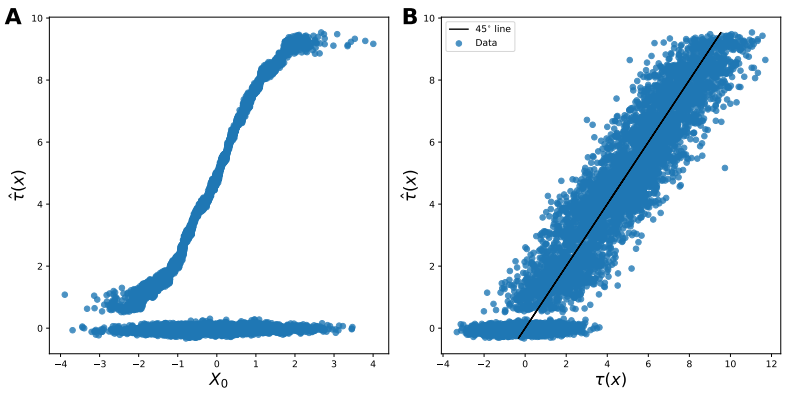

Fig 2. Fit of estimated treatment effects $\hat{\tau}$.

In this final problem we move back into python to graphically evaluate the performance of our causal forest model. First, load the estimated treatment effects from R into a `pd.DataFrame`

```
ites = pd.read_csv('individual_treatment_effects.csv')\
         .drop('Unnamed: 0', axis = 1)
```

(note the \ works as a *line-continuation* character in python. It can be used to split long single-line expressions across multiple lines in the editor). Drawing the figure is almost identical to how we did it in problem 5.1.2, first we set up a figure object with two associated axes

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

Then draw the first plot showing $\hat{\tau}$ against X_0

```
ax[0].scatter(X[:,0], ites, alpha = 0.8)
ax[0].set_xlabel('$X_0$', fontsize = 18)
ax[0].set_ylabel('$\hat{\\tau}(x)$', fontsize = 18)
```

And the second panel showing $\tau(x)$ against $\hat{\tau}(x)$

```
ax[1].scatter(Tau, ites, alpha = 0.8)
ax[1].set_xlabel('$\\tau(x)$', fontsize = 18)
ax[1].set_ylabel('$\hat{\\tau}(x)$', fontsize = 18)
```

Finally call `fig.tight_layout` to autoadjust the padding between the two subplots and add some panel identifiers.

```
fig.tight_layout()
fig.text(0.01,0.95, 'A', fontsize = 24, fontweight = 'bold')
fig.text(0.51,0.95, 'B', fontsize = 24, fontweight = 'bold')
```

Having done this you should end up with a figure like figure 2. Panel **A** looks similar to the plot we created in figure 1, except the true treatment effects are significantly more noisy. This makes sense as we have added a random component ν_i to the treatment effects. Since ν_i is by construction unpredictable, the best we can do is matching the mean treatment effect conditional on X and D. Looking at panel **B** we match this conditional mean very precisely, as the deviations from the 45° line look very much like normally distributed random noise in $\tau(x)$.